This is the last preprint. The final paper appeared in"The Curves Seminar at Queen's", vol. 13, ed. by A.V. Geramita, Queen's Papers in Pure and Applied Mathematics, volume 119, (2000), 121-140.

AN INTRODUCTION TO CRYPTOGRAPHY

A. LANGUASCO

In this introduction to Cryptography, we start by giving some basic notions from elementary number theory and see how they can be applied to send "secret" messages. After introducing congruence theory, Fermat's little theorem and the Euler-Fermat theorem, we examine, from a historical point of view, some classic cryptographic systems. Then we give the main properties of modern cryptographic systems and, in particular, we define public key cryptography. One of the most important public-key systems (which exploits the fact that in $\mathbb{Z}$ primality algorithms are much "faster" than factorization ones) is presented: the R.S.A. cryptosystem. We also present another public key system which exploits the discrete logarithm problem and which is also consistently used as an alternative to R.S.A. (for example the U.S. government offices use it for their digital signature scheme). An analogue of the discrete logarithm problem is used in the so-called "elliptic curves cryptosystems". Unfortunately, the amount of mathematical theory needed to understand the elliptic curves method is too great to be explained here. So, we will say, in the last paragraph, just a few words on the differences between the discrete logarithm problem and its elliptic curve analogue. This article is geared toward the amateur interested in the subject.

Acknowledgment. The author wishes to thank Professor T. Geramita for his kind help and encouragement and Mr. M. Buckles (a student of the University of Notre Dame) for kindly checking the manuscript and offering some helpful improvements to the text.

1. Some facts about elementary number theory

1.1. The fundamental theorem of Arithmetic

First of all we recall the following definition.

Definition. *We call a natural number $p > 1$ prime if it can only be divided by 1 and itself.*

For example the first few primes are $2, 3, 5, 7, 11, 13, 17, 19$.
We say that a natural number n *divides* a natural number m if and only if there exists $k \in \mathbb{N}$ such that $m = kn$. We also recall that *if a prime p divides a product ab then p divides a or p divides b.* One of the most important and well-known results about prime numbers is the following.

Fundamental theorem of Arithmetic. *Every natural number greater than or equal to 2 can be written as a product of prime numbers in only one way (up to a change of order of the prime factors).*

Proof. Using the induction method we prove the decomposition of a number into a product of primes exists. $2 = 2$ gives 2 as the product of primes. Now suppose we know that every number $m < n$ can be expressed as a product of primes. If n is prime, then $n = n$ expresses n as the product of primes. Otherwise $n = ab$ with $a, b < n$ natural numbers. Both a and b can be written as a product of primes (by the inductive hypothesis); hence $n = ab$ is a product of primes also. Now we show that the decomposition of a number into primes is uniquely determined up to the order of the factors. Let $n = \prod_{i=1}^{r} p_i$ with the p_i primes satisfying $p_i \leq p_{i+1}$. Suppose that there exists another decomposition $n = \prod_{j=1}^{s} q_j$ where the q_j are prime numbers. Now p_1 is a prime which divides n and so p_1 divides $q_1 q_2 \ldots q_s$. Therefore p_1 divides q_1 or p_1 divides $q_2 q_3 \ldots q_s$. In the first case $p_1 = q_1$ since p_1 and q_1 are both prime. Otherwise it is easy to see that $p_1 = q_i$ for some i. By repeating the above argument applied to $n/p_1 (= n/q_i)$, we get $p_2 = q_l$ for some $l \neq i$. Ultimately, we conclude that $r = s$ and $\{p_i : i = 1, \ldots, r\} = \{q_j : j = 1, \ldots, r\}$. ■

1.2. The Euclidean algorithm

We now show how to "easily" compute the greatest common divisor (gcd) of two positive natural numbers a and b. We denote the gcd of a and b by (a, b).
Suppose a is larger than b.
If we divide a by b and get remainder r_1, we can write $a = bq_1 + r_1$ with $0 \leq r_1 < b$.
If we divide b by r_1 and get remainder r_2, we can write $b = r_1 q_2 + r_2$ with $0 \leq r_2 < r_1$.
If we divide r_1 by r_2 and get remainder r_3, we can write $r_1 = r_2 q_3 + r_3$ with $0 \leq r_3 < r_2$.
Since the remainders $r_1, r_2, r_3, \ldots$ form a strictly decreasing sequence of natural numbers, we can continue the above procedure until we get a remainder which is equal to zero, i.e. until we obtain, for some k, the equation $r_{k-2} = r_{k-1} q_k + r_k$ with $r_k = 0$. Then $r_{k-1} = (a, b)$.
This follows from the following observations. Every common divisor of a and b divides every r_i (this can be seen by rewriting the above equations with the remainders all on one side). So if r_{k-1} divides a and b then it is the largest number that does, and we are done. The above list of equations when read off in reverse order reveal to us that r_{k-1} divides r_{k-2} and then that r_{k-1} divides r_{k-3} and so on. Hence we get that r_{k-1} divides b.
Then, since r_{k-1} divides r_1 and b, it also divides a. Note that since every common divisor of a and b divides every r_i, every common divisor of a and b divides the gcd of a and b.
Let's look at a particular example.

Example. Compute $(1752, 612)$ using the euclidean algorithm:

$$1752 = 2 * 612 + 528; \qquad 612 = 1 * 528 + 84; \qquad 528 = 6 * 84 + 24$$
$$84 = 3 * 24 + 12; \qquad 24 = 2 * 12 + 0.$$

From the above computation we conclude that $(1752, 612) = 12$.

We now say what we mean by a "fast" algorithm. How can we measure the speediness of an algorithm? First of all we recall that a computer represents a natural number n by using its *binary representation*. Such a representation can be built in the following way:

1) write n as a sum of powers of two (starting with 2^0) (for example: $1 = 1 * 2^0$, $2 = 1 * 2^1 + 0 * 2^0$, $3 = 1 * 2^1 + 1 * 2^0$,..., $10 = 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0$, and so on);
2) write such coefficients (which are in $\{0, 1\}$) from the one used for the highest power to the one used for the lowest power;
3) the *binary representation* of n is such a string of zeroes and ones and can be written using the brackets and the index 2 (for example: $1 = (1)_2$, $2 = (10)_2$, $3 = (11)_2$,..., $10 = (1010)_2$, and so on).
Now we can give the following

Definition. *a) Every coefficient of the binary representation of n is called a bit (BInary digiT) of n.*
b) Every operation performed over a bit is called bit operation.

