PRIME NUMBERS AND ENCRYPTION

Jorge L. Anicama[1]

December, 2007

Abstract

In this article we will deal with the prime numbers and its current use in encryption algorithms. Encryption algorithms make possible the exchange of sensible data in internet, such as bank transactions, email correspondence and other internet transactions where privacy is important.

Keywords: *Prime Numbers, Modular Arithmetic, RSA, Encryption*

1. *Oracle Corporation, USA*

1 Introduction

Encryption is the process of transforming information to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key.

Modern encryption algorithms depends heavily on number theory,with primality testing, factoring, discrete logarithms and elliptic curves being perhaps the most prominent subject areas. Encryption of electronic messages currently is applied in many areas of human life, such as the signature of digital documents, the implementation of electronic money, copyright protection, bank electronic transactions, ATM machines, Smart cards, etc. Because of these important applications I would like to explain in this article the mathematical foundation of one famous encryption algorithm : RSA algorithm. I assume the reader has some basic knowledge on number theory and computer science terminology, the rest is explained and proved here.

2 Divisibility and GCD Algorithm

Definition 2.1 *Let $a, b \in \mathbb{Z}$ with $a \neq 0$. We say a divides b, denoted by $a \mid b$, if $\exists\ c \in \mathbb{Z}$ such that $b = ac$. When a divides b, we say that a is a* divisor *(or* factor*) of b, and b is a* multiple *of a. If a does not divide b, we write $a \nmid b$. If $a \mid b$ and $0 < a < b$, then a is called a* proper divisor *of b.*

Theorem 2.1 *Let a,b and c be integers. Then*

- *if $a \mid b$ and $a \mid c$, then $a \mid (b + c)$.*
- *if $a \mid b$, then $a \mid bc$, for any integer c.*
- *if $a \mid b$ and $b \mid c$, then $a \mid c$.*

Theorem 2.2 (Euclid's Division Algorithm)
Let $a, b \in \mathbb{Z}$, $b \neq 0$. Then there exists unique determined $q, r \in \mathbb{Z}$ such that

$$a = q.b + r \quad \text{where} \quad 0 \leq r < |b| \tag{1}$$

The number q is called the quotient *and r is called the* remainder.

Proof: Consider the arithmetic progression

$$\cdots, -3b, -2b, -b, 0, b, 2b, 3b, \cdots$$

Then there must be an integer q such that

$$qb \leq a \leq (q+1)b$$

Let $a - qb = r$, then $a = bq + r$ with $0 \leq r < b$. To prove the uniqueness of q and r, suppose there is another pair q_1 and r_1 satisfying the same condition in equation (1), then

$$a = bq_1 + r_1, \quad 0 \leq r_1 < b$$

We first show that $r_1 = r$. For it not, we may presume that $r < r_1$, so that $0 < r_1 - r < b$, and then we see that $b(q - q_1) = r_1 - r$, and so $b \mid (r_1 - r)$, which is impossible. Hence, $r = r_1$, and also $q = q_1$. ■

Algorithm 2.1 (Euclid's Division Algorithm)

Input : $a, b \in \mathbb{Z}, \ b \neq 0$
Output : q, r *such that* $a = q \cdot b + r$ *and* $r = 0$ *or* $r < b$
Method :

$q := 0$
$r := a$
While $r \neq 0$ *and* $b < r$ **Do**
$u := r/b$
$q := q + u$
$r := r - u \cdot b$

Definition 2.2 *A number $p \in \mathbb{Z}$ is called* **prime** *if its only positive divisors are* 1 *and p (itself). Numbers which are not prime are called* **composite**.

Theorem 2.3 *Every integer $n > 1$ has a prime divisor*

Proof: The integer n has a divisor that is greater than 1, namely n. Among all divisors of n that are greater than 1, let p be the smallest. Then p must be prime. Otherwise, p would have a divisor m with

$$1 < m < p \leq n$$

this contradicts the assumption that p is the smallest divisor of n greater than 1. ■

Theorem 2.4 (Euclid) *In the ring $\mathbb{Z}$ the number of primes is infinite.*

Proof: The proof is by contradiction. Obviously there is at least one prime, namely 2. Suppose that the largest prime is p. Form the number

$$N = 2 \cdot 3 \cdot 5 \cdots p + 1. \tag{2}$$

Obviously $N > p$, so by assumption N cannot be prime. Since $N \in \mathbb{Z}$ for the above theorem, then N has a prime factor, say q, and clearly q cannot be among $2, 3, \ldots, p$, for no such $q \leq p$ can divide the right side of equation (2). So $q > p$, a contradiction. We conclude that there is no largest prime number. ■

Theorem 2.5 *If n is composite, then n has a prime divisor p such that $p \leq \sqrt{n}$*

Proof: Let p be the smallest prime divisor of n. If $n = rs$, then $p \leq r$ and $p \leq s$. Hence, $p^2 \leq rs = n$. That is, $p \leq \sqrt{n}$. ■

This theorem can be used to find all the prime numbers up to a given positive integer n, this procedure is called the *Sieve of Eratosthenes*, assuming that n is relatively small.

