APPLICATIONS OF NUMBER THEORY IN CRYPTOGRAPHY

ZIHAO JIANG

Abstract. Cryptography is the practice of hiding information, converting some secret information to not readable texts. Applications of cryptography include military information transmission, computer passwords, electronic commerce, and others. This paper aims to introduce the reader to applications of number theory in cryptography. We will briefly talk about the idea of encryption and public key cryptosystem in the context of algebra and elementary number theory.

Contents

1. Topics in Elementary Number Theory

Before start studying of cryptography, here is some background that shall be required. We start with time estimates for algorithms.

Numbers in different bases. A nonnegative integer n written in *base* b is a notation for n of the form $(d_{k-1}d_{k-2}...d_1d_0)_b$, where the d_i's are called *digits*. This notation simply means $n = d_{k-1}b^{k-1} + d_{k-1}b^{k-2} + ... + d_1b + d_0$. We shall omit the parentheses and subscript in the case of the usual decimal system (b=10). The simplest and probably the most applied is the binary system with numbers written in base 2.

Number of digits. It is easy to see that an integer n satisfying $b^{k-1} \leq n < b^k$ has k digits in base b. By the definition of logarithms, this gives the following formula for the number of base b digits:

$$\text{number of digits} = \lfloor \log_b n \rfloor + 1 = \lfloor \tfrac{\log n}{\log b} \rfloor + 1.$$

Bit operations. In order to evaluate the time needed for a specific algorithm, we need to know what kind of arithmetic operations are involved and how much time each takes to implement. Here we introduce the concept of a *bit operation*, which is the operation of adding two one-digit binary numbers together, for example,

$1+1=10$, $1+0=1$, $0+1=1$, $0+0=0$ and their counterparts in subtraction are also bit operations. Let A and B be integers and m and n their number of binary digits respectively. Suppose $m > n$, it will take m bit-operations to add these two numbers. During the addition, we need first do n adding operations to get the partial sum of B and the last n digits of A. Then we need to add up the first $m-n$ digits of A with $(m-n)$ 0's. So in total we need m bit-operations. When we are speaking of estimating the "time" it takes to accomplish something, we mean finding an estimate for the number of bit operations required. In these estimates, we usually just ignore the time required for "bookkeeping", like remembering the carries generated during the process of addition or other logical steps other than bit-operation, since such time is asymptotically insignificant.

Similarly subtraction of A and B takes m bit-operations, while multiplication and division of A and B take roughly mn bit-operations.

In order to estimate the upper time bound of some algorithms, we now introduce a very convenient notation

Definition 1.1. Let $f, g : \mathbf{N}^r \to \mathbf{R}$ be functions of all $r-$tuples of positive integers, and suppose that there exist constants B and C such that whenever all of the n_j are greater than B the two functions are defined and positive, and $f(n_1, n_2, ..., n_r) < Cg(n_1, n_2, ..., n_r)$. Then we say that f is *bounded* by g and write $f = O(g)$. Note the "=" in the notation $f = O(g)$ should be thought of as more like a "<" and the big-O should be thought of as meaning "some constant multiple."

Example 1.2. When written in binary form, an integer A has $[\frac{\log A}{\log 2}] + 1$ digits, which is roughly $\frac{\log A}{\log 2}$. So it takes roughly $\frac{\log A}{\log 2}$ bit-operations to add and $(\frac{\log A}{\log 2})^2$ to multiply. If we measure time in bit operations (which will be kept as convention in this paper), we can see by definition then, it takes $O(\log A)$ time to add and $O((\log A)^2)$ time to multiply.

Now we shift gears towards topics in elementary number theory.

Definition 1.3. Given integers a and b, we say that *a divides b* (or b is divisible by a) and we write $a|b$ if there exists an integer d such that $b = ad$. In that case we call a a divisor of b.

The following properties of divisibility are easy to verify:
(a) If $a|b$ and c is any integer, the $a|bc$;
(b) If $a|b$ and $b|c$, then $a|c$;
(c) If $a|b$ and $a|c$, then $a|b \pm c$.

The Fundamental Theorem of Arithmetic states that any natural number n can be written uniquely as a product of prime numbers. By this theorem, we can derive directly two more properties of divisibility:
(d) If a prime number p divides ab, then either $p|a$ or $p|b$.
(e) If $m|a$ and $n|a$, and if m and n have no divisors greater than 1 in common, then $mn|a$.

Another powerful consequence of The Fundamental Theorem of Arithmetic is that if a number $n = p_1^{\alpha_1} p_2^{\alpha_2} ... p_n^{\alpha_n}$, then it has in total $(\alpha_1 + 1)(\alpha_2 + 1)...(\alpha_r + 1)$ different divisors, because for every prime factor we can pick its 0^{th} up to n^{th} order to be included in the product.

Definition 1.4. The *greatest common divisor* of a and b, denoted $d = g.c.d(a, b)$, is an integer which satisfies the following two properties
(1) $d|a$ and $d|b$, and also
(2) if $d'|a$ and $d'|b$, then $d'|d$.

By definition, if there are two greatest common divisors of a and b then, they divide each other and are therefore equal. Thus, the greatest common divisor is unique if it exists.

The Euclidean algorithm. If we are working with large integers, it's unlikely that we will know their prime factorizations. In fact, factoring large integers is an important and difficult problem. There's a quick way to find $g.c.d.(a, b)$ called Euclidean algorithm.

The algorithm works as follows. Assume that we have two integers a and b, $a > b$, we first divide b into a and write the quotient q_1 and the remainder r_1, that is $a = q_1 b + r_1$. Perform the second division dividing b by r_1, $b := q_2 r_1 + r_2$. And keep on until we get some r_{n+1} divides r_n.

Proposition 1.5. *The Euclidean algorithm always gives the greatest common divisor in a finite number of steps. In addition, for $a > b$, the runtime of the algorithm is $O(\log^3(a))$*

Proof. Firstly, the remainders are strictly decreasing from one step to the next, and so must eventually reach zero. Use the second part of the definition of the $g.c.d.$, if any number divides both a and b, it must divide r_1, and then, since it divides b and r_1, it must divide r_2, and so on. We can conclude then it must divide the last nonzero remainder. In the other direction, the last remainder must divide all of the previous remainders and a,b. Thus it is the g.c.d., because g.c.d. is the only number which divides both a and b and at the same time is divisible by any other number which divides a and b

We now want to estimate the time of running this algorithm.
Claim. $r_{j+2} < \frac{1}{2} r_j$.
Proof of claim. If $r_{j+1} \leq \frac{1}{2} r_j$, then $r_{j+2} < r_{j+1} \leq \frac{1}{2} r_j$. If $r_j > \frac{1}{2} r_j$, because $r_j = 1 \cdot r_{j+1} + r_{j+2}$, then $r_{j+2} = r_j - r_{j+1} < \frac{1}{2} r_j$.