For example the sum $2 + 3 = 5$ can be written as $(10)_2 + (11)_2 = (101)_2$ and to perform it we use three bit operations.
We measure the speediness of an algorithm by counting the number of bit operations a computer needs to perform to implement the algorithm. The estimation of this quantity is called the *computational complexity* of the algorithm.
It is easy to see that to add two natural numbers n and m with $m < n$, we will execute at most $c \log n$ bit operations ($c > 0$ is a fixed constant and $\log n$ is essentially the number of digit of n). So we can say that *adding two numbers has computational complexity $O(\log n)$*.
The big-O notation is useful when we want to give upper bounds for a function. It is defined by

$$f(n) = O(g(n)) \text{ if and only if there is a } c > 0 \text{ such that } |f(n)| \leq c|g(n)|$$
$$\text{for every } n \text{ sufficiently large.}$$

Now we estimate the computational complexity of the euclidean algorithm. We need the following proposition.

Proposition. *The remainders of the euclidean algorithm have the property: $r_{j+2} < \frac{1}{2}r_j$.*

Proof. We consider two cases. Case 1: $r_{j+1} \leq \frac{1}{2}r_j$ holds. Then we have $r_{j+2} < r_{j+1} \leq \frac{1}{2}r_j$. Case 2: $r_{j+1} > \frac{1}{2}r_j$. Using the definition of r_j we get $r_j = 1 * r_{j+1} + r_{j+2}$ and so $r_{j+2} = r_j - r_{j+1} < \frac{1}{2}r_j$. ▪

For every two steps performed in the euclidean algorithm the remainder is divided by 2 and so, since it is ≥ 1, in the euclidean algorithm we perform at most $2 * [\log_2 a]$ divisions. Since it is easy to see that *every product or division has complexity $O(\log^2 a)$*, we therefore conclude that *the Euclidean algorithm has complexity $O(\log^3 a)$*.

Remark. Using a little sharper argument, see Koblitz [7], one can in fact prove that the estimate $O(\log^2 a)$ also holds for the Euclidean algorithm.

We conclude this section with one last important result.

Proposition. *Let $d = (a, b)$ with $a > b$. Then there are integers u and v such that $d = ua + vb$ and u and v can be computed with complexity $O(\log^3 a)$.*

Proof. Backtracking the equivalences of the euclidean algorithm and writing (at every step) d as a function of the remainders, we get the result. At every step we perform a sum and a product and so the total complexity is dominated by the complexity of the product.▪

Example. Using the same numbers of the previous example we get:

$$12 = 84 - 3 * 24 = 84 - 3 * (528 - 6 * 84) = 19 * 84 - 3 * 528 = 19(612 - 1 * 528) - 3 * 528 = 19 * 612 - 22 * 528 = 19 * 612 - 22(1752 - 2 * 612) = (-22) * 1752 + 63 * 612$$

and so $u = -22$, $v = 63$.

1.3. Congruence theory

We recall the

Definition. *Let a, b and n be three integers. We say that a is congruent to b modulo n (and we write $a \equiv b \pmod{n}$) if and only if n divides $a - b$ ($n|(a-b)$).*

Since the remainder r of a/n is between 0 and $n-1$, we can say that every integer a is congruent $\pmod{n}$ to one of the integers $0, 1, 2, \ldots, n-1$. Moreover, it is easy to see that two integers are congruent $\pmod{n}$ if and only if, when divided by n, their remainders are equal. So they are "mapped" to the same integer r in $\{0, 1, 2, \ldots, n-1\}$. This set is called the set of residue classes mod n and it is denoted by $\mathbb{Z}/_{n\mathbb{Z}}$.

Addition and multiplication in $\mathbb{Z}/_{n\mathbb{Z}}$ is defined just like it is for natural numbers except that, if the result m lands outside the set $\{0, 1, 2, \ldots, n-1\}$, then the final result r is the remainder of the integral division m/n. Consequently, m is congruent mod n to the final answer r. These rules satisfy the associative and commutative laws for addition and multiplication and the distributive law of multiplication over addition. This is a consequence of the fact that they are satisfied by natural number arithmetic and the following proposition.

Proposition. *Suppose $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$. Then $a + c \equiv b + d \pmod{n}$, $a - c \equiv b - d \pmod{n}$, and $ac \equiv bd \pmod{n}$.*

Proof. The proposition follows immediately if one observes that $a \equiv b \pmod{n}$ is equivalent to "there exists a suitable integer k such that $a = b + nk$". ▪

We now give a proposition which tells us when we can "cancel" a number c from both sides of a congruence relation. It is when c is relatively prime to the modulus, i.e. when $(c, n) = 1$. This occurs exactly when c and n have no common factors in their prime factorization.

Proposition. *Let $ac \equiv bc \pmod{n}$. Then $a \equiv b \pmod{\frac{n}{(c,n)}}$.*

Proof. Let $d = (c, n)$. Then, since $n|(a-b)c$, we get $\frac{n}{d}|(a-b)\frac{c}{d}$. But we have $(\frac{c}{d}, \frac{n}{d}) = 1$ and hence $\frac{n}{d}|(a-b)$. ▪

As a corollary one has the following statement (which can also be proved by using the properties of primes and the fundamental theorem of arithmetic): *if $n|bc$ and $(n, c) = 1$ then $n|b$.*

The use of such arithmetic allows us to work with "small" numbers (as one can see in the following example).

Example. Compute $5^{100} \pmod{101}$.
To calculate the power we use the "repeated squaring method":

$$\begin{gathered}5^5 = 3125 \equiv -6 \pmod{101} \\ 5^{10} = (5^5)^2 \equiv (-6)^2 \equiv 36 \pmod{101} \\ 5^{20} = (5^{10})^2 \equiv (36)^2 = 1296 \equiv -17 \pmod{101} \\ 5^{40} = (5^{20})^2 \equiv (-17)^2 = 289 \equiv -14 \pmod{101} \\ 5^{50} = 5^{40}5^{10} \equiv (-14) * 36 = -504 \equiv 1 \pmod{101} \\ 5^{100} = (5^{50})^2 \equiv 1^2 = 1 \equiv 1 \pmod{101}.\end{gathered}$$

As we will see later, in fact such a result is a consequence of Fermat's little theorem.

Now we show, using the euclidean algorithm, which elements of $\mathbb{Z}/_{n\mathbb{Z}}$ have multiplicative inverses.