Algorithm 2.2 (Sieve of Eratosthenes)
Input : $n > 1, n \in \mathbb{Z}$
Output : *All prime numbers up to n.*
Method :

1. *Create a list of integers from* 2 *to* n
2. *For prime numbers* p_i $(i = 1, 2, \ldots)$ *from 2,3,5, up to* $\lfloor \sqrt{n} \rfloor$, *delete all the multiples* $p_i < p_i m \leq n$ *from the list;*
3. *Print the integers remaining in the list.*

2.1 Greatest Common Divisors

Definition 2.3 *Let* a *and* b *be integers, not both zero. The largest divisor* d *such that* $d \mid a$ *and* $d \mid b$ *is called the* Greatest Common Divisor (gcd) *of* a *and* b. *Such* d *is denoted as :*

$$d = gcd(a, b)$$

Definition 2.4 *We say that* a *and* b *are* **relatively prime** *or in short* **coprime** *if* $gcd(a, b) = 1$

Theorem 2.6 *Let* a, b *be integers not all zero then gcd(a,b) exists and is unique. As a consequence there exist integers* x, y *such that*

$$d = gcd(a, b) = ax + by$$

Proof: Consider the set $A = \{au + bv, \quad where \quad u, v \in \mathbb{Z}\}$. Clearly A contains positive, negative values as well as 0. Choose x and y such that $m = ax + by$ is the smallest positive integer in A. Use the Division algorithm, to write $a = mq + r$, where $0 \leq r < m$ then

$$r = a - mq = a - q(ax + by) = (1 - qx)a + (-qy)b$$

and hence $r \in A$. But $r < m$, so it follows from the definition of m that $r = 0$. Thus $a = mq$, that is $m \mid a$; similarly, $m \mid b$. Therefore, m is a common divisor of a and b. Since $d \mid a$ and $d \mid b$, $d \leq m$. Since $d = gcd(a, b)$, we must have $d = m$. Uniqueness: If there are two greatest common divisors d_1 and d_2 then it follows from its definition that $gd_1 = d_2$ and $hd_2 = d_1$ for some $g, h \in \mathbb{Z}^+$;

hence $d_2 = ghd_2$ thus $1 = gh$, and so $g = h = 1$. We conclude that $d_1 = d_2$. ■

We will see later in Theorem (2.10) how the values of x and y can explicitly be calculated.

Theorem 2.7 *If a prime number divides the product of two integers, then it divides at least one factor*

Proof: Suppose the prime number p divides ab but not a. Since p is a prime number, we must have $gcd(a, p) = 1$. By Theorem(2.6) there are x, y with $1 = ax + py$. This implies

$$b = abx + pby$$

Since p divides abx and pby, then by Theorem(2.1) we can conclude that p is a divisor of b. ■

2.2 Euclid's GCD Algorithm

Euclid's algorithm for finding the greatest common divisor of two integers is probably the oldest nontrivial algorithm that has survived up to the present day. It is based in the Division algorithm (eq (1)) and in the following theorem.

Theorem 2.8 *Let $a, b, q, r \in \mathbb{Z}$ with $b > 0$ and $0 \leq r < b$ such that $a = bq + r$. Then $gcd(a, b) = gcd(b, r)$*

Proof: Let $X = gcd(a, b)$ and $Y = gcd(b, r)$, it suffices to show that $X = Y$. If integer c is a divisor of a and b, it follows from the equation $a = bq + r$ and divisibility properties that c is also a divisor of r. By the same argument, every common divisor of b and r is a divisor of a. ■

Theorem 2.9 *[Euclid's GCD method]*
Let $a, b \in \mathbb{Z}^+$ with $a > b$. if $b \mid a$ then $gcd(a, b) = b$. If $b \nmid a$, then apply the

division algorithm *repeatedly as follows:*

$$\begin{array}{lcll} a & = & r_0 & \\ b & = & r_1 & \\ r_0 & = & r_1 q_1 + r_2 & 0 < r_2 < r_1, \\ r_1 & = & r_2 q_2 + r_3 & 0 < r_3 < r_2, \\ r_2 & = & r_3 q_3 + r_4 & 0 < r_4 < r_3, \\ \cdots & \cdots & & \\ r_{n-2} & = & r_{n-1} q_{n-1} + r_n & 0 < r_n < r_{n-1}, \\ r_{n-1} & = & r_n q_n + 0 & (r_{n+1} = 0) \end{array} \tag{3}$$

Then r_n, the last nonzero remainder, is the greatest common divisor *of a and b. That is,*

$$gcd(a, b) = r_n$$

Values for x, y in $gcd(a, b) = ax + by$ can be obtained by writing each r_i as a linear combination of a, b.

Proof: The chain of equations is obtained by dividing r_{i-1} into r_i. (Note that we have written the inequalities for the remainder without an equality sign). The process stops when the division is exact, that is, whenever $r_{i+1} = 0$ for $i = 1, 2, \ldots, n$.

We now prove that r_n is the greatest common divisor of a and b. One sees that for $i := 1, \ldots, n$, we have $r_{i-1} := r_i q_i + r_{i+1}$, from Theorem(2.8) it follows that the common divisors of r_{i-1} and r_i are the same as the common divisors of r_i and r_{i+1}, and hence

$$gcd(r_{i-1}, r_i) = gcd(r_i, r_{i+1})$$

Then by mathematical induction, we have

$$\begin{aligned} gcd(a, b) = gcd(r_0, r_1) = gcd(r_1, r_2) = \cdots = gcd(r_{n-1}, r_n) \\ = gcd(r_n, 0) = r_n \end{aligned}$$

To see that r_n is a linear combination of a and b, we argue by induction that each r_i is a linear combination of a and b. Clearly r_2 is a linear combination of

a and b, since $r_2 = a - bq_1$, so does r_3. In general, r_i is a linear combination of r_{i-1} and r_{i-2} (by definition). By inductive hypothesis we may suppose that these latter two numbers are linear combinations of a and b, and it follows that r_i is also a linear combination of a and b. ■

Euclid's GCD Algorithm
Input : $a, b \in \mathbb{Z}$, $b > 0$
Output : $r := gcd(a, b)$
Method :

While ($b > 0$) **Do**
$r := a \bmod b$
$a := b$
$b := r$

Let a and b be non-negative integers, and let $d := gcd(a, b)$. Then we know from Theorem (2.6) that there exist integers x and y such that $ax + by = d$.

If during the computation of the $gcd(a, b)$ by means of the Euclidean algorithm we keep track of additional values then we can efficiently compute x and y. This algorithm is called the **extended Euclidean algorithm** and will allow us later in the computation of the inverse of a number modulo another.