Since every two steps must result in reducing the remainder at least in half, the whole process will not take more than $2 \cdot [\log_2 a]$ divisions. This takes time $O(\log a)$. Also, each division involves number smaller or equal to a. So each division takes less than $O(\log^2 a)$. In total the time needed for doing this algorithm is less than $O(\log^3 a)$

□

Proposition 1.6. *Let $d = g.c.d.(a, b)$, where $a > b$. Then there exist integers u and v such that $d = ua + bv$. In other words, the g.c.d. of two numbers can be expressed as a linear combination of the numbers with integer coefficients. In addition, finding the integers u and v can be done in $O(\log^3 a)$ bit operations.*

Proof. The procedure is a simple reverse of the Euclidean algorithm. We just need to write the g.c.d in the form of earlier remainders. We run the Euclidean algorithm concluding $d = r_n$. Then $r_i = r_{i-2} + q_i r_{i-1}$. So each remainder can be written as a linear combination of the two preceding remainders. Using repeated substitution, we can then write $d = r_n$ in terms of $b = r_0$ and $a = r_{-1}$. And in each step we need a multiplication and an addition or subtraction. Not hard to see such procedure also takes $O(\log^3 a)$ bit operations. □

Corollary 1.7. *If $a > b$ are relatively prime integers, then 1 can be written as an integer linear combination of a and b in $O(\log^3 a)$ bit operations*

Definition 1.8. Let n be a positive integer. The *Euler phi-function* $\varphi(n)$ is defined to be the number of nonnegative integers b less than n which are relatively prime to n:

$$\varphi(n) = |\{0 \leq b < n | g.c.d.(b, n) = 1\}|.$$

By definition, $\varphi(1) = 1$ and $\varphi(p) = p - 1$ for any prime p. It is also easy to show that for prime powers

$$\varphi(p^\alpha) = p^\alpha - p^{\alpha-1}.$$

Corollary 1.9. *The Euler phi-function is "multiplicative", meaning that $\varphi(mn) = \varphi(m)\varphi(n)$ whenever g.c.d.(m,n)=1.*

Proof. We know that $g.c.d.(x, mn) = 1 \iff g.c.d(x, m) = 1$ and $g.c.d(x, n) = 1$. By the Chinese Remainder Theorem, there is a bijection between $(x\ mod\ mn)$ and $(x\ mod\ m,\ x\ mod\ n)$. So $|\{x\!: g.c.d.(x\ mod\ mn,\ mn)=1\}|=|\{(u,v)\!:g.c.d.(u\ mod\ m,\ m)=1$ and $g.c.d.(v\ mod\ n,\ n)=1\}|$.So $\varphi(nm) = \varphi(m)\varphi(n)$ □

Proposition 1.10. *We have the equality $\sum_{d|n} \varphi(d) = n$.*

Proof. Let $f(n)$ denote the left side of the equality. We need to show that $f(n) = n$. First we show that $f(n)$ is multiplicative, i.e., that $f(mn) = f(m)f(n)$ whenever $g.c.d.(m, n) = 1$. Since n and m have no common factor, any divisor $d|mn$ can be written uniquely in the form $d_1 \cdot d_2$, where $d_1|m, d_2|m$. Since $g.c.d.(d_1, d_2) = 1$, we have $\varphi(d) = \varphi(d_1)\varphi(d_2)$ by Corollary 1.9.. We get all divisors d of mn by taking all possible pairs of d_1 and d_2. Thus, $f(mn) = \sum_{d_1|m} \sum_{d_2|n} \varphi(d_1)\varphi(d_2) = (\sum_{d_1|m} \varphi(d_1))(\sum_{d_2|n} \varphi(d_2)) = f(m)f(n)$. Now assume the prime factorization of n as $n = p_1^{\alpha_1} p_2^{\alpha_2} ... p_r^{\alpha_r}$. By the claim we just proved, it suffices to show $f(p^\alpha) = p^\alpha$. The divisors of p^α are p^j for $0 \leq j \leq \alpha$. Consequently, $f(p^\alpha) = \sum_{j=0}^{\alpha} \varphi(p^j) = 1 + \sum_{j=1}^{\alpha} \varphi(p^j) = p^\alpha$. □

Definition 1.11. Given three integers a, b and m, we say that "a is congruent to b modulo m" and write $a \equiv b\ mod\ m$, if the difference $a - b$ is divisible by m. The number m is called the *modulus* of the congruences. The following properties follow from the definition

(1) (i) $a \equiv a\ mod\ m$; (ii) $a \equiv b\ mod\ m$ if and only if $b \equiv a\ mod\ m$; (iii) if $a \equiv b\ mod\ m$ and $b \equiv c\ mod\ m$. Therefore, for fixed m, (i)-(iii) mean that congruence modulo m is an *equivalence relation.*
(2) For fixed m, each *equivalence class* with respect to congruence modulo m has one and only one representation between 0 and $m - 1$. The set of equivalence classes (called *residue classes*) will be denoted $\mathbf{Z}/m\mathbf{Z}$.
(3) If $a \equiv b\ mod\ m$ and $c \equiv d\ mod\ m$, then $a \pm c \equiv b \pm d\ mod\ m$ and $ac \equiv bd\ mod\ m$. So the set of equivalence classes $\mathbf{Z}/m\mathbf{Z}$ is a commutative ring, and we have a homeomorphism, $\mathbf{Z} \longrightarrow \mathbf{Z}/m\mathbf{Z}$, $a \longmapsto [a]$

(4) If $a \equiv b \ mod \ m$, then $a \equiv b \ mod \ d$ for any $d|m$.
(5) If $a \equiv b \ mod \ m$, $a \equiv b \ mod \ n$, and m and n are relatively prime, then $a \equiv b \ mod \ mn$.

Theorem 1.12. ***(Fermat's Little Theorem).*** *Let p be a prime. Any integer a satisfies $a^p \equiv a \ mod \ p$, and any integer a not divisible by p satisfies $a^{p-1} \equiv 1 \ mod \ p$.*

Proof. If $p|a$ it is obvious that a^p and a are both divisible by p, so they are both congruent to 0. Now suppose p does not divide a. We claim that $0a, 1a, 2a, 3a, ..., (p-1)a$ form a complete set of residues modulo p. If the claim does not hold, then $ia \equiv ja \ mod \ p$, for some $i \neq j$, but this means $p|(i-j)a$, and since a is not divisible by p, we would have $p|(i-j)$. But we know i, j are both less than p, this cannot happen. We conclude that integers $a, 2a, ..., (p-1)a$ are simply a rearrangement of $1, 2, ..., p-1$ modulo p. Thus the products of these two sets of numbers are equal to modulo p, i.e., $a^{p-1}(p-1)! \equiv (p-1)! \ mod \ p$. Thus, $p|((p-1)!(a^{p-1}-1))$. Since $(p-1)!$ is not divisible by p, so $p|(a^{p-1}-1)$, hence $a^{p-1} \equiv 1 \ mod \ p$. If we multiply both sides by a, we have that $a^p \equiv a \ mod \ p$. □

Theorem 1.13. ***Chinese Remainder Theorem.*** *Suppose that we want to solve a system of congruences :*

$$\begin{gathered} x \equiv a_1 \ \ mod \ m_1 \\ x \equiv a_2 \ \ mod \ m_2 \\ \dots \ \dots \\ x \equiv a_3 \ mod \ m_r. \end{gathered}$$

Suppose further g.c.d.(m_i, m_j)=1 for $i \neq j$. Then there exists a simultaneous solution x to all of the congruences, and any two solutions are congruent to one another modulo $M = m_1 m_2 ... m_r$

Proof. First we prove uniqueness modulo M. Suppose that x' and x'' are two solutions. Let $x = x' - x''$. Then x must be congruent to 0 modulo each m_i, and hence modulo M.