Proposition. *A necessary and sufficient condition for an element a of $\mathbb{Z}/_{n\mathbb{Z}}$ to have multiplicative inverse is that $(a, n) = 1$. The inverse of a, which is denoted by a^{-1}, can be computed with complexity $O(\log^3 n)$.*

Proof. Let $d = (a, n)$. If $d > 1$ then a is not invertible (if there is such a b such that $ab \equiv 1 \pmod{n}$ we would have $d|(ab - 1)$ and so, since $d|a$, we would get $d|1$, a contradiction). If $d = 1$, then by the euclidean algorithm, we have a u and v in $\mathbb{Z}$ such that $ua + vn = 1$ and u and v are computable with complexity $O(\log^3 n)$. We now have u as the inverse of a (i.e. $u = a^{-1}$) in $\mathbb{Z}/_{n\mathbb{Z}}$. ▪

Linear congruences

Let a, b be two given residues modulo n and let x be an indeterminate residue class such that

$$ax \equiv b \pmod{n}. \tag{1}$$

Such a congruence is equivalent to the diophantine equation $ax = b + ny$ which has solutions (in $\mathbb{Z}$) if and only if $(a, n)|b$. Moreover, if $(a, n) = d$, equation (1) is equivalent to $\frac{a}{d}x \equiv \frac{b}{d} \pmod{\frac{n}{d}}$ in which $(\frac{a}{d}, \frac{n}{d}) = 1$. Hence the problem can be reduced to solving $a'x \equiv b' \pmod{n'}$ with $(a', n') = 1$.

Theorem. *The congruence $ax \equiv b \pmod{n}$, with $(a, n) = 1$, has one and only one solution, namely, $x \equiv a^{-1}b \pmod{n}$.*

Proof. Let $x \in \mathbb{Z}/_{n\mathbb{Z}}$. It is easy to see that the values of $ax \pmod{n}$, for x running over $\mathbb{Z}/_{n\mathbb{Z}}$, are all distinct numbers (since, having $(a, n) = 1$, $ax_1 \equiv ax_2 \pmod{n}$ implies $x_1 \equiv x_2 \pmod{n}$). Hence such values must be in n distinct residual classes and so there is only one value of $x \pmod{n}$ for which the congruence $ax \equiv b \pmod{n}$ holds. Such x is equal to $a^{-1}b$ since a has a multiplicative inverse. ▪

If we consider simultaneously the equations:

$$\begin{cases} a_1 x \equiv b_1 \pmod{n_1} \\ \dots \\ a_k x \equiv b_k \pmod{n_k} \end{cases} \quad \text{with } (a_i, n_i) = 1,\ i = 1, \dots, k \tag{2}$$

then we know that every congruence has one and only one solution in $\mathbb{Z}/_{n_i\mathbb{Z}}$. Can we find a solution satisfying all the equations simultaneously? The following theorem answers this question.

The Chinese remainder theorem. *If $(n_i, n_j) = 1$ for all $i \neq j$, (2) has one and only one solution $x \pmod{\prod_{i=1}^{k} n_i}$.*

Proof. Assume $k = 2$. From the first congruence we have $a_1 x = b_1 + n_1 t$ and, multiplying it by n_2, we get $n_2 a_1 x = n_2 b_1 + n_2 n_1 t$. Analogously from the second congruence it follows $n_1 a_2 x = n_1 b_2 + n_1 n_2 u$. Subtracting them we get $(n_2 a_1 - n_1 a_2) x = n_2 b_1 - n_1 b_2 + n_2 n_1 (t - u)$, and so the congruence

$$(n_2 a_1 - n_1 a_2) x \equiv n_2 b_1 - n_1 b_2 \pmod{n_1 n_2}$$

has only one solution mod $n_1 n_2$ since $(n_2 a_1 - n_1 a_2, n_1) = (n_2 a_1, n_1) = 1$ and $(n_2 a_1 - n_1 a_2, n_2) = (n_1 a_2, n_2) = 1$ imply, by the fundamental theorem of Arithmetic, that $n_2 a_1 - n_1 a_2$ is relatively prime to $n_1 n_2$. Hence we are done since it is easy to see that (2) and the previous congruence have the same solutions. Assume now that $k > 2$ and that the theorem is proved for $k - 1$. The first $k - 1$ congruences have simultaneously one and only one solution to the modulus $n_1 n_2 \dots n_{k-1}$ and so this case can be solved by re-applying the previous argument. ▪

1.4. Residues modulo a prime p

From what we have seen in the previous section, it is clear that in $\mathbb{Z}/_{p\mathbb{Z}}$, where p is a prime number, every element not equal to zero has a multiplicative inverse. This fact, together with the fact that modular arithmetic satisfies the normal laws of arithmetic gives us, trivially, the following

Theorem. *Suppose p is a prime number. Then $\mathbb{Z}/_{p\mathbb{Z}}$ is a field.*

Now we can prove a well known theorem which will be a fundamental tool in what follows.

Fermat's Little Theorem. *Let p be a prime and $a \in \mathbb{N}$ such that $a \not\equiv 0 \pmod{p}$. Then $a^{p-1} \equiv 1 \pmod{p}$.*

Proof. We claim that $\{1a, 2a, \dots, (p-1)a\}$ is a complete residue set modulo p. If not then ia and ja, for some $i \neq j$, would be in the same residue class $\pmod{p}$, i.e. $ia \equiv ja \pmod{p}$. In that case $p|(i - j)a$ implies $p|(i - j)$. This is a contradiction since $i < p$ and $j < p$ implies $i = j$. So $1a, 2a, \dots, (p - 1)a$ are congruent, in some order, to $1, 2, \dots, (p - 1)$ and hence the product of the elements in the first sequence is congruent $\pmod{p}$ to the product of the elements in the second one, i.e. $a^{p-1}(p-1)! \equiv (p-1)! \pmod{p}$. Then p divides $(a^{p-1}-1)(p-1)!$. But p does not divide $(p-1)!$ and so $p|(a^{p-1}-1)$. ▪

Remark: if we multiply the previous equality by a we get $a^p \equiv a \pmod{p}$ which holds also for $a \equiv 0 \pmod{p}$. This weak version of Fermat's little theorem holds for every $a \in \mathbb{Z}/_{p\mathbb{Z}}$.

Primitive residues

If n is not a prime, $\mathbb{Z}/_{n\mathbb{Z}}$ cannot be a field since there are some elements which have no inverse (e.g. 3 and 5 in $\mathbb{Z}/_{15\mathbb{Z}}$).