Theorem 2.10 *[Euclid's Extended GCD method]*
Let $a, b, r_0, \ldots, r_{n+1}, q_1, \ldots, q_n$ as in Theorem (2.9). Define integers $s_0, \ldots, s_{n+1}$, $t_0, \ldots, t_{n+1}$ as follows :

$$\begin{array}{ll} s_0 := 1 & t_0 := 0, \\ s_1 := 0 & t_1 := 1 \end{array} \tag{4}$$

And for $i := 1, \ldots, n$

$$s_{i+1} := s_{i-1} - s_i q_i \quad t_{i+1} := t_{i-1} - t_i q_i \tag{5}$$

Then

(i) *For $i := 0, \ldots, n+1$, we have $s_i a + t_i b = r_i$; in particular, $s_n a + t_n b = gcd(a, b)$*

(ii) *For* $i := 0, \ldots, n$ *we have* $s_i t_{i+1} - t_i s_{i+1} = (-1)^i$

(iii) *For* $i := 0, \ldots, n+1$ *we have* $gcd(s_i, t_i) = 1$

(iv) *For* $i := 0, \ldots, n$ *we have* $t_i t_{i+1} \leq 0$ *and* $|t_i| \leq |t_{i+1}|$
For $i := 1, \ldots, n$ *we have* $s_1 s_{i+1} \leq 0$ *and* $|s_i| \leq |s_{i+1}|$

(v) *For* $i := 0, \ldots, n+1$ *we have* $r_{i-1}|t_i| \leq a$ *and* $r_{i-1}|s_i| \leq b$

Proof: (i) is easily proved by induction on i. For $i := 0, 1$ the statement is clear. For $i := 2, ..., n+1$, we have:

$$\begin{aligned} s_i a + t_i b &= (s_{i-2} - s_{i-1}q_{i-1})a + (t_{i-2} - t_{i-1}q_{i-1})b \\ &= (s_{i-2}a + t_{i-2}b) - (s_{i-1}a + t_{i-1}b)q_i \\ &= r_{i-2} - r_{i-1}q_{i-1} \ (\textit{by induction}) \\ &= r_i \end{aligned} \tag{6}$$

(ii) is also easily proved by induction on i. For $i := 0$, the statement is clear. For $i := 1, \ldots, n$, we have

$$\begin{aligned} s_i t_{i+1} - t_i s_{i+1} &= s_i(t_{i-1} - t_i q_i) - t_i(s_{i-1} - s_i q_i) \\ &= -(s_{i-1}t_i - t_{i-1}s_i) \ (\textit{after expanding and simplifying}) \\ &= -(-1)^{i-1} \\ &= (-1)^i \ (\textit{by induction}) \end{aligned} \tag{7}$$

(iii) follows directly from (ii).
For (iv) one can easily prove both statements by induction on i. The statement involving the t_i is clearly true for $i := 0$. For $i := 1, \ldots, n$, we have $t_{i+1} := t_{i-1} - t_i q_i$, and since by induction hypothesis t_{i-1} and t_i have opposite signs and $|t_i| \geq |t_{i-1}|$, it follows that $|t_{i+1}| = |t_{i-1}| + |t_i|q_i \geq |t_i|$, and the sign of t_{i+1} is the opposite of that of t_i. The proof of the statement involving the s_i is the same, except that we start the induction at $i := 1$.
For (v), one considers the two equations:

$$\begin{aligned} s_{i-1}a + t_{i-1}b &= r_{i-1} \\ s_i a + t_i b &= r_i \end{aligned} \tag{8}$$

Subtracting t_{i-1} times the second equation from t_i times the first, applying (ii), and using the fact that t_i and t_{i-1} have opposite sign, we obtain:

$$a = |t_i r_{i-1} - t_{i-1} r_i| \geq |t_i| r_{i-1}$$

from which the inequality involving t_i follows. The inequality involving s_i follows similarly, subtracting s_{i-1} times the second equation from s_i times the first. ■

Extended Euclid's GCD Algorithm

Input : $a, b \in \mathbb{Z}, \quad b > 0$

Output : $\bar{v} = (d, s, t)$

Method :

Let $\bar{u} := (a, 1, 0)$

Let $\bar{v} := (b, 0, 1)$

While ($v_1 > 0$) **Do**

Let $\bar{w} := \bar{u} - \lfloor u_1 / v_1 \rfloor \bar{v}$

Let $\bar{u} := \bar{v}$

Let $\bar{v} := \bar{w}$

Example 2.1 *Suppose* $a := 100, b := 35$ *then the numbers* s_i *and* t_i *are easily computed from the* q_i*:*

i	0	1	2	3	4
r_i	100	35	30	5	0
q_i		2	1	6	
s_i	1	0	1	-1	7
t_i	0	1	-2	3	-20

So we have $gcd(a, b) := 5 = -a + 3b$

This is the code in Maple:

```
> restart;
> extended_gcd:=proc(a,b)
>  local r0,r1,s0,s1,t0,t1,temp,q;
```

```
> r0:=a; r1:=b;
> s0:=1; s1:=0;
> t0:=0; t1:=1;
> while r1>0 do
>   q:= floor(r0/r1);
>   temp:= r0-r1*q; r0:=r1; r1:=temp;
>   temp:= s0-s1*q; s0:=s1; s1:=temp;
>   temp:= t0-t1*q; t0:=t1; t1:=temp;
> od;
> print(r0,s0,t0);
> end:
>
> extended_gcd(100,35);
```

$$5, -1, 3$$

There are more efficient ways to compute the values s_i and t_i see for example [6](pag. 61)

3 Modular Inverses and the CRT

One application of the Extended Euclidean Algorithm is to the problem of computing multiplicative inverses in $\mathbb{Z}_n$, where $n > 1$. It also is used in the CRT (Chinese Remainder Theorem) which is part of the RSA algorithm, as we explain later in detail.