Now we construct such an x. Define $M_i = M/m_i$ to be the product of all of the moduli *except for* the i-th. Clearly $g.c.d.(m_i, M_i) = 1$, so by the Euclidean algorithm we can find an N_i such that $M_i N_i \equiv 1 \ mod \ m_i$. Now set $x = \sum_i a_i M_i N_i$. Then for each i, all the terms in the sum other than the i-th term are divisible by m_i. Thus, for each i: $x \equiv a_i M_i N_i \equiv a_i, \ mod \ m_i$. □

Proposition 1.14. *If g.c.d.(a,m)=1, then $a^{\varphi(m)} \equiv 1 \ mod \ m$*

Proof. We first prove the proposition when m is a prime power, i.e. when $m = p^\alpha$. We use induction here. The case $\alpha = 1$ is precisely Fermat's Little Theorem. Suppose that $\alpha \geq 2$, and the formula holds for the $(\alpha-1)$-st power of p. Then $a^{p^{\alpha-1}-p^{\alpha-2}} = 1 + p^{\alpha-1}b$ for some integer b, by the assumption of induction. Raise both sides to the p-th power. So the right side of the equation is a binomial expansion. Besides the first time 1^p, all other times are divisible by p^α, including the last one $(p^{\alpha-1}b)^p$. Then $a^{p^\alpha - p^{\alpha-1}}$ is equal to 1 plus a sum with each term divisible by p^α, which is equivalent to $a^{\varphi(m)} \equiv 1 \ mod \ m$

Finally by the multiplicity of the Euler's phi-function, raising both sides of $a^{\varphi(p^\alpha)} \equiv 1 \ mod \ p^\alpha$ to appropriate power we can get $a^{\varphi(m)} \equiv 1 \ mod \ p^\alpha$. For every prime factor p this holds, and so by Property 5 following Definition 1.10., we conclude $a^{\varphi(m)} \equiv 1 \ mod \ m$ □

2. Finite Fields and Quadratic Residues

Finite fields. We shall here briefly summarize some characteristics of finite fields assuming the reader has some knowledge of the concept of a field. Let $\mathbf{F}_q$ denote a field which has a finite number q of elements in it. Such a field cannot have characteristic zero; let p be the characteristic of $\mathbf{F}_q$. Then p is prime, $\mathbf{F}_q$ contains the field $\mathbf{F}_p = \mathbf{Z}/p\mathbf{Z}$, and so is a finite dimensional vector space over $\mathbf{F}_p$. Thus $q = p^n$, where $n = \dim_{\mathbf{F}_p}\mathbf{F}_q$.

Proposition 2.1. *The order, which is the least positive power of an element that equals 1, of any $a \in \mathbf{F}_q^*$ divides $q-1$.*

Proof. In $\mathbf{F}_q^*$, we list all $q-1$ elements and multiply each of them by a. We get a permutation of the same elements, because any two distinct elements remain distinct after multiplication, since $ax = ay$ implies $a^{-1}ax = a^{-1}ay$. We have $x = y$. Thus the products of these two sets of numbers are equal. So $a^{q-1} = 1$. Let d be the order of a. If d does not divide $q-1$, then $q-1 = bd + r$ such that $a^r = a^{q-1-bd} = 1$. This contradicts the minimality of d. □

Definition 2.2. A *generator* g of a finite field $\mathbf{F}_q$ is an element of order $q-1$; equivalently, the powers of g run through all of the elements of $\mathbf{F}_q^*$.

Proposition 2.3. *Every finite field has a generator. If g is a generator of $\mathbf{F}_q^*$, then g^j is also a generator if and only if g.c.d.(j,q-1)=1. In particular, there are a total of $\varphi(q-1)$ different generators of $\mathbf{F}_q^*$.*

Proof. Suppose that $a \in \mathbf{F}_q^*$ has order d, i.e., a^d=1 and no lower power of a gives 1. By the previous proposition we know d divides $q-1$. Since $a^d = 1$ is the smallest power which equals 1, it follows then $a, a^2, ..., a^d = 1$ are distinct.

We claim that the elements of order d are precisely the $\varphi(d)$ values a^j for which $g.c.d. = (j,d) = 1$. Firstly, since the d distinct powers of a all satisfy the equation $x^d = 1$, these are the roots of the equation. Thus any element of order d must be among the powers of a. However, we shall note that not all powers of a have order d, since if $g.c.d.(j,d) = d' > 1$, then a^j has lower order. Conversely, we now show that a^j does have order d if $g.c.d.(j,d) = 1$. If a^j had a smaller order d'', then $a^{d''}$ raised to j-th or d-th power would give 1, and hence $a^{d''}$ raised to the $g.c.d(j,d) = 1$ would also give 1. But this contradicts the minimality of the order d for a. This proves the claim.

This tells us that if there is any element of order d, then there exist $\varphi(d)$ elements of order d. Now every element has some order $d|(q-1)$. There are either 0 or $\varphi(d)$ elements of order d. By Proposition 1.10., $\sum_{d|(q-1)} \varphi(d) = q-1$, which is the number of elements in $\mathbf{F}_q^*$. Thus every element has some order $d|(q-1)$ implies there are always $\varphi(d)$ elements of order $q-1$; In particular, there are $\varphi(q-1)$ elements of order $q-1$, then the other elements of order $q-1$ are precisely the powers g^j where $g.c.d.(j, q-1) = 1$. □

Quadratic residues. Suppose that p is an odd prime. We are interested in knowing which of the nonzero elements $\{1, 2, ..., p-1\}$ of $\mathbf{F}_p$ are squares. If some $a \in \mathbf{F}_p^*$ is a square, say $b^2 = a$, then $\pm b$ are two square roots. Thus, the squares in

$\mathbf{F}_p^*$ can be found by computing b^2 *mod* p for $b = 1, 2, 3, ..., (p-1)/2$. Therefore, we have $(p-1)/2$ residues and $(p-1)/2$ nonresidues.

The Legendre symbol. Let a be an integer and $p > 2$ a prime. We define the *Legendre symbol* $(\frac{a}{p})$ as follows:

$$\left(\frac{a}{p}\right) = \begin{cases} 0, & \text{if } p|a; \\ 1, & \text{if a is a quadratic residue modulo } p; \\ -1, & \text{if a is a nonresidue modulo } p. \end{cases}$$

Proposition 2.4. $(\frac{a}{p}) \equiv a^{(p-1)/2}$ *mod p.*

Proof. If a is divisible by p, then both sides are congruent to 0 *mod* p. Suppose $p \not| a$. By Fermat's Little Theorem, in $\mathbf{F}_p$ the square of $a^{(p-1)/2}$ is 1, so $a^{(p-1)/2}$ is ± 1. Let g be a generator of $\mathbf{F}_p^*$, and let $a = g^j$. As we saw, a is a residue if and only if j is even, and $a^{(p-1)/2} = g^{j(p-1)/2} = 1$ if and only if $j(p-1)/2$ is divisible by $p-1$, i.e., if and only if j is even. Thus, both sides of the congruence in the proposition are ± 1 in $\mathbf{F}_p$, and each side is $+1$ if and only if j is even. □

Proposition 2.5. *The legendre symbol satisfies the following properties:*
(a) $(\frac{a}{p})$ *depends only on the residue of a modulo p;*
(b) $(\frac{ab}{p}) = (\frac{a}{p})(\frac{b}{p})$*;*
(c) for b relatively prime to p, $(\frac{ab^2}{p}) = (\frac{a}{p})$*;*
(d) $(\frac{1}{p}) = 1$ *and* $(\frac{-1}{p}) = (-1)^{(p-1)/2}$*.*