Definition. *If a is a residue class modulo n, we say that a is a primitive residue class modulo n if $(a, n) = 1$. These are the residue classes which have multiplicative inverses.*

We denote by $(\mathbb{Z}/_{n\mathbb{Z}})^*$ the set of primitive residues to the modulus n and by $\varphi(n)$ (*the Euler phi-function*) its cardinality.
Now we find some properties of the function $\varphi(n)$. Suppose $n = p^\alpha$. Then $(a, n) > 1 \Leftrightarrow p|a$. In other words $a \in \{p, 2p, 3p, \dots, p^{\alpha-1}p\}$. The remaining $p^\alpha - p^{\alpha-1}$ elements less than p^α are distinct primitive residues to the modulus p^α and so $\varphi(p^\alpha) = p^{\alpha-1}(p-1)$. In particular for $\alpha = 1$ one has $\varphi(p) = p - 1$. Now let $n = \prod_i p_i^{\alpha_i}$. We use the following

Proposition. *The Euler phi-function is multiplicative, i.e. $\varphi(mn) = \varphi(m)\varphi(n)$ for $(m, n) = 1$.*

Proof. Let $j \in \{0, \dots, mn-1\}$ and $j_1 \equiv j \pmod{m}$, $j_2 \equiv j \pmod{n}$. By the Chinese remainder theorem, for every pair (j_1, j_2) there exists only one $j \in \{0, \dots, mn-1\}$ such that $j_1 \equiv j \pmod{m}$, $j_2 \equiv j \pmod{n}$. Observe that $(j, mn) = 1$ if and only if $(j, m) = 1 = (j, n)$ and that the last condition is equivalent to $(j_1, m) = 1$ and $(j_2, n) = 1$.
The two sets $\{j \in \{0, \dots, mn-1\} : (j, mn) = 1\}$ and $\{(j_1, j_2) : j_1 \in \{0, \dots, m-1\}, j_2 \in \{0, \dots, n-1\}$ and $(j_1, m) = 1 = (j_2, n)\}$ are then in bijective correspondence and so have the same cardinality. I.e. $\varphi(mn) = \varphi(m)\varphi(n)$. ▪

If n is a natural number we can write $n = \prod_i p_i^{\alpha_i}$ with $p_i \neq p_j$ if $i \neq j$. Then

$$\varphi(n) = \prod_i p_i^{\alpha_i - 1}(p_i - 1) = n \prod_i \big(1 - \frac{1}{p_i}\big).$$

In the following proposition we calculate the computational complexity of $\varphi(n)$ for $n = pq$, p, q primes, assuming one knows p and q and viceversa. We will use it later when we will examine the computational complexity of the R.S.A. cryptosystem.

Proposition. *Let $n = pq$ with p and q prime numbers. Then, using p and q we can compute $\varphi(n)$ with computational complexity $O(\log n)$ and, using $\varphi(n)$, we can compute p and q with computational complexity $O(\log^3 n)$.*

Proof. If n is even then we can say that $p = 2$ and hence $q = \frac{n}{2}$. So $\varphi(n) = \frac{n}{2} - 1$. If n is odd then $\varphi(n) = (p-1)(q-1) = n + 1 - (p + q)$. Hence, knowing p, q and n, we can obtain

$\varphi(n)$ with a sum and a subtraction (complexity $O(\log n)$). Otherwise, knowing $\varphi(n)$ and n, we get $p+q = n+1-\varphi(n)$. So we have that n and $p+q$ are known quantities and hence we can obtain p and q as the solutions of $x^2-(p+q)x+n$. Letting $2b = p+q$, we obtain $p, q = b \pm \sqrt{b^2-n}$ and the total computational complexity is $O(\log^3 n)$ (the complexity of computing the square root of a number). ▪

Now we can prove a generalization of Fermat's little theorem.

Euler-Fermat Theorem. *Let* $(a, n) = 1$. *Then* $a^{\varphi(n)} \equiv 1 \,(\mathrm{mod}\ n)$.

Proof. Suppose $n = p^k$. If $k = 1$, then $\varphi(n) = p-1$ and the theorem follows from Fermat's little theorem. Now suppose the theorem has been verified to be true for $k-1$. We now show that this implies the theorem holds for k.
Since the theorem holds for $k-1$, we know that there is a b such that $a^{p^{k-1}-p^{k-2}} = 1+p^{k-1}b$. Taking the pth-powers of both sides and using Newton's formula, we have

$$(a^{p^{k-1}-p^{k-2}})^p = (1+p^{k-1}b)^p = \sum_{j=0}^{p} \binom{p}{j} (p^{k-1}b)^j.$$

Since $p | \binom{p}{j}$ for every $j \neq 0, p$, we obtain

$$\sum_{j=0}^{p} \binom{p}{j} (p^{k-1}b)^j = 1 + p^k \sum_{j=1}^{p} a_j.$$

Hence $a^{p^k-p^{k-1}} \equiv 1 \,(\mathrm{mod}\ p^k)$, i.e. $a^{\varphi(p^k)} \equiv 1 \,(\mathrm{mod}\ p^k)$, and so the theorem holds for k. By the induction principle, the theorem holds for every prime power.
Now suppose $n = p^\alpha q^\beta$, where p and q are distinct primes and α and β are natural numbers. We know that $a^{\varphi(p^\alpha)} \equiv 1 \,(\mathrm{mod}\ p^\alpha)$ and $a^{\varphi(q^\beta)} \equiv 1 \,(\mathrm{mod}\ q^\beta)$. Now we take the $\varphi(q^\beta)$-th power of the first one and the $\varphi(p^\alpha)$-th power of the second one, thus obtaining

$$\begin{cases} (a^{\varphi(p^\alpha)})^{\varphi(q^\beta)} \equiv 1 \,(\mathrm{mod}\ p^\alpha) \\ (a^{\varphi(q^\beta)})^{\varphi(p^\alpha)} \equiv 1 \,(\mathrm{mod}\ q^\beta) \end{cases}$$

which, by the fundamental theorem of Arithmetic, implies $a^{\varphi(p^\alpha)\varphi(q^\beta)} \equiv 1 \,(\mathrm{mod}\ p^\alpha q^\beta)$. The theorem follows due to the multiplicativity of φ. The general case follows by using again the fundamental theorem of Arithmetic and repeating the above argument as many times as necessary. ▪

We observe that, using the Euler-Fermat theorem, we can easily compute the quotient of two residues as follows: $ab^{-1} \equiv ab^{\varphi(n)}b^{-1} \equiv ab^{\varphi(n)-1} \,(\mathrm{mod}\ n)$.
The following corollary will be needed for proving the correctness of the deciphering algorithm in the R.S.A. cryptosystem.