3.1 Fundamental Theorem of Arithmetic

The Fundamental Theorem of Arithmetic states that all numbers are expressible as a unique product of primes. It may seem obvious—how could it be any other way?. In fact there are other "number systems" which look like the integers (with primes, factorization, etc.) but for which unique factorization fails. Here is a simple example: Let $2\mathbf{Z}$ denote the set of even integers. You can easily verify that $2\mathbf{Z}$ is closed under addition and multiplication. Call a number $n \in 2\mathbf{Z}$ *composite* if it's a product of two other numbers in $2\mathbf{Z}$, *prime* otherwise. It is easy to think of some primes in $2\mathbf{Z}$: $2, 6, 10, \ldots$. In fact, you can check that anything of the form $4k + 2$ is prime. Similarly, anything of the

form $4k$ is composite. Now notice that 60 factors into primes in two ways, as

$$60 = 6 \cdot 10 = 2 \cdot 30.$$

Theorem 3.1 *Let $a \in \mathbb{Z} \setminus \{0, 1, -1\}$, there exist determined prime numbers $p_1, \ldots, p_n$, $p_1 < \ldots < p_n$ and $\alpha_i \in \mathbb{N}^+$ with:*

$$a = sgn(a) \prod_{i=1}^{n} p_i^{\alpha_i}$$

Proof: (Based on [3]) First we'll show that any integer can be factored. Start with some integer $n > 1$. If n is prime, we are done. Otherwise $n = ab$ where $a, b > 1$. If a and b are prime, we are done. If one or both are not prime apply the same argument to each piece. For example, if a is prime but b is not then we can write $b = b_1 b_2$, and so $n = ab_1 b_2$. If all pieces are prime we are done, otherwise apply the argument again to any composite factor. At each stage the composite factors decrease by at least a factor of 2. It is clear this can continue for only a finite number of steps before all of the factors are prime.

Proving that the factorization is unique is a little harder. The proof is by contradiction. Suppose that at least one integer greater than one can be factored in TWO ways. Then there is a smallest integer which factors in two ways—let n be that integer, and suppose

$$n = p_1 p_2 \cdots p_r = q_1 q_2 \cdots q_s$$

where all p's and q's represent primes. Of course all of the p's and q's must be distinct, for if, for example, $p_1 = q_k$ for some k then we could divide both sides by p_1 and find a smaller n with two distinct factorizations. We can suppose that $p_1 < q_1$. Start with $p_1 p_2 \cdots p_r = q_1 q_2 \cdots q_s$ and subtract $p_1 q_2 q_3 \cdots q_s$ from both sides to obtain

$$p_1(p_2 p_3 \cdots p_r - q_2 q_3 \cdots q_s) = (q_1 - p_1) q_2 q_3 \cdots q_s \tag{9}$$

Define a new integer N by $N = (q_1 - p_1) q_2 q_3 \cdots q_s$ (N equals both sides of the above equation). Obviously $1 < N < n$, so N must factor uniquely into

prime numbers. Now, we can find a prime factorization of N which contains the prime p_1—just finish factoring the remaining stuff in the parentheses on the left side of equation (9). But if we finish the factorization of N using the right side of (9) (by factoring $q_1 - p_1$), this leads to a factorization which cannot include p_1, since $q_1 - p_1$ is not divisible by p_1. This means that $N < n$ factorizes into primes in two different ways, contradicting our choice of n as the smallest such example. Thus the unique factorization must be true for all integers. ■

With the Fundamental Theorem at our disposal we can now prepare the demonstration of the Chinese Remainder theorem.

3.2 Modular Arithmetic and CRT

Modular Arithmetic plays a crucial role in the RSA algorithm as we will see later. This notion was first introduced by Gauss in his *Disquisitiones Arithmeticae* in 1801, though the ancient Greeks and Chinese had already had the idea.

Definition 3.1 *Let $a \in \mathbb{Z}$ and $n \in \mathbb{Z}^+, n > 1$. We define "a mod n" to be the remainder $r \in \mathbb{Z}$ when a is divided by n, that is*

$$r = a \bmod n = a - \lfloor a/n \rfloor n$$

We may also say that "r is equal to a reduced modulo n"

Given the well-defined notion of the remainder of one integer when divided by another, it is convenient to provide a special notion to indicate equality of remainders.

Definition 3.2 *Let $a, b \in \mathbb{Z}$ and $n \in \mathbb{Z}^+$. We say that "a is congruent to b modulo n", denoted by*

$$a \equiv b \pmod{n}$$

if n is a divisor of $a - b$, or equivalently, if $n \mid (a - b)$.

Similarly, we write

$$a \not\equiv b \ (mod\ n)$$

If a is not congruent (or incongruent) to b modulo n, or quivalently, if $n \nmid (a - b)$.

Definition 3.3 *If* $a \equiv b \ (mod\ n)$, *then b is called a* residue *of a modulo n. If* $0 \leq b \leq n - 1$, *b is called the* least nonnegative residue *of a modulo n.*

From these two definitions the following equivalences are valid:

$$a \equiv b \ (mod\ n) \Longleftrightarrow a \bmod n \ = \ b \bmod n$$

Theorem 3.2 *If* a, b, c *and* $d \in \mathbb{Z}$ $(c \neq 0)$, *the following statements hold (Equivalence Relation):*

- *Reflexive:* $a \equiv a \ (mod\ c)$
- *Symmetric: if* $a \equiv b \ (mod\ c)$, *then* $b \equiv a \ (mod\ c)$
- *Transitive: if* $a \equiv b \ (mod\ c)$ *and* $b \equiv d \ (mod\ c)$ *then* $a \equiv d \ (mod\ c)$

Theorem 3.3 *Suppose* $a \equiv b \ (mod\ n)$ *and* $c \equiv d \ (mod\ n)$ *Then :*

- $a \pm b \equiv c \pm d \ (mod\ n)$
- $ab \equiv cd \ (mod\ n)$
- *if* $a \equiv b \ (mod\ n)$ *then for* $k \in \mathbb{Z}$, $a^k \equiv b^k \ (mod\ n)$