Proof. Part (a) follows directly from definition. Part(b) follows from Proposition 2.4., because the right side is congruent modulo p to $a^{(p-1)/2} . b^{(p-1)/2)} = (ab)^{(p-1)/2}$, as is the left side. Part (c) follows from Part (b). For Part (d), we note $1^2 = 1$, and follows from Proposition 2.4. □

The Jacobi symbol. Let a be an integer, and let n be any positive odd numbers. Let $n = p_1^{\alpha_1} \cdots p_r^{\alpha_r}$ be the prime factorization of n. Then we define the *Jacobi symbol* $(\frac{a}{n})$ as the product of the Legendre symbols for the prime factors of n,

$$(\tfrac{a}{n}) = (\tfrac{a}{p_1})^{\alpha-1} \cdots (\tfrac{a}{p_r})^{\alpha_r}.$$

3. Primality and Factoring

In cryptography, it is often necessary to find big prime numbers and the factors of large integers. Given an integer n, a simple method to find its factorization or test its primality is to divide n by all integers smaller than n. However, if n is very large, this method has a very long run time. Here we introduce some quicker ways of testing primality and factoring.

According to Fermat's Little Theorem, we know that, if n is prime, then for any b such that $g.c.d.(b, n) = 1$ one has

$$b^{n-1} \equiv 1 \ mod \ n \tag{3.1}$$

Definition 3.2. If n is an odd composite number and b is an integer such that $g.c.d.(n, b) = 1$ and (3.1) holds, then n is a weak *pseudoprime* to the *base b*.

Proposition 3.3. *Let n be an odd composite integer.*
(a) The number n is a pseudoprime to the base b, where g.c.d.(b,n)=1, if and only if the order of b in $(\boldsymbol{Z}/n\boldsymbol{Z})^*$ divides n-1.
(b) If n is a pseudoprime to the bases b_1 and b_2, then n is a pseudoprime to the base b_1b_2 and also to the base $b_1b_2^{-1}$
(c) If n fails the test (3.1) *for a single base $b \in (\boldsymbol{Z}/n\boldsymbol{Z})^*$, then n fails* (3.1) *for at least half of the possible bases $b \in (\boldsymbol{Z}/n\boldsymbol{Z})^*$.*

Proof. Parts (a) and (b) are easy to verify. To prove (c), let $\{b_1, b_2, ..., b_s\}$ be the set of all bases for which n is a pseudoprime, i.e., the set of all integers $0 < b_i < n$ for which the congruence (3.1) holds. Let b be a fixed base for which n is not a pseudoprime. If n were a pseudoprime for any of the bases bb_i, then, by Part (b), it would be a pseudoprime for the base $b \equiv (bb_i)b_i^{-1}$ *mod* n , which contradicts the assumption. Thus if we have a set of good bases $\{b_1, b_2, ..., b_s\}$, then there would be a bad base $\{bb_1, bb_2, ..., bb_s\}$ to which n fails to be a pseudoprime. So there are at least as many bases for which n fails for (3.1) as for which (3.1) holds. $\square$

By Part (c) of the previous proposition, if n is a composite number, and if the test fails for some base b, the probability that n is a pseudoprime with respect to k different bases is at most $\frac{1}{2^k}$. This suggests that the number is prime *unless* it is a composite number with the very special property that (3.1) holds for every b less than n. The question now is if such special numbers exist. The answer turns out to be yes.

Definition 3.4. A *Carmichael number* is a composite integer n such that (3.1) holds for every $b \in (\mathbf{Z}/n\mathbf{Z})^*$.

Proposition 3.5. *Let n be an odd composite integer.*
(a) If n is divisible by a perfect square > 1, then n is not a Carmichael number.
(b) If n is square free, then n is a Carmichael number if and only if $p - 1|n - 1$ for every prime p dividing n.

Proof. (a) Suppose that $p^2|n$, let g be a generator modulo p^2. Let n' be the product of all primes other than p which divide n. By the Chinese Remainder Theorem, there is an integer b satisfying the two congruences: $b \equiv g$ *mod* p^2 and $b \equiv 1$ *mod* n'. Then b is a generator modulo p^2, relatively prime to n. If n is a pseudoprime to base b, then we automatically have $b^{n-1} \equiv 1$ *mod* p^2. Since $\varphi(p^2)$ we must have $p(p - 1)|n - 1$. However, $n - 1 \equiv -1$ *mod* p, since $p|n$, and hence $n - 1$ is not divisible by $p(p - 1)$, which is a contradiction.

(b) First suppose that $p - 1|n - 1$ for every p dividing n. Let b be any base, where $g.c.d.(b, n) = 1$. Then for every prime p dividing n we have that: b^{n-1} is a power of b^{p-1}, and so is congruent to 1 *mod* p. Thus, $b^{n-1} - 1$ is divisible by all of the prime factors p of n, and their product n. Thus (3.1) holds for all bases b. Conversely, suppose that there is a p such that $p-1$ does not divide $n-1$. Then for a generator g in $(\mathbf{Z}/p\mathbf{Z})^*$, find an integer b satisfying $b \equiv g$ *mod* p and $b \equiv 1$ *mod* n/p. Then $g.c.d.(b, n) = 1$, and $b^{n-1} \equiv g^{n-1}$ *mod* p. But g^{n-1} is not congruent to 1 *mod* p, because $n - 1$ is not divisible by the order $p - 1$ of g. Thus $b^{n-1} \not\equiv 1$ *mod* p $\square$

Proposition 3.6. *A Carmichael number must be the product of at least three distinct primes.*

Proof. By the previous Proposition, a Carmichael number must be a product of distinct primes. So we need to show that the product of two distinct prime numbers cannot be Carmichael number. Let $n = pq$, where $p < q$ are both prime numbers. Then if n were a Carmichael number, we would have $n - 1 \equiv 0$ *mod q-1*. But $n - 1 = p(q - 1 + 1) - 1 \equiv p - 1$ *mod q-1*, and this is not divisible by $q - 1$, since $0 < p - 1 < q - 1$. $\square$

In 1956, Erdos proved that $C(n) < ne^{\frac{-k \log n \log\log\log n}{\log\log n}}$, where $C(n)$ equals to the number of Carmichael numbers less than a given integer n. In the other direction, Harman in 2005 proved $C(n) > n^{0.332}$. Because of this, using (3.1) as a primality test has significant problems. However we can strengthen (3.1) into a better test which does not have an analogue of Carmichael numbers.

Euler pseudoprimes. Let n be an odd integer, and let $(\frac{b}{n})$ denote the Jacobi symbol. According to Proposition 2.4., if n is a prime number, then

$$b^{(n-1)/2} \equiv (\frac{b}{n}) \ mod\ n \tag{3.7}$$

Definition 3.8. If n is an odd composite number and b is an integer such that $g.c.d.(n, b) = 1$ and (3.7) holds, then n is called an *Euler pseudoprime* to the base b.

Proposition 3.9. *If n is an Euler pseudoprime to the base b, then it is a pseudoprime to the base b.*

Proof. If we square both sides of (3.7), then (3.1) holds. Thus if (3.7) holds, we automatically have (3.7). $\square$

Thus we know Euler pseudoprimes are all pseudoprimes. We get a better chance of eliminating composite numbers by letting the numbers run through this test. They are both probabilistic methods, because one is not sure one has a prime. However, there are no Euler-Jacobi pseudoprimes to all bases.