Corollary. *Let* n *be a square-free number and* d *and* e *natural numbers such that* $\varphi(n) | (de-1)$. *Then*

$$a^{de} \equiv a \,(\mathrm{mod}\ n) \quad \textit{for every } a \in \mathbb{Z}/_{n\mathbb{Z}}.$$

Proof. We know that $n = \prod_i p_i$, where p_i are prime numbers and $p_i \neq p_j$ if $i \neq j$. Since $\varphi(n)|(de-1)$, we can write $(p_i - 1)|(de-1)$ for every p_i that divides n. So, using Fermat's little theorem, we get for every $p_i|n$ and for every $(a, p_i) = 1$ that $a^{p_i - 1} \equiv 1 \,(\text{mod}\ p_i)$. Since $de - 1 = k_i(p_i - 1)$, we obtain

$$a^{de-1} \equiv 1 \,(\text{mod}\ p_i) \quad \text{for every } p_i|n \text{ and for every } (a, p_i) = 1.$$

Hence, for every $a \in \mathbb{Z}/_{n\mathbb{Z}}$, we have $a^{de} \equiv a \,(\text{mod}\ p_i)$ for every p_i dividing n. By the fundamental theorem of Arithmetic, we know that each p_i appears in the factorization of $a^{de} - a$ and so we get $a^{de} \equiv a \,(\text{mod}\ n)$ for every $a \in \mathbb{Z}/_{n\mathbb{Z}}$. ▪

2. Cryptography

2.1. Introduction

Cryptography is the study of methods that can be used to send information in disguised form so that only the intended recipients can remove the disguise and read the message. We denote, respectively, by

$$\mathcal{P} = \{\text{plaintext}\} \quad \mathcal{C} = \{\text{ciphertext}\}$$

the messages in clear form and in disguised form.
A *cryptographic transformation* is an injective function $f : \mathcal{P} \to \mathcal{C}$ so that we have an inverse function f^{-1} defined on $f(\mathcal{P})$

$$\mathcal{P} \xrightarrow{f} f(\mathcal{P}) \xrightarrow{f^{-1}} \mathcal{P}.$$

The function f needs to be injective since we don't want any ambiguity in deciphering the message transmitted. We will call

$$f : \text{ the } \textit{enciphering function} \quad \text{and} \quad f^{-1} : \text{ the } \textit{deciphering function.}$$

A *cryptosystem* is a quadruplet $(\mathcal{P}, \mathcal{C}, f, f^{-1})$.

2.2. A short history

One of the first cryptographic methods was used by the Roman emperor Julius Caesar. Using modern terminology we say that the Roman used a method based on modular arithmetic for the modulus n (where n is the number of the letters in the alphabet of the language the message is being sent in). In fact they used a bijective correspondence between the alphabet and the set $\mathbb{Z}/_{n\mathbb{Z}}$ and they enciphered the messages by a "shift" transformation of a fixed quantity m (they added $m \,(\text{mod}\ n)$ to every number corresponding to each letter).

For example, if $n = 21$ and $m = 5$, to cypher the message P, which, for the sake of simplicity, we consider it to be one letter, they used the enciphering function $C \equiv P + 5 \pmod{21}$ and the deciphering function $P \equiv C - 5 \pmod{21}$.
It is easy to see that a frequency analysis, based on the letters that appear in the message, would allow us to break the system (i.e. one can read P from C by reconstructing the deciphering function).

Some variations of the previously described cryptosystem can be easily developed: for example sending messages of two letters using 22 letters (21 letters and a blank). Then $P = \alpha\beta$ can be translated to the number $22x + y \in \{0, \ldots, 483\}$, $x, y \in \{0, \ldots, 21\}$, where x and y are, respectively, the numbers corresponding to the symbols α and β. In such a way each letter can be seen as a 22-base digit and every two-letter message is enciphered using a two-digit number in base 22.
Such an argument can be easily extended to the case in which every message is built with k letters in an alphabet with n symbols (the messages are enciphered using the integers between 0 and $n^k - 1$).

In the 16-th century, again using a technique based on congruences, Vigenère proposed a variation of the previous system which is more difficult to break. Taking k-letter blocks (a vector of $(\mathbb{Z}/_{n\mathbb{Z}})^k$) the enciphering transformation is performed by shifting it with a k-letter "password" (one adds to the message $P \in (\mathbb{Z}/_{n\mathbb{Z}})^k$ a fixed vector $B \in (\mathbb{Z}/_{n\mathbb{Z}})^k$).
As in the previous case, this system can be broken by a frequency analysis on the arithmetic progressions modulo k. Moreover one can also reconstruct the "password" B.

In 1931, Hill used for f the matrix-product by an invertible matrix. Letting $\mathcal{P} = \mathcal{C} = (\mathbb{Z}/_{n\mathbb{Z}})^k$, he considered an invertible matrix A whose elements are in $\mathbb{Z}/_{n\mathbb{Z}}$. The enciphering transformation is then obtained by multiplying A and $P \in \mathcal{P}$. The deciphering transformation is obtained using the product of $C \in \mathcal{C}$ with A^{-1}.
Using the linear algebra to the modulus n, Hill's system can be broken provided we are able to know a certain number of pairs (P, C).

As one can see in the previous examples, early cryptosystems exploited only elementary algebra and number theory. This was essentially the situation until the seventies.

2.3. Classic cryptography

All the examples seen already are from *classic cryptography* (or *private key cryptography*). Anyone who has enough informations to encipher the message, can automatically (or with a little effort), decipher it. Hence every pair of users, who wish to secretly communicate, has to exchange their enciphering and deciphering keys in a secure way (e.g. using a courier they trust). In the previous examples the keys were:

	encipher K_E	decipher K_D
Caesar	m	$-m$
Vigenère	vector v	$-v$

Hill	matrix A	A^{-1}

The classical methods also have the following property: the *enciphering and the deciphering are computationally equivalent* (i.e. their computational complexities are equal or of the same order). Such remarks suggests to us a classification based on the computational complexity.

Classic cryptography: the cryptosystems in which (after knowing the enciphered information) the deciphering transformation can be implemented in approximatively the same time needed to encipher (in other terms: the above mentioned computational complexities have the same order).

If a cryptosystem has a deciphering complexity that has a bigger order than its enciphering complexity, we can say that such a cryptosystem has the potential to be transformed into a *public key* cryptosystem (we will be more precise about that later). For example, if the enciphering transformation is $O(\log^{c_1} B)$, where B is the main system parameter, and the deciphering one is $O(B^{c_2})$, then we have the potential for a public key system.

2.4. Public Key Cryptography

Public Key Cryptography was invented by *Diffie* and *Hellman* [4] in 1976 and its definition can be given as follows:

In a *public key cryptosystem* one knows only how to encipher and *cannot use the enciphering key to construct the deciphering key* without a prohibitively lengthy computation.