Theorem 3.4 (Chinese Remainder) *if* $p, q \in \mathbb{Z}$ *prime numbers such that*

$$m \equiv a \ (mod\ p) \quad and \quad m \equiv a \ (mod\ q)$$

then

$$m \equiv a \ (mod\ pq)$$

Proof: $m \equiv a \ (\ mod\ p\ )$ if and only if $m - a = kp$, for some $k \in \mathbb{Z}$. By Theorem(3.1) $m - a$ can be expressed as $p_1.p_2.p_3 \ldots$. If $m - a$ is divisible by both p, q and $GCD(p, q) = 1$ then p and q must be one of $p_1.p_2.p_3 \ldots$. Therefore, $m - a$ is divisible by $p.q$. ■

Theorem 3.5 *If* $a \equiv b \ (\ mod\ n)$ *and* $gdc(a, n) = 1$ *then* $gcd(b, n) = 1$

Proof: There $\exists k \in \mathbb{Z}$ such that $a = b + kn$. Since $gdc(a, n) = 1$ then by Theorem(2.6) $\exists x, y \in \mathbb{Z}$ such that $xa + yn = 1$ so

$$1 = x(b + kn) + yn = xb + (xk + y)n = xb + y_1 n, \quad where\ y_1 \in \mathbb{Z}$$

therefore $gcd(b, n) = 1$. ■

Definition 3.4 *If* $a\tilde{a} \equiv 1 \ (\ mod\ n\ )$, *we say that* $\tilde{a}$ *is the* **inverse** *of a modulo n.*

Theorem 3.6 *If* $gcd(a, b) = 1$ *then a has a unique inverse* $\tilde{a}$ *modulo b*

Proof: We have that $gcd(a, b) = 1 = ax + by$ for some $x, y \in \mathbb{Z}$. Therefore we have: $ax = 1 - by = 1 + by_1$ making $y_1 = -y$). So $ax \equiv 1 \ (\ mod\ b\ )$ and as consequence x is the inverse of a modulo b. Now, let us assume that a has two inverses a' and a'' then $aa' \equiv 1 \ (\ mod\ b\ )$ and $aa'' \equiv 1 \ (\ mod\ b\ )$. Therefore $a(a' - a'') \equiv 0 \ (\ mod\ b\ )$ ■

We are now in the position to provide a method to compute the multiplicative inverse of an integer in $\mathbb{Z}_n$.

Algorithm 3.1 [Multiplicative inverses in $\mathbb{Z}_n$]
Input : $a \in \mathbb{Z}_n$
Output: a^{-1} *mod n, provided it exists*
Method:

1. *Use the Extended Euclidean algorithm (Theorem 2.10) to compute* $x, y \in \mathbb{Z}$ *such that* $d = ax + ny$ *where* $d := gcd(a, n)$

2. *If* $d \neq 1$ *then* a^{-1} *mod n does not exist. Otherwise, Return(x).*

Modular exponentiation can be performed efficiently with the repeated square-and multiply algorithm , which is crucial for many cryptographic protocols. One version of this algorithm is based on the following observation. Let the binary representation of k be $\sum_{i:=0}^{t} k_i 2^i$ where $k_i \in \{0, 1\}$:

$$a^k = \prod_{i:=0}^{t} a^{k_i 2^i} = (a^{2^0})^{k_0} (a^{2^1})^{k_1} \ldots (a^{2^t})^{k^t}$$

Algorithm 3.2 [Fast exponentiation algorithm in $\mathbb{Z}_n$]

Input : $a \in \mathbb{Z}_n$ *and* $0 \leq k < n$ *with binary representation:* $\sum_{i:=0}^{t} k_i 2^i$
Output : a^k *mod n*
Method :

1. *Set* $b := 1$. *If* $k = 0$ *then return(b)*
2. *Set* $A := a$
3. *If* $k_0 = 1$ *then set* $b := a$
4. *For i from 1 to t do the following:*
  4.1 *Set* $A := A^2$ *mod n*
  4.2 *If* k_i*=1 then set* $b := A.b$ *mod n*
5. *Return(b)*

Definition 3.5 (Euler Totient ϕ function)
Let $n \in \mathbb{Z}^+$, the Euler phi-Function $\phi(n)$ is defined to be the number of non-negative integers b less than n which are prime to n:

$$\phi(n) := |\{\, b \in \mathbb{N},\ 0 \leq b < n :\ gcd(b, n) = 1 \,\}|$$

Theorem 3.7 *If p is a prime number, then* $\phi(p) = p - 1$

Proof: Since p is prime, then $\forall k := 1, \ldots, p-1 \ :\ gcd(k, p) = 1$. So it follows from the definition of Euler's ϕ-function that $\phi(p) = p - 1$. ■

Theorem 3.8 *If p, q are prime numbers, then* $\phi(p.q) = (p - 1)(q - 1)$

Proof: We count the numbers from 1 to $p.q$ which share factors with $p.q$:

$$1.p,\ 2.p,\ \ldots,(q-1)p$$

$$1.q,\ 2.q,\ \ldots,(p-1)q$$

$$p.q$$

The rest are coprime to $p.q$. This count results in:

$$\phi(p.q) = p.q - (p-1) - (q-1) - 1 = (p-1)(q-1)$$

■

Theorem 3.9 *If p is prime and $k > 0$ then $\phi(p^k) = p^k - p^{k-1}$*

Proof: Only numbers that are a multiple of p have a common factor with p^k:

$$1.p,\ 2.p,\ \ldots,\ p^{k-1}.p$$

and the rest do not share any factors, so are coprime. Therefore

$$\phi(p^k) = p^k - p^{k-1}$$

■

Theorem 3.10 *if m, n are coprime, i.e. $gcd(m,n) = 1$ then $\phi(m.n) = \phi(m).\phi(n)$*

Proof: Organize into a matrix of m columns, and n rows

1	2	3	$\cdots$	r	$\cdots$	m
$m+1$	$m+2$	$m+3$	$\cdots$	$m+r$	$\cdots$	$2m$
$2m+1$	$2m+2$	$2m+3$	$\cdots$	$2m+r$	$\cdots$	$3m$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$(n-1).m+1$	$(n-1).m+2$	$(n-1).m+3$	$\cdots$	$(n-1).m+r$	$\cdots$	nm

Step(1) Eliminate columns: if $gcd(m,r) = 1$ then $gcd(m, km + r) = 1\ \forall k := 1,\ldots,n$ (see Theorem(3.5)) therefore all cells under that r^{th} column have no common factors with m. So, others have a common factor with $m.n$ and can be eliminated, as consequence $\phi(m)$ columns survive.