Fermat factorization. There is a way to factor a composite number n that is efficient if n is a product of two integers which are close to one another. This method, called *Fermat factorization*, is based on the fact that n is then equal to a difference of two squares, one of which is very small.

Proposition 3.10. *Let n be a positive odd integer. There is a 1-to-1 correspondence between factorizations of n in the form $n = ab$, where $a \geq b > 0$, and representations of n in the form $t^2 - s^2$, where s and t are nonnegative integers. The correspondence is given by the equations*

$$t = \frac{a+b}{2}, \quad s = \frac{a-b}{2}; \qquad a = t + s, \quad b = t - s.$$

Proof. $n = ab = ((a + b)/2)^2 - ((a - b)/2)^2$, so we obtain the representation as a difference of two squares. Conversely, given $n = t^2 - s^2$ we can factor it as $(t + s)(t - s)$. $\square$

If $n = ab$ with a and b close together, then $s = (a - b)/2$ is small, and so t is only slightly larger than $\sqrt{n}$. Then we can find a and b by trying all values for t starting with $\lfloor\sqrt{n}\rfloor + 1$, until we find one such that $t^2 - n = s^2$ is a perfect square.

Example 3.11. Factor 200819.
Solution. We have $\lfloor\sqrt{200819}\rfloor + 1 = 449$. Now $449^2 - 200819 = 782$, which is not a perfect square. Next, we try $t = 450$, and $450^2 - 200819 = 1681 = 41^2$. Thus, $200819 = 450^2 - 41^2 = (450 + 41)(450 - 41) = 491 \cdot 409$.

Factor bases. There is a generalization of the idea behind Fermat factorization which leads to a much more efficient factoring method. Namely, we use the fact that any time we are able to obtain a congruence of the form $t^2 \equiv s^2$ *mod* n with $t \not\equiv \pm s$ *mod* n, we immediately find a factor of n by computing $g.c.d.(t + s, n)$. This is because we have $n|t^2 - s^2 = (t + s)(t - s)$, while n does not divide $t + s$ or $t - s$; thus $g.c.d.(t + s, n)$ must be a proper factor a of n, and then $b = n/a$ divides $g.c.d.(t - s, n)$.

Definition 3.12. A *factor base* is a set $B = \{p_1, p_2, ..., p_h\}$ of distinct primes, except that p_1 may be the integer -1. We say that the square of an integer b is a $B - number$ (for a given n) if residue b^2 *mod* n can be written as a product of numbers from B.

Example 3.13. For $n = 4633$ and $B = \{-1, 2, 3\}$, the squares of the three integers 67,68 and 69 are $B - numbers$, because $67^2 \equiv -144$ *mod* 4633, $68^2 \equiv -9$ *mod* 4633, and $69^2 \equiv 128$ *mod* 4633.

Let $\mathbf{F}_2^h$ denote the vector space over the field of two elements which consists of h-tuples of zeros and ones. Given n, a factor base B containing h numbers, and a $B - number$ b, we have a corresponding vector $\overrightarrow{\epsilon_b} = ([\alpha_1], ..., [\alpha_n])$ where $b^2 = \Pi p_j^{\alpha_j}$ and $[\alpha_i] = \alpha_i$ *mod 2*.

Example 3.14. In the situation of Example 3.11., the vector corresponding to 67 is {1,0,0}, the vector corresponding to 68 is {1,0,0}, and the vector corresponding to 69 is {0,1,0}.

Suppose that we have some set of "B-numbers" $\{b_i^2\}$ *mod* n such that the corresponding vectors $\overrightarrow{\epsilon}_i = \{\epsilon_{i1}, ..., \epsilon_{ih}\}$ add up to the zero vector in $\mathbf{F}_2^h$. Then the product of the least absolute residues of b_i^2 is equal to a product of *even* powers of all of the p_j in B. That is, if for each i we let a_i denote the least absolute residue of b_i^2 *mod* n and we write $a_j = \prod_{j=1}^h p_j^{\alpha_{ij}}$, we obtain

$$\prod a_i = \prod_{j=1}^{h} p_j^{\sum_i \alpha_{ij}},$$

with the exponent of each p_j an even number on the right. Then the right hand side is the square of $\prod_j p_j^{\gamma_j}$ with $\gamma_j = \frac{1}{2}\sum_i \alpha_{ij}$. Thus, if we set $b = \prod_i b_i$ *mod* n and $c = \prod_j p_j^{\gamma_j}$ *mod* n, we obtain two numbers b and c, constructed in different ways, whose squares are congruent modulo n.

In case $b \equiv \pm c$ *mod* n, then we must start again with another collection of B-numbers whose corresponding vectors sum to zero. In practice, one method is to start with B consisting of the first h primes and choose random b_i's until we find several whose squares are B-numbers. Another method is to start by choosing some b_i's for which b_i^2 *mod* n is small in absolute value. Then choose B to consist of a small set of small primes with $p_1 = -1$, so that several of the b_i^2 *mod* n can be expressed in terms of the numbers in B.

When can we be sure that we have enough b_i to find a sum of $\overrightarrow{\epsilon}_i$ which is the zero vector? This requires the collection of vectors to be linearly dependent over the field $\mathbf{F}_2$, which is guaranteed to occur if we have $h+1$ vectors.

Factor base algorithm. This is a systematic method to factor a very large n using a *random* choice of the b_i. Choose an integer y of intermediate size, e.g., if n is a 50-decimal-digit integer, we might choose y to be a number with 5 or 6 decimal digits. Let B consist of -1 and all primes $\leq y$. Choose a large number of random b_i, and try to express b_i^2 *mod* n as a product of the primes in B. Once we get a large quantity of B-numbers b_i^2 *mod* n ($\pi(y)+2$ is good enough, where $\pi(y)$ denotes the number of primes $\leq y$), take the corresponding vectors in $\mathbf{F}_2^h$ and by row-reduction determine a subset of the b_i whose corresponding $\overrightarrow{\epsilon}_i$ sum to zero. Then we compute $b = \prod b_i \ mod\ n$ and $c = \prod p_j^{\gamma_j} \ mod\ n$. If we unluckily get $b \equiv \pm c$ *mod* n, then we start again. Otherwise we compute $g.c.d.(b+c, n)$ or $g.c.d.(b-c, n)$, which will be nontrivial factor of n.

Time estimate of factor base algorithm. We give a very rough time estimate here just to demonstrate the difficulty of factorization. Before we start, we need to know two facts:

Fact 1 The $\log(n!)$ function is approximately $n \log n - n$.

By "approximately", we are talking about the limit as n approaches infinity. This can be proved by observing that $\log(n!)$ is the upper Riemann sum for the integral $\int_1^n \log x dx = n \log n - n + 1$.

Fact 2 Given a positive integer N and a positive number u, the total number of nonnegative integer N-tuples (α_j) such that $\sum_{j=1}^{N} \alpha_j \leq u$ is the binomial coefficient $\binom{\lfloor u \rfloor + N}{N}$.

This can be proved by letting each N-tuple solution α_j correspond to the following choice of N integers β_j from among 1,2,...,$\lfloor u \rfloor + N$. Let $\beta_1 = \alpha_1 + 1$, and for $j \geq 1$ let $\beta_{j+1} = \beta_j + \alpha_{j+1} + 1$, i.e., we choose the β_j's so that there are α_j numbers strictly between β_{j-1} and β_j. This gives a 1-to-1 correspondence between the number of solutions and the number of ways of choosing N numbers from a set of $\lfloor u \rfloor + N$ numbers.