In other words $f : \mathcal{P} \to \mathcal{C}$ can be easily computed once one knows K_E (the enciphering key) but to compute $f^{-1} : f(\mathcal{P}) \to \mathcal{P}$, without having additional information (i.e. without knowing the deciphering key K_D), is very hard.
Such functions are called *one-way functions* (or *trap-door functions*).

At present there are no known functions which have been *proved* to be one-way functions (even if there are some conjectured one-way functions which are used in practical applications). It is conceivable, from a theoretical point of view, that, in the future, some transformation will be proved to be one-way. We also need to remark that, since the performance of microprocessors is improving dramatically, functions that, in 2000, are considered to be one-way in 2010 (or even in the 2003) might not have such a status.

An application of a general public key cryptosystem is a *digital signature* algorithm, which we now explain.

Let A and B be two system users. We know that f_A and f_B (their enciphering functions) are public and their deciphering functions f_A^{-1} and f_B^{-1} are secret. We are assuming for this application that f_A and f_B are onto functions.
If A wishes to send the message P to B, he has to send the enciphered message $f_B(P)$ and, to certify his identification, he signs the message by sending also $f_B(f_A^{-1}(signature_A))$.

B deciphers the message using f_B^{-1} (he is the only one who knows it) and he obtains $f_B^{-1}(f_B(P)) = P$. To be sure that it was sent by A, B checks the identification appended to the message by applying $f_A f_B^{-1}$; he gets $f_A f_B^{-1}(f_B f_A^{-1}(signature_A)) = signature_A$. This system works since only A was able to sign the message in such a way (A is the only user who knows f_A^{-1}).
Finally we remark that a public-key system can be used to build a secure key-exchange method for a classic system (this can be useful since, in general, classic systems can use smaller keys than the ones used in public-key systems and hence the amount of time needed to cypher and to decipher is, in practice, lower).

2.5. The R.S.A. cryptosystem

One of the most important examples of a conjectured one-way function (and also the first public key cryptosystem to be developed) was furnished by *Rivest*, *Shamir* and *Adleman* [2] in 1978. Their approach was to use the differences between the computational complexity involved in locating large primes and the computational complexity involved in integer factorization.

The R.S.A. system works as follows: *each user* has to

1) randomly choose two large primes p, q (nowadays 300 decimal digits is a safe choice) and set $n = pq$;
2) compute $\varphi(n) = (p-1)(q-1) = \#(\mathbb{Z}/_{n\mathbb{Z}})^*$;
3) randomly choose an integer e such that $1 \le e \le \varphi(n)$ and $(e, \varphi(n)) = 1$ (e is coprime with $\varphi(n)$);
4) compute $d \equiv e^{-1} \pmod{\varphi(n)}$ (he can do it easily since he knows p and q).

Remarks: a) The *random* choice is a fundamental request if one wishes to obtain a secure cryptosystem. We will discuss in section 2.7 below how one goes about finding large prime numbers.
b) The calculation of d in item 4) can be obtained during the verification that $(e, \varphi(n)) = 1$ in item 2) (as we have seen in the first part of these notes).

For *every user* we define the following quantities:
Public key: the public key is disclosed to all the other users so they can use it to communicate with the key's owner,

$$K_E = (n, e);$$

Private key: d is the private key since it allows its owner to decipher the received messages,

$$K_D = (n, d);$$

Enciphering function: $f : \mathbb{Z}/_{n\mathbb{Z}} \to \mathbb{Z}/_{n\mathbb{Z}}$ is given by $f(P) \equiv P^e \pmod{n}$;
Deciphering function: $f^{-1} : \mathbb{Z}/_{n\mathbb{Z}} \to \mathbb{Z}/_{n\mathbb{Z}}$ is given by $f^{-1}(C) \equiv C^d \pmod{n}$.

To verify that the system works, we look at an example; a system which has only two users:

	Anna	**B**ob
public	$K_{E_A} = (n_A, e_A)$	$K_{E_B} = (n_B, e_B)$
private	$K_{D_A} = (n_A, d_A)$	$K_{D_B} = (n_B, d_B)$.

Anna wishes to send a message P to Bob: she uses Bob's public key to compute a C such that

$$C \equiv P^{e_B} \,(\mathrm{mod}\ n_B)$$

and sends C to Bob. Bob receives C, uses his deciphering key (and he is the *only one who knows it*!) and computes

$$C^{d_B} \,(\mathrm{mod}\ n_B) \equiv P^{e_B d_B} \,(\mathrm{mod}\ n_B).$$

Since $e_B d_B \equiv 1 \,(\mathrm{mod}\ \varphi(n_B))$, by the Euler-Fermat theorem, if $(P, n_B) = 1$, and by its corollary, if $(P, n_B) > 1$, we have

$$C^{d_B} \,(\mathrm{mod}\ n_B) \equiv P \,(\mathrm{mod}\ n_B)$$

and hence Bob can read Anna's message in its plain form.

If there are more than two users, a file which contains all the users' public keys should be sent to all of them, thus allowing each one to send messages to every other system user.

2.6. System's safety

The main characteristic which assures the system's safety is that in order to *break* the system (i.e. find f^{-1} without knowing K_D) one *presumably* has to be able to compute d knowing only e and n. The only obvious way to do that is to be able to find $\varphi(n)$ knowing only n and one number relatively prime to $\varphi(n)$. This one number relatively prime to $\varphi(n)$ does not restrict very much what $\varphi(n)$ may be. Trying to find $\varphi(n)$ by using n (and without knowing its factorization in primes) seems to be a difficult problem. Recall that, in the first part of such notes, we saw that if one knows p and q, one can calculate $\varphi(n)$ with $O(\log n)$ bit operations and if one knows $\varphi(n)$, one can calculate p, q with $O(\log^3 n)$ bit operations; or to put it more succinctly:

to know $\varphi(n)$ is computationally equivalent to knowing the factorization of n.

Hence breaking R.S.A., by all appearances, seems to depend on being able to find the factorization of n in a reasonable amount of time. It is comparatively easy to build a large n with large prime factors, as opposed to trying to factor a large n with large prime factors. Trying to find efficient ways of factoring numbers that does not involve the prohibitively long process of simply checking all possible factors has proved to be a hard problem for mathematicians to solve.

2.7. Some information on primality and factorization algorithms

One has to be a little careful, however, in choosing $n = pq$ in practice. The reason is that certain choices of p (or q) which have a "special form" (for example, a classical "special form" is a prime of the type $2^n - 1$ or $2^n + 1$) should not be used in cryptography since, in such cases, there do exist fast factorization algorithms.