Step(2) Examine cells in remaining columns: No two cells in a column are congruent *mod n* (otherwise if $i.m+r \equiv j.m+r$ (*mod n*) then $i.m+r-j.m-r = k.n$ then $n|(i-j)$, which is not possible because $i - j < n$). As there are n (non-congruent) cells in each column, label them as $0, 1, 2, \ldots, n-1$ in some order. Then $\phi(n)$ cells in each column coprime to n therefore $\phi(n).\phi(m)$ cells left that are coprime to both m and n. ■

Theorem 3.11 *If* $gcd(a,n) = 1$ *and* $x_1, x_2, \ldots, x_{\phi(n)}$ *are coprime to n, then* $a.x_1, a.x_2, \ldots, a.x_{\phi(n)}$ *are congruent to* $x_1, x_2, \ldots, x_{\phi(n)}$ *in some order.*

Proof: $a.x_i \not\equiv a.x_j$ (*mod n*), otherwise $a(x_i - x_j) = k.n$, but $gcd(a,n) = 1$ then $n|(x_i - x_j)$, which is impossible because $x_i - x_j < n$. Using Theorem(3.5) let $a.x_i \equiv b$ (*mod n*), since $gcd(a.x_i, n) = 1$ then $gcd(b,n) = 1$. Therefore b must be one of x_j. ■

Theorem 3.12 (Euler)
Let $n \in \mathbb{N}$, $n \geq 2$ then

$$\forall a \in \mathbb{N},\quad \gcd(a,n) = 1 :\qquad a^{\phi(n)} \equiv 1 \bmod n$$

Proof: Let $x_1, x_2, \ldots, x_{\phi(n)} < n$ and coprime to n. Since a is also coprime to n, from Theorem (3.11) $\exists x_i, x_j :\ a.x_i \equiv x_j$ (*mod n*) which are unique. Since there are $\phi(n)$ of the $a.x_i$ and $\phi(n)$ of the x_j. Thus

$$ax_1.ax_2.\ldots.a.x_{\phi(n)} \equiv x_1.x_2.\ldots.x_{\phi(n)}\ (\ mod\ n\ )$$

Hence, if $R := x_1.x_2.\ldots.x_{\phi(n)}$, then

$$a^{\phi(n)}.R \equiv R\ (\ mod\ n\ )$$

since $n \nmid R$ (because R is the product of integers each of which is relatively prime to n) then $n|(a^{\phi(n)} - 1)$ ■

Theorem 3.13 (Fermat's Little Theorem)
Let $p \in \mathbb{N}$ be a prime number, then

$$\forall a \in \mathbb{Z}, \qquad a^{p-1} \equiv 1 \bmod p$$

Proof: Since p is prime then $\phi(p) = p - 1$. Now apply the Theorem (3.12) ■

4 The RSA Algorithm

In 1977, three MIT researchers Ronald Rivest, Adi Shamir and Leonard Adleman proposed the first practical public-key cryptosystem (i.e a mechanism through which people can send secret messages without the need of exchanging previously a secret key). The RSA cryptosystem is based on the following assumption:

RSA Assumption: It is not difficult to find two large prime numbers, but it is very difficult to factor a large composite into its prime factorization form.

4.1 How to choose the keys

It is important to define the appropriate keys because they will the determine how the message will be encrypted and decrypted.

Theorem 4.1 (Choosing Key)
Given p, q prime numbers and $e \in \mathbb{Z}^+$, such that $gcd(e, \phi(p.q)) = 1$, then

$$\exists\, d \in \mathbb{Z}^+ \;:\; e.d \equiv 1 \;(\; mod\; \phi(p.q)\;)$$

Proof: This follows directly from Theorem (3.6). Let $b := \phi(p.q)$ and $a := e$. Also recall, that in this case since p, q are prime numbers then by Theorem (3.8) $\phi(p.q) = (p - 1)(q - 1)$ ■

Theorem 4.2 (RSA Encryption)
Given $p, q \in \mathbb{Z}^+$ prime numbers. Take $n := p.q$ and e and d as in Theorem (4.1), then

$$\forall M \in \mathbb{Z}^+ \ : \ M^{e.d} \equiv M \ (\ mod \ n \ )$$

Proof: Since $e.d \equiv 1 \ (\ mod \ \phi(p.q) \ )$ then $e.d = 1 + k.\phi(p.q)$ for some integer k. Thus:

$$M^{e.d} = M^{1+k.\phi(p.q)} = M^{1+k.(p-1).(q-1)} = M.(M^{(p-1).(q-1)})^k$$

If M is relatively prime to p then by Fermat's little theorem (Theorem 3.13):

$$M^{e.d} = M.(M^{p-1})^{k.(q-1)} \equiv M.(1)^{k.(q-1)} \equiv M \ (\ mod \ p \ ) \qquad (10)$$

If M is not relatively prime to p, i.e. multiple of p then equation (10) still holds because both sides will be zero, modulo p. By exactly the same reasoning

$$M^{e.d} = M.(M^{q-1})^{k.(p-1)} \equiv M.(1)^{k.(p-1)} \equiv M \ (\ mod \ q \ ) \qquad (11)$$

If we apply the Chinese remainder Theorem (3.4) to equations (10) and (11), we obtain the result we want: $M^{e.d} \equiv M \ (\ mod \ n \ )$ ■

Algorithm 4.1 (RSA Key Generation Algorithm)
Summary: *Each entity (A and B) creates an RSA* ***public key (n,e)*** *and corresponding* ***private key(n,d)****. Each entity A should do the following:*