In order to estimate the time of our algorithm, we need to know the probability that a random number less than x will be a product of primes less than y. To do this, we first let u denote the ratio $\frac{\log x}{\log y}$ That is , if x is an r-bit integer and y is an s-bit integer, then u is approximately the ratio of digits r/s. We shall here ignore smaller terms by assuming that u is *much* smaller than y. We let $\pi(y)$, denote the number of primes less than y. Typically, in practice we have

$$y \approx 10^6 \quad (\pi(y) \approx 7 \cdot 10^4, \quad \log y \approx 14); \quad u \approx 8; \quad x \approx 10^{48}.$$

We let $\Psi(x, y)$ denote the number of integers less than x which are not divisible by any prime greater than y. So there is a 1-to-1 correspondence between $\pi(y)$-tuples of nonnegative integers α_j for which $\prod_j p_j^{\alpha_j} \leq x$, where α_j are nonnegative integers and p_j are primes less than y and integers less than x which are a product of p_i. Thus, $\Psi(x, y)$ is equal to the number of integer solutions α_j to the inequality $\sum_{j=1}^{\pi(y)} \alpha_j \log p_j \leq \log x$. Because most of the primes less than y have almost the same number of digits as y, we replace $\log p_j$ by $\log y$ in the previous inequality. Then if we divide both sides of the inequality by $\log y$ and replace $\log x / \log y$ by

u, we can say $\Psi(x, y)$ is approximately equal to the number of solutions to the inequality $\sum_{j=1}^{\pi(y)} \alpha_j \leq u$.

We now make another important simplification, expanding the number of variables from $\pi(y)$ to y. We have $\Psi(x, y)$ is roughly equal to the number of solutions to the inequality $\sum_{j=1}^{y} \alpha_j \leq u$, which by Fact 2, is approximately $\binom{\lfloor u\rfloor+y}{y}$. We now estimate $\log\Big(\frac{\Psi(x,y)}{x}\Big)$. Notice that $\log x = u \log y$, by the definition of u. We use the approximation for $\Psi(x, y)$ and Fact 1:

$$\log\Big(\frac{\Psi(x, y)}{x}\Big) \approx \log\Big(\frac{(\lfloor u\rfloor + y)}{\lfloor u\rfloor ! y!}\Big) - u \log y \approx (\lfloor u\rfloor + y) \log(\lfloor u\rfloor + y)$$

$$-(\lfloor u\rfloor + y) - (\lfloor u\rfloor + \log\lfloor u\rfloor - \lfloor u\rfloor) - (y \log y - y) - u \log y$$

We now make another approximation, by replacing $\lfloor u\rfloor$ by u. Next because u is assumed to be much smaller than y, we can replace $\log(u + y)$ by $\log y$. We obtain

$$log\Big(\frac{\Psi(x, y)}{x}\Big) \approx -u \log u,$$

i.e.,

$$\frac{\Psi(x, y)}{x} \approx u^{-u}$$

We are now ready to estimate the number of bit operations required to carry out the factor base algorithm described above, where, for simplicity, we assume that our factor base B consists of the first $h = \pi(y)$ primes. We just estimate the number of bit operations required to carry out the following steps: (1) choose random numbers b_i^2 module n as a product of primes less than y if it can be expressed, continuing this process until we find $\pi(y) + 1$ different b_i's for which b_i^2 *mod* n is written as such a product; (2) find linear dependence between the vectors associated to the b_i to obtain a congruence of the form $b^2 \equiv c^2$ *mod* n; (3) if $b \equiv \pm c$ mod n, repeat (1) and (2) with new b_i until you obtain $b^2 \equiv c^2$ *mod* n with $b \not\equiv \pm c$ *mod* n, which yields a nontrivial factor of n by computing $g.c.d.(b + c, n)$.

Suppose that n is an r-bit integer and y is an s-bit integer; then u is very close to r/s. We claim that the number of operations needed for testing each b_i is a polynomial in r and y, i.e., it is $O(r^l e^{ks})$ for some integers k and l. It takes a fixed amount of time to generate a random bit, and so $O(r)$ bit operations to generate a random integer b_i between 1 and n. Then computing b_i^2 *mod* n takes $O(r^2)$ bit operations. We must then divide b_i^2 *mod* n by all primes $\leq y$ which divide it evenly. This can be done by dividing it successively through 2, 3, 5, and so on. Since a division of an integer of less than r bits by an integer of $\leq s$ bits takes time $O(rs)$, we see that each test of a randomly chosen b_i takes $O(rsy)$ bit operations.

To complete step (1) requires testing approximately $u^u(\pi(y) + 1)$ values of b_i, in order to find $\pi(y) + 1$ values for which b_i^2 *mod* n is a product of primes $\leq y$. Since $\pi(y) \approx \frac{y}{\log y} = O(y/s)$, this means that step (1) takes an expected $O(u^u r y^2)$ bit operations. Step (2) involves operations which are polynomial in y and r (matrix reduction and finding b and c modulo n). Thus it takes $O(y^j r^h)$ bit operations for some integers j and h. Thus combining these two steps we have

$$O(u^u r y^2 + y^j r^h) = O(r^h u^u y^j) = O(r^h (r/s)^{r/s} e^{ks}),$$

for suitable integers h and k. Since we can pick y for this equation, we choose a value of s to minimize the runtime for the algorithm. We calculate

$$0 = \frac{d}{ds}(\frac{r}{s}\log\frac{r}{s} + ks) = -\frac{r}{s^2}(\log\frac{r}{s} + 1) + k \approx -\frac{r}{s^2}\log r + k,$$

The last approximation step is due to the fact that log $\frac{r}{s}$ is relatively large compared to 1. And we find approximately

$$0 \approx -\frac{r}{s^2}\log r + k,$$

Plugging the expression for s to the equation $r^h(\frac{r}{s})^{\frac{r}{s}}$ gives the time estimate of

$$O(e^{C\sqrt{r\log r}}).$$

4. Simple cryptosystems

Finally, we get to the main topic of the paper, cryptography. We begin with some basic notions. The message we want to send is called the *plaintext* and the disguised message is called the *ciphertext*. The plaintext and ciphertext are written in some *alphabet* with N letters. The term "letter" can refer not only to the familiar A-Z, but also to numerals, blanks, punctuation marks, or any other symbols that we allow ourselves to use when writing the messages. The process of converting a plaintext to a cipertext is called *enciphering* or *encryption*, and the reverse process is called *deciphering* or *decryption*. The plaintext and ciphertext are broken up into message units. A message unit might, for example, be a single letter, a pair of letters (*diagraph*), a triple of letters, or a block of 50 letters. Usually we use f to denote the enciphering transformation and f^{-1} as the deciphering transformation. Notice here that a enciphering transformation must be 1-to-1 and onto, which is equivalent to having an inverse, otherwise we could not always reveal the plaintext.

Simple shifting map. Let us start with single letter units message. Suppose we have an N-letter alphabet labeled by the integers $0, 1, 2, ..., N-1$. Then such an enciphering transformation is a permutation of these N numbers.