Now we talk a little bit about how to locate large prime numbers. We give just an overview since the mathematical tools needed to give an exhaustive presentation are too complicated for these notes. First of all we have the

Definition. *A primality algorithm is a finite set of computations which (without searching for the factors of n) proves that n is prime.*

Some of the best known primality tests are based on certain congruence properties (which, in some sense, can be seen as a "converse" of Fermat's little theorem) and, even if the easiest ones are not able to obtain anything more than a probabilistic evaluation of the primality of a given integer n, one of them (the *Miller-Rabin* test) can, under the assumption of the Generalized Riemann Hypothesis, *prove* the primality of n with computational complexity $O(\log^5 n)$.

Remark. Some of the most used mathematical softwares tests the primality of an integer by an algorithm based on Fermat's little theorem. Essentially they test if the number n verifies $a^{n-1} \equiv 1 \pmod{n}$ for a in some subset of $\mathbb{Z}/_{n\mathbb{Z}}$. Since it can be proved that, *if n is a composite number and there exists a $b \in \mathbb{Z}/_{n\mathbb{Z}}$ such that $b^{n-1} \not\equiv 1 \pmod{n}$, then $a^{n-1} \not\equiv 1 \pmod{n}$ for at least $1/2$ of the numbers $a \in \mathbb{Z}/_{n\mathbb{Z}}$*, we can, after the first used a, say that n is prime with probability greater than $1 - 1/2$ or that n is a composite number that verifies $a^{n-1} \equiv 1 \pmod{n}$ for such a. A composite number n such that $a^{n-1} \equiv 1 \pmod{n}$ for every $a \in \mathbb{Z}/_{n\mathbb{Z}}$ is called a *Carmichael* number; for example $561 = 3 * 11 * 17$ is a Carmichael number. By repeating the above procedure k-times, we obtain: if n is not a Carmichael number then n is prime with probability greater than $1 - (1/2)^k$.
For large k, the probability that n is prime is approximately 1. And this seems to be very good... but one has to prove that n is not a Carmichael number! However, in 1994, *Alford*, *Granville* and *Pomerance* proved that the Carmichael numbers are infinite and so the number of times the "Fermat" test "fails" can be large.
In fact the problem is that the set of congruences used is too weak to prove primality; one has to use stronger conditions (for example, no composite numbers verify the congruences used in the Miller-Rabin test for all the $a \in \mathbb{Z}/_{n\mathbb{Z}}$, i.e., in this case, an analogue of the Carmichael numbers does not exist).
So, when you are using mathematical software, please pay attention to the description of the algorithm used to detect primes. If it has implemented a so-called "pseudo-prime test" or a "probable-prime test" you should know that the answer could be wrong.

Without assuming any analytic hypothesis, the best primality algorithm nowadays known was invented in 1983 by *Adleman-Pomerance-Rumely* [1] and has a computational complexity (proved using some analytic number theory techniques) less than

$$O((\log n)^{c \log\log\log n}), \text{ where } c > 0 \text{ is a fixed constant,}$$

i.e. "almost-polynomial" in $\log n$.

From the point of view of trying to factor large numbers, the trivial *factorization algorithm* (an algorithm which furnishes at least a prime factor of n) consists in trying to divide n by all the primes less than or equal to its square root. This algorithm has computational complexity $O(\sqrt{n}\log n)$.
In the last twenty years many researchers have tried to solve the factorization problem using sophisticated techniques, but with our current knowledge this problem seems to be intrinsically harder than the primality one.
In fact, the best known factorization algorithm (*Pollard* [9] developed the main idea in 1993 and then other authors generalized it, see Lenstra-Lenstra's book [8]) does not have a computational complexity comparable with the primality algorithms. This method is based on certain properties of number fields (which are too complicated to write in these short notes) and it has conjectured computational complexity

$$O(e^{c\sqrt[3]{\log n(\log\log n)^2}}), \text{ where } c > 0 \text{ is a fixed constant,}$$

which is higher than the computational complexity of the primality problem.

As an example, using a computer which anyone can buy in a computer-store, one can build an integer (product of two "general" primes) of 140 decimal digits in a few seconds. To factorise the same number, using some parallel computers, the computer-time needed is nearly one month! So it's a good idea to use fairly large numbers; at present an integer (product of two general primes) with 220 decimal digits is considered a secure choice for a personal use (but professional use requires a larger number).

We wish to note that, since proving the existence of high lower bounds for the general case of the factorization problem is an open problem, the R.S.A. cryptosystem is only conjectured to be a public key cryptosystem.

Much more information about primality and factorization algorithms, can be found in books written by Koblitz [6], Riesel [10] and Cohen [3].

3. The Discrete logarithm problem

Now we see another topic which can be used to build a (conjectured) one-way function: the *Discrete Logarithm Problem.* The situation is the following: we take a finite field $\mathbb{F}_q$, where q is a prime number p or a prime power p^f, a and b are in $\mathbb{F}_q$ and we assume that $y \in \mathbb{F}_q$ is a power of b (i.e. there exists a x such that $y = b^x$). We call x the *discrete logarithm* of y.

Now the *discrete logarithm problem* is: knowing only b, y and q, compute x.

If we were in $\mathbb{R}$, we know that the exponential and the logarithm functions can be calculated with the same complexity, but in the finite field case, we only know that b^x can be calculated

in a "fast way" (via the repeated squaring method), but the discrete logarithm problem is conjectured to be intrinsically harder.
As in the previous two sections, we now explain how to use the discrete logarithm problem to build a cryptographic system and then we will discuss its security.

3.1. Diffie-Hellman key-exchange method

As we have remarked above, one application of a public-key system is to exchange the keys of a classical cryptosystem. Diffie and Hellman used the discrete logarithm problem to build a safe method to do that. We assume that Anna and Bob agree *publicly* on a prime p and on a non-zero element $g \in \mathbb{F}_p$ (which is obviously a generator of $\mathbb{F}_p^*$). Now Anna chooses a random element a, $1 \leq a \leq p-1$, keeps it secret, computes g^a and publishes it. Analogously Bob chooses a random element b, $1 \leq b \leq p-1$, keeps it secret, computes g^b and publishes it. The *secret key* they can use to exchange their coded messages is g^{ab}. In fact both Anna and Bob can compute it since $(g^a)^b = g^{ab} = (g^b)^a$, but a third person who wishes to break the system has to solve the so-called

Diffie-Hellman problem: given a prime p and $g, g^a, g^b \in \mathbb{F}_p$, find g^{ab}.