Input : *None*
Output : *Public Key* (n, e) *and Private Key* (n, d)
Method :

1. *Generate* ***p,q*** *: two large (random) and distinct primes*
2. *Compute* ***n,φ****: $n := p.q$ and $\phi := (p-1)(q-1)$*
3. *Choose* ***e****: Such that $gcd(e, \phi) = 1$*
4. *Compute* ***d*** *: Use Algorithm (3.1) to compute $d := e^{-1} \ (mod \ \phi)$*

Algorithm 4.2 (RSA Public Key Encryption Algorithm)
Summary: *B encrypts a messege m for A, which A decrypts*

I. Encryption. *B should do the following*
Input : A's Public Key (n, e) *and Message m*
Output : Encrypted Message c for A
Method :

1. *Obtain A's Public Key* (n, e)
2. *Compute* $c := m^e \ mod \ n$
3. *Send the encrypted message c to A*

II. Decryption. *To recover the text m from c, A should do the following:*
Input : A's Private Key (n, d)*, encrypted message c*
Output : Original message m for A
Method :

1. *Use A's Private Key* (n, d) *to compute* $m := c^d \ mod \ n$

Example 4.1 Take $p = 47$ and $q = 71$ then $n := 3337$ The encryption key e, must have no factors in common with

$$\phi := (p-1)(q-1) = 46.70 = 3220$$

Choose e (at random) to be 79. In that case (using algorithm 3.1)

$$d := 79^{-1} \ (mod \ 3220) = 1019$$

Publish e and n and keep d secret. Discard p and q. To encrypt the message

$$m := 6882326879666683$$

first break it into small blocks. Three-digit block work nicely in this case. The message is split into six blocks, m_i, in which:

$$\begin{aligned} m_1 &= 688 \\ m_2 &= 232 \\ m_3 &= 687 \\ m_4 &= 966 \\ m_5 &= 668 \\ m_6 &= 003 \end{aligned} \tag{12}$$

The first block is encrypted as

$$688^{79} \ mod \ 3337 = 1570 = c_1$$

Performing the same operation on the subsequent blocks generates an encrypted message:

$$c := 1570\ 2756\ 2091\ 2276\ 2423\ 158$$

Decrypting the message requires performing the same exponentiation using the decryption key of 1019, so:

$$1570^{1019} \ mod \ 3337 = 688 = m_1$$

The rest of the message can be recovered in this manner.

5 RSA using Maple

The first thing to do to use RSA is to find two large primes. For this we will be looking for primes that are about 60 digits long. We do this with the rand and nextprime functions in Maple.

```
> M1 := rand(10^64)();
> M2 := rand(10^64)();
> P1 := nextprime(M1);
> P2 := nextprime(M2);
```

$$M1 := 9081321110693270343633073697474256143563558458718976746753830538$$

$$M2 := 2085722974121768604305613921745580037409259811952655310075487163$$

$$P1 := 9081321110693270343633073697474256143563558458718976746753830829$$

$$P2 := 2085722974121768604305613921745580037409259811952655310075487789$$

Next, we compute the values of n and ϕ:

```
> n := P1*P2;
> phi := (P1-1)*(P2-1);
```

n := 1894112007594997081980860359826587859602862500955039020
60559744454130761375068380983333438227587569461850188480
434411606561247081

phi := 1894112007594997081980860359826587859602862500955039020
60559744342460320526917991503946562035389207652122005773
71779549731928464

It seems that the best criterion for mathematical ease is that e represented base 2 should be almost all zeroes.

```
> e:= 2^16+1;  isprime(e); gcd(e,phi);
```

$$e := 65537$$
$$true$$
$$1$$

Recall that we had to make sure that e is relatively prime to ϕ. That is easier because we have chosen e to be prime. Now we need to compute d, the multiplicative inverse of e mod ϕ. We could do that directly with the extended Euclidean algorithm, but we will just use the modular arithmetic of Maple

```
> d := eval(1/e mod phi);
```

d := 1235101952249449933601661160522899006364776065097785869746
55585355562570452901170891131965650826796754525586945908158735642369393

We are ready to define encoding and decoding procedures.

```
> encoder := (m,e,n) -> Power(m,e) mod n:
> decoder := (c,d,n) -> Power(c,d) mod n:
```

Let us suppose the message is $m := 5$ which encrypted will be *enc* and decrypted will be *deco*

```
> m:=5:enc := encoder(m,e,n); deco :=decoder(enc,d,n);
```

enc := 1649261803527220202325280913722581577396504584823090691737774692060932427924773111918603876574359728038896263512525359993692180 6

$deco$:= 5

Let us take another example. This is another way to show the encryption and decryption, using :

```
> m := convert("RSA Algorithm is cool", bytes);
```

m := [82, 83, 65, 32, 65, 108, 103, 111, 114, 105, 116, 104, 109, 32, 105, 115, 32, 99, 111, 111, 108]

Now, let us encrypt this message (i.e each letter), as follows:

```
> emess := [seq(powmod(m[k], e, n), k=1..nops(m))];
```