Example 4.1. Suppose we are using the 26-letter alphabet A-Z with numerical equivalents 0-25. Let the letter $P \in \{0, 1, ..., 25\}$ stand for a plaintext message unit. Define a function f from the set $\{0, 1, ..., 25\}$ to itself by the rule

$$f(P) = \begin{cases} P+3, & \text{if } x < 23; \\ P-23, & \text{if } x \geq 23; \end{cases}$$

With this system, to encipher the word "YES" we first convert to numbers, 24, 4 and 18, then add 3 modulo 26, we get 1, 7 and 21, then translate back to letters: "BHV". Deciphering a cipher text "ZKB", for example, yields the plaintext "WHY".

Affine map: $C \equiv aP + b \ mod \ N$, where a and b are fixed integers and $g.c.d.(a, N) = 1$. Then the deciphering transformation will be $P \equiv a'C + b' \ mod \ N$, where

$$a' \equiv a^{-1} \in (\mathbf{Z}/N\mathbf{Z})^*, b' = -a^{-1}b.$$

Suppose we have a long stream of data, to break such a cryptosystem, we use frequency analysis, a technique described in the example below.

Example 4.2. Suppose a ciphertext that we know uses the 28-letter alphabet consisting of A-Z, a blank, and ?, where $A - Z$ have numerical equivalents 0-25, blank=26, ?=27. We also know that the enciphering function is affine. Then examining the ciphertext we find that the most commonly used letters of ciphertext are "B" and "?" and the most commonly used letters in English are "blank" and "E". Then we assume the enciphering function takes "blank" to "B" and "E" to "?". This leads to the two congruences: $a' + b' \equiv 26 \; mod \; 28$, $27a' + b' \equiv 4 \; mod \; 28$. Thus $a' \equiv 11 \; mod \; 14$ and this gives $a' \equiv 11 \; mod \; 28$, $b' \equiv 15 \; mod \; 28$, or $a' \equiv 25 \; mod \; 28$, $b' \equiv 1 \; mod \; 28$. Then we check these two answers to see which one makes more sense in English.

Diagraph transformations. In the N-letter alphabet, we assign the two-letter block XY a numerical equivalent $P = XN + Y$, where $X, Y \in \{0, 1, ..., N-1\}$ and then encipher the plaintext into ciphertext.

Example 4.3. Suppose we are working in the 26-letter alphabet and using the digraph enciphering transformation $C \equiv 159P + 580 \; mod \; 676$. Then the digraph "NO" has numerical equivalent $13 \cdot 26 + 14 = 352$ and is taken to the ciphertext digraph $159 \cdot 352 + 580 = 440 \; mod \; 676$, which is "QY". The digraph "ON" by computation is taken to "NV". Notice that once we change the unit, there is no relation between the encryption of one digraph and that of another one that has a letter in common with it.

To break such a system, frequency analysis is also needed. Statistically speaking, "TH" and "HE" are the two most frequently occurring diagraphs, in that order.

5. Public Key

The *parameters* we used in a cryptosystem, like the a, b we used for the affine map, are called keys of the cryptosystem. The values of the parameters in the enciphering map are called the *enciphering key* K_E, and the set of parameters of the deciphering map are called *deciphering key.* In the simple cryptosystem we introduced above, having knowledge of the enciphering key is equal to having the knowledge of deciphering key. Suppose we have more than two people using the same cryptosystem, and each pair of them wants to keep their communication secret. In this case, a user A should only know the recipient B's enciphering key but not the deciphering key, otherwise he would know all the messages intended for B. The idea of public key cryptosystem has the property that someone who knows only the enciphering key cannot find the deciphering key without a prohibitively lengthy computation. In essence, such a key shall be a function that cannot be easily inverted. In this last section, we introduce three different public key cryptosystems.

RSA. Named after its co-inventors, the RSA public key system is based on the computational difficulty of factoring a big integer. Each user A chooses two primes p_A and q_A and a random number e_A which has no common factor with $(p_A - 1)(q_A - 1)$. Next, A computes $n_A = p_A q_A$, $\Phi(n_A) = (p_A - 1)(q_A - 1) = n_A + 1 - p_A - q_A$, and also the multiplicative inverse of $d_A = e_A^{-1} \; mod \; \Phi(n_A)$. Then he makes public the enciphering key $K_{E,A} = (n_A, e_A)$ and conceals the deciphering key $K_{D,A} = (n_A, d_A)$. The enciphering transformation is the map from $\mathbf{Z}/n_A\mathbf{Z}$ to itself given by $f(P) \equiv P^{e_A} \; mod \; n_A$

Now we need to show that given n and d, we can find P.

Claim: Let d and e be positive integers such that $de-1$ is divisible by $p-1$ for every prime divisor p of n. Then $a^{de} \equiv a \ mod \ n$ for any integer a.

Proof. It suffices to prove that $a^{de} \equiv a \ mod \ p$ for any integer a and each prime divisor p of n. This is obvious if $p|a$. If not then this is just by Fermat's little theorem. $\square$

Thus the enciphering transformation has an inverse which is the one we just constructed, namely $f^{-1}(C) \equiv c^d \ mod \ n$. It is believed that for generic choice of p, q, breaking the system is equivalent to factoring n. As we saw in the last section, for large p, q, this is currently a difficult problem.

Example 5.1. We choose $N = 26$, $k = 3$, $l = 4$,. That is, the plaintext consists of trigraphs and the ciphertext consists of four-graphs in the usual 26-letter alphabet. To send the message "YES" to a user with enciphering key $(n_A, e_A) = (46927, 39423)$, we first get the numerical equivalent of "YES," namely: $24 \cdot 26^2 + 4 \cdot 26_18 = 16346$, and then compute 16346^{39423} *mod* 46927, which is $21166 = 1 \cdot 26^3 + 5 \cdot 26^2 + 8 \cdot 26 + 2 =$ "$BFIC$".

Discrete log When working with the real numbers, exponentiation is not significantly easier than finding the logarithm, but in a finite group, this is not the case. The problem of inverting exponentiation in a finite group is called the "discrete" logarithm problem.

Definition 5.2. If G is a finite group, b is an element of G, and y is an element of G which is a power of b, then the *discrete logarithm* of y to the base b is any integer x such that $b^x = y$.

No efficient classical algorithm for computing general discrete logarithms is known. There exists an efficient quantum algorithm due to *Peter Shor*. [1] Thus we assume that finding discrete logarithms is not computationally feasible.

Diffie-Hellman assumption. It is computationally infeasible to compute g^{ab} knowing only g^a and g^b. Under this assumption, we have a new key exchange system, the Diffie-Hellman key exchange system.

Suppose that users A and B want to agree upon a key, which is a random element of $\mathbf{F}_q^*$. They first pick a generator g of the group. User A chooses a random number a, then computes g^a and makes it public. User B does the same thing by picking b and publishing g^b. Then both of them can compute g^{ab}, but under the Diffie-Hellman assumption, a third party knows only g^a and g^b cannot compute g^{ab}.

The Massy-Omura cryptosystem for message transmission. We suppose that everyone has agreed upon a finite field $\mathbf{F}_q$, which is fixed and publicly known. Each user of the system secretly selects a random integer e between 0 and $q-1$ such that $g.c.d.(e, q-1) = 1$ and, using the Euclidean algorithm, computes its inverse $d = e^{-1}$ *mod q-1*. If user Alice wants to send a message P to Bob, first she sends the element P^{e_A}. This means nothing to anyone but Alice, who is the only one who knows d_A. But then Bob can send $P^{e_A e_B}$ back to Alice. Alice raises the number to d_A-th power and sends back to Bob. Bob finally raise it to d_B-th power.

[1]Shor, Peter (1997). "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". *SIAM J.Sci.Statist.Comput.* 26: 1484.