It is easy to see that anyone who is able to solve the discrete logarithm problem can solve the Diffie-Hellman problem, but the converse is not known. So it is possible, from a theoretical point of view, that someone could break the Diffie-Hellman key-exchange system without being able to solve the discrete logarithm problem. In fact the two problems are conjectured to be equivalent even though, at the present state of knowledge, such an equivalence has not been proved. By the way, for practical purposes, a key-exchange by this method is considered to be safe.

3.2. Massey-Omura and ElGamal methods

As in the previous paragraph we assume that $\mathbb{F}_q$ is fixed and public.
In the *Massey-Omura method* every user, as in the R.S.A. cryptosystem, has:
1) to choose a random integer e such that $0 \leq e \leq q-1$ and $(e, q-1) = 1$;
2) to compute $d \equiv e^{-1} \pmod{q-1}$ which has to be kept *secret*.
Assuming that Anna wants to send a message P to Bob, the transmission runs as follows (every calculation has to be done in $\mathbb{F}_q$):
a) Anna, after the calculation of P^{e_A}, sends it to Bob;
b) Bob doesn't know d_A, so he computes $(P^{e_A})^{e_B}$ and sends it back to Anna;
c) Anna applies d_A to what she has received, thus obtaining P^{e_B}; then she sends it to Bob;
d) finally Bob is able to read the original message P by applying d_B to P^{e_B}.
In some sense every user applies a "padlock" that no one else can remove. The drawback of such an idea is the fact that it has to be used together with a strong signature method; otherwise every user cannot be sure of the identity of its correspondent and, for example, a third person could feign to be Bob sending $(P^{e_A})^{e_C}$ to Anna and, after Anna has removed her "padlock", the third party could read a message not originally sent to him.

In the *ElGamal method* the system's administrator has to choose an element $g \in \mathbb{F}_q$ (usually a generator) which is public.
Then every user has to choose and keep secret an integer a, $0 < a < q-1$. The user then computes g^a which will be his *public key*. Every other user who wishes to send a message P to Anna chooses a random integer k and then sends the pair (g^k, Pg^{ak}) to her (g^a is Anna's public key). She is the only person who is able to decipher the message by multiplying the second element of the pair by $g^{k(q-1-a)}$.

Both the Massey-Omura and the ElGamal method, can be broken easily by anyone who knows how to solve the discrete logarithm problem in a "fast" way.

3.3. Some information on the complexity of the discrete logarithm problem

Here the unit measure is the number of digits of q (which is the cardinality of $\mathbb{F}_q$). The situation is similar to what we saw for the primality and factorization case.
In fact, the best known algorithm (the so-called *index-calculus algorithm*) has sub-exponential complexity of the form
$$O(e^{c\sqrt{\log q}}),$$
where q has a "general" form and $c > 0$ is a suitable constant. Such an estimate has to be compared with the complexity $O(\log^3 q)$ of the enciphering/deciphering transformation (which is an exponentiation in $\mathbb{F}_q$).
If $q-1$ has only small prime factors, than another algorithm (the *Silver-Pohlig-Hellman* algorithm) is faster, and hence, to build a safe cryptosystem, one should use sufficiently large (at least 512 binary digits) integers of general form.

We remark that cryptosystems based on the discrete logarithm problem can be considered to be a public key cryptosystem only using the conjectural point of view; in fact:
1) at present no results are known about high lower bounds for the complexity of solving the discrete logarithm problem in the general case;
2) system safety depends on the intractability of the Diffie-Hellman problem which is only conjecturally equivalent to the discrete logarithm problem.

As for primality and factorization, Koblitz's, Riesel's and Cohen's books, see respectively [6], [10] and [3], are three excellent references.

3.4. Some words on the elliptic analogue of the discrete logarithm problem

This paragraph is, of necessity, very brief. An excellent presentation of the elliptic curve cryptosystems is given in Koblitz's, [6],[7], books (*Koblitz* was also the first who used the elliptic curves defined over finite fields to do cryptography). As we said in the introduction, one can use a well-known geometrical object (the elliptic curves defined on a fixed finite field $\mathbb{F}_q$) to build a public key cryptosystem by using an analogue of the discrete logarithm problem. In fact the main property used is that the set of points of an elliptic curve $E(\mathbb{F}_q)$ (i.e. the pairs of elements of $\mathbb{F}_q$ which lie on the curve) is, with some suitable operations, a finite commutative group. But now, even if $\mathbb{F}_q$ is fixed, the cardinality of the set $E(\mathbb{F}_q)$ can, varying the elliptic curve, run over a suitable range (which is given by *Hasse's Theorem*). So we have, theoretically, a more flexible tool in which we can build a suitable analogue of the discrete logarithm problem (formally it works as in the classic case; the only difference is that it is formulated in terms of the points of $E(\mathbb{F}_q)$ instead of the elements of $\mathbb{F}_q$). In fact such an approach has both pros (since one has the advantage that one can use a wide variety of curves) and cons since, recently, some researchers (*Smart* and *Semaev*) have independently proved that, roughly speaking, in many cases the elliptic analogue of the discrete logarithm problem can be solved in polynomial time. Such results imply that one has to *carefully* choose the elliptic curve needed. The general case is still considered to be a hard problem, essentially equivalent, for a fixed curve, to the classical discrete logarithm problem.

References

[1] L.M. Adleman, C. Pomerance, R. Rumely - *On distinguishing prime numbers from composite numbers* - Annals of Math. **117** (1983), 173-206.
[2] L.M. Adleman, R.L. Rivest, A. Shamir - *A method for obtaining digital signature and public key cryptosystems* - Comm. of ACM **21** (1978), 120-126.
[3] H. Cohen - *A Course in Computational Algebraic Number Theory* - Springer 1994.
[4] W. Diffie, M.E. Hellman - *New directions in cryptography* - IEEE Trans. Information Th. **22** (1976), 644-654.
[5] D. Kahn - *The Codebreakers, the Story of Secret Writing* - Macmillan 1967.
[6] N. Koblitz - *A Course in Number Theory and Cryptography* - Springer 1987.
[7] N. Koblitz - *Algebraic Aspects of Cryptography* - Springer 1998.
[8] A.K. Lenstra, H.W. Lenstra (eds) - *The development of the Number Field Sieve* - Springer L.N. 1993.
[9] J.M. Pollard - *Factoring with cubic integers* - in “The development of the Number Field Sieve”, A.K. Lenstra and H.W. Lenstra (eds), Springer L.N. 1993, 4-10.
[10] H. Riesel - *Prime Numbers and Computer Method for factorization* - II ed. - Birkhäuser 1994.

Alessandro Languasco
Dipartimento di Matematica Pura e Applicata
Università di Padova
Via Belzoni 7
35131 Padova, Italy
e-mail: languasco@math.unipd.it