emess := [1870143554033981736010183063292124744746438619910485941390872935036869678006295756055057722597470788133832946022533390192126865.
831327084050213382203824642606025285869539068966744764395650843488145312271778685475850969262718615538846728467647485412647052 6.
351068841781410801452568318085248116192930719991549115097931604485505722976613317749083678836693418833219494421428949874348840 0.
406854673102273833935658492907859315520799677311507389689693432374854999221899400277221585806031300020052610350989036396221974.
351068841781410801452568318085248116192930719991549115097931604485505722976613317749083678836693418833219494421428949874348840 0.
106630071397633350561952563421419794952192540234543135833319170892910463780214085719011546946470473283138056073166009835348563193.
810040640836394383501083127981884492443557074316472620200013919450476768281378245435752611103986658236745444724747912430557578 55.
959190857671655783265865608223676505841366806396358945098273410096901417953981944237543845330052152793431500278262675020258918 2.
134836975718653275869701386470171874726694048133751909125339208330601957623519322240311610442132955339521272343357579373119631 25.
171693842013871022656968489798790547352295459255422608640695411124427502451966167222529050166388258394912250333065008211691701 7.
340330581880555066861077391408676080210284226272716896048125119002139992375907237754603173437568082634533891899344710912099514.
603669517098875408866813580700362040579916225430891237324113761854604267725404344077690323861823558463034910495328523426528041.
183474889679457928219093371225969319740413033916243039729261291563092274425522765411364415616575801758625935788670244978219 24979.
406854673102273833935658492907859315520799677311507389689693432374854999221899400277221585806031300020052610350989036396221974.
171693842013871022656968489798790547352295459255422608640695411124427502451966167222529050166388258394912250333065008211691701 7.
380948481339746064756937128853347176386291872268378378163751384278275797230630885022924006891639115466883028948730959828170541 4.
406854673102273833935658492907859315520799677311507389689693432374854999221899400277221585806031300020052610350989036396221974.
195273572723577230759184143014218446649719867856925820218138592387063280946074933048707179938804271836911147975143851036783031.
959190857671655783265865608223676505841366806396358945098273410096901417953981944237543845330052152793431500278262675020258918 2.
959190857671655783265865608223676505841366806396358945098273410096901417953981942375438453300521527934315002782626750202589182.
106630071397633350561952563421419794952192540234543135833319170892910463780214085719011546946470473283138056073166009835348563193}
```

```
> messg := [seq(powmod(emess[k], d, n),k=1..nops(emess))];
```

messg := [82, 83, 65, 32, 65, 108, 103, 111, 114, 105, 116, 104, 109, 32, 105, 115, 32, 99, 111, 111, 108]
```

```
> writebytes(default, messg):
```

RSA Algorithm is cool 21

6 On Primality Tests

It is clear that the most important task in the construction of the RSA algorithm is to find two large prime numbers and the most natural method to generate a prime number is to generate a random number n of appropriate size, and check if it is prime.

As we know, this can be performed by checking whether n is divisible by any of the prime numbers which are $\leq \sqrt{n}$ as in Theorem (2.5).

Nevertheless more efficient methods are required in practice. Most of the commercials software in the market use the following approach:

1. Generate as candidate a random odd number n of appropriate size.
2. Test n for primality.
3. If n is composite, return to the first step.

In step 1, a slight modification is to consider candidates restricted to some search sequence starting from n; a trivial search sequence which may be used is $n, n+2, n+4, n+6, \ldots$. Using specific search sequences may allow one to increase the expectation that a candidate is prime, and to find primes possessing certain additional desirable properties a priori.

In step 2, the test for **primality** might be either:

a. A test which proves that the candidate is prime. In which case the outcome of the generator is called a **provable prime** for example using the **Elliptic Curve method**, or

b. A test which establishes a weaker result, such as that n is **probably prime**. In which case the outcome of the generator is called a probable

prime: Fermat, Solovay-Strassen, Miller-Rabin probabilistic primality tests. See for example: [4],[6] and [7].

In the latter case, careful consideration must be given to the exact meaning of this expression. Most so-called probabilistic primality tests are absolutely correct when they declare candidates ***n*** to be **composite**, but do not provide a mathematical proof that ***n*** is **prime** in the case when such a number is declared to be *probably* so. Such tests are more properly called compositeness tests than probabilistic primality tests.

Other techniques exist whereby candidates ***n*** are specially constructed such that it can be established by mathematical reasoning whether a candidate actually is prime. These are called constructive prime generation techniques.

A final distinction between different techniques for prime number generation is the use of randomness. Candidates are typically generated as a function of a random input. The technique used to judge the primality of the candidate, however, may or may not itself use random numbers. If it does not, the technique is deterministic, and the result is reproducible; if it does, the technique is said to be randomized. Both deterministic and randomized probabilistic primality tests exist.

7 Summary

We have shown one technique to encrypt messages by means of a function $RSA(n, e, x) := x^e \ mod \ n$, where the case of interest is that n is the product of two large prime numbers p and q satisfying $gcd(e, \phi(n)) = 1$. We have seen RSA is easy to compute but the key is the difficulty of reversing RSA i.e. given n try to find its factors. If n can be factored then by the CRT we could reverse RSA easily. Current commercial software in the market make use of RSA and variations of it simply because there is already an infrastructure in place which would be very costly to replace (something that is happening but slowly) by more complex encryption techniques like AES (Advanced Encryption Standard).

References

[1] J.A. Buchmann, *Introduction to Cryptography*. Springer-Verlag, (2004), 2nd. Ed. ISBN 0-387-20756-2

[2] O. Foster, *Algoritmische Zahlentheorie*. Vieweg Verlag. (1996). ISBN 3-528-06580-X

[3] P. Giblin, *Primes and Programming*. Cambridge University Press, (1994), 2nd. Ed. ISBN 0-521-40998-8

[4] N. Koblitz, *A Course in Number Theory and Cryptography*. Springer-Verlag, (1988), 2nd. Ed. ISBN 0-387-96576-9

[5] H. Scheid, *Zahlentheorie*. Spektrum Akademischer Verlag. 3. Auflage (2003). ISBN 3-8274-1365-6

[6] V. Shoup, *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press. Reprint (2006). ISBN 0-521-85154-8

[7] S.Y. Yan, *Number Theory for Computing*. Springer-Verlag, (2002), 2nd. Ed. ISBN 3-540-43072-5

Resumen

En este artículo tratamos sobre los números primos y su uso actual en algoritmos de cifrado. Estos algoritmos hacen posible el intercambio por internet de datos sensibles, tales como transacciones bancarias, correos electrónicos y otras transacciones por Internet en las que la privacidad es importante.

Palabras Clave: Números Primos, Aritmética Modular, RSA, Cifrado

Jorge L. Anicama
Oracle Corporation, 6505 Blue Lagoon,
Drive Suite 400 Miami, FL 33126, USA
janicama@yahoo.com