The idea behind this system is simple. However if some intruder Cathy wants to see the message sent by Alice to Bob, she can pretend to be Bob and sends $P^{e_A e_C}$ back to Alice, where e_C is her own enciphering key. So under this circumstance, we may need some special signature scheme.

The ElGamal cryptosystem. We start by fixing a very large finite field $\mathbf{F}_q$ and a generator $g \in \mathbf{F}_q^*$. We suppose that we are using plaintext message units with numerical equivalents P in $\mathbf{F}_q$. Each user A randomly chooses an integer $a = a_A$, say in the range $0 < a < q-1$. This integer a is the secret deciphering key. The public enciphering key is the element g^a.

To send a message P to the user A, we choose an integer k at random, and then send A the following pair of elements

$$(g^k, Pg^{ak}).$$

Notice that we can compute g^{ak} without knowing a, simply by raising g^a to the k-th power. Now A, who knows a, can recover P from this pair by raising the first element g^k to the a-th power and dividing the result into the second element.

Algorithms for finding discrete logs in finite fields. Though in general, finding the discrete logarithm in a finite field is hard, under certain specific conditions, we have fast algorithms. Suppose all the prime factors of $q-1$ are small. In this case we call $q-1$ "smooth". We find the discrete log of an element $y \in \mathbf{F}_q^*$ to the base b. For simplicity, we suppose b is a generator of $\mathbf{F}_q^*$. This algorithm is due to Silver, Pohlig and Hellman.

First, for each prime p dividing $q-1$, we compute the p-th roots of unity $r_{p,j} = b^{j(q-1)/p}$ for $j = 0, 1, ..., p-1$. Our goal is to find x, $0 \leq x < q-1$, such that $b^x = y$. If $q-1 = \prod_p p^\alpha$ is the prime factorization of $q-1$, then it suffices to find $x \ mod \ p^\alpha$ for each p dividing $q-1$ by Chinese Remainder Theorem. Now we fix a prime p dividing $q-1$, and show how to determine $x \ mod \ p^\alpha$

Suppose $x \equiv x_0 + x_1 p + \cdots + x_{\alpha-1}p^{\alpha-1} \ (mod \, p^\alpha)$ with $0 \leq x_i < p$. To find x_0, we compute $y^{(q-1)/p}$. We get a p-th root of 1, since $y^{q-1} = 1$. Since $y = b^x$, it follows that $y^{(q-1)/p} = b^{x(q-1)/p} = b^{x_0(q-1)/p} = r_{p,x_0}$. Thus, we compare $y^{(q-1)/p} = b$ with the $\{r_{p,j}\}_{0 \leq j < p}$ and set x_0 equal to the value of j for which $y^{(q-1)/p} = r_{p,j}$.

Next, to find x_1, we replace y by $y_1 = y/b^{x_0}$. Then y_1 has discrete log $x - x_0 \equiv x_1 p + \cdots + x_{\alpha-1}p^{\alpha-1} \ (mod \ p^\alpha)$. Since $y_1$is a p-th power, we have $y_1^{(q-1)/p} = 1$ and $y_1^{(q-1)/p^2} = b^{(x-x_0)(q-1)/p^2} = b^{(x_1+x_2p+\cdots)(q-1)/p} = b^{x_1(q-1)/p} = r_{p,x_1}$. So we can compare $y_1^{(q-1)/p^2}$ with $\{r_{p,j}\}$ and set x_1 equal to the value of j for which $y_1^{(q-1)/p^2} = r_{p,j}$.

Following this procedure, we can find all $x \ mod \ p^\alpha$ for all p dividing $q-1$.

The knapsack problem. Given a set $\{v_i\}$ of k positive integers and an integer V, the knapsack problem is to find a k-bit integer $n = (\epsilon_{k-1}\epsilon_{k-2}\cdots\epsilon_1\epsilon_0)_2$ (where the $\epsilon_i \in \{0,1\}$ are the binary digits of n) such $\sum_{i=0}^{k-1} \epsilon_i v_i = V$, if such an n exists. This kind of problem may or may not have answer, and it may or may not be unique. However, for a special case of the knapsack problem, the *superincreasing knapsack problem*, the solution is unique if it exists. By *superincreasing* we mean that for each v_i, it is bigger than the sum of the previous $i-1$ terms. Uniqueness of the solution is easy to show. Suppose that there is a solution. Then we start with the biggest i to see if v_i is bigger than V or not. If it is bigger than V, then v_i cannot be in the solution. If not, then v_i must be included in the solution, since the sum of the rest v's is smaller than v_i and thus smaller than V. So if v_i is not included then the maximum of the sum of the rest v's will never give V. Proceeding downward we find ϵ_i for each i.

Now we use this nice property to build a public key cryptosystem. Each user chooses a superincreasing k-tuple $\{v_0, \cdots, v_{k-1}\}$, an integer m which is greater than $\sum_{i=0}^{k-1} v_i$, and an integer a coprime to m, $0 < a < m$. Then one computes $b \equiv a^{-1}$ *mod m*, and also computes the k-tuple $\{w_i\}$ defined by $w_i = av_i \ mod \ m$. The user keeps the numbers v_i, m, a, and b all secret, but publishes the k-tuple of w_i. The enciphering key is $K_E = \{w_0, \cdots, w_{k-1}\}$. The deciphering key is $K_D = (b, m)$

Someone who wants to send a plaintext k-bit message $P = (\epsilon_{k-1}\epsilon_{k-2}...\epsilon_1\epsilon_{k-1})_2$ to a user with enciphering key $\{w_i\}$ computes $C = f(P) = \sum_{i=0}^{k-1} \epsilon_i w_i$, and transmits that integer.

To read the message, the user looks at the least positive residue V of bC modulo m. Since $bC \equiv \sum \epsilon_i b w_i \equiv \epsilon_i v_i \ mod \ m$, it follows that $V = \sum \epsilon_i v_i$. It is then possible to use the above algorithm for superincreasing knapsack problems to find the unique solution $(\epsilon_{k-1} \cdots \epsilon_0)_2 = P$ of the problem of finding a subset of the $\{v_i\}$ which sums exactly to V.

Note that an intruder who knows only $\{w_i\}$ is faced with the knapsack problem $C = \sum \epsilon_i w_i$, which is not a superincreasing problem and hence cannot recover the message easily.

Example 5.3. Suppose that our plaintext message units are single letters with 5-bit numerical equivalents from $(0000)_2$ to $(11001)_2$. Suppose that our deciphering key is (2,3,7,15,31). Let us choose $m = 61, a = 17$, then $b = 18$ and the enciphering key is (34,51,58,11,39). To send the message 'WHY' our correspondent would compute $`W' = (10110)_2 \mapsto 51 + 58 + 39 = 148$, $`H' = (00111)_2 \mapsto 34 + 51 + 58 = 143$, $`Y' = (11000)_2 \mapsto 11 + 39 = 50$. To read the message 148,143,50, we multiply by 18 modulo 61, obtaining V=41,12,46 and recover the plaintext.

Acknowledgments. It is a pleasure to thank my mentor, Robin Walters, without whose generous help and support, I would not possibly have picked this interesting topic and finished this paper.

References

[1] Neal Koblitz. A Course in Number Theory and Cryptography. New York: Springer-Verlag, 1994.

[2] "Carmichael Number." Wikipedia. <http://en.wikipedia.org/wiki/Carmichael_number> 10 August 2011.