Simple Math: Cryptography

1 Introduction

This section develops some mathematics before getting to the application. The mathematics that I use involves simple facts from number theory. Number theory is an area of mathematics where it is possible to state a lot of interesting propositions very simply ("Are there an infinite number of prime numbers?" "For fixed $n > 2$ is it possible to find positive numbers a, b, c such that $c^n = a^n + b^n$?" "For which integers c is it possible to find integers a and b for which $c = a^2 + b^2$?), but sometimes these propositions are difficult to prove. Much of number theory developed because people were attracted by the sheer beauty and elegance of the logical activity. The mathematics was not motivated by specific applications. In fact, number theory is often thought of as the purest of pure mathematics. It is interesting, therefore, to see that some of the simple results from number theory have applications.

I plan to introduce the mathematics abstractly and then explain how it can be used to solve a "practical" problem. The mathematics is challenging because it is abstract and to follow it you will need to be able to keep track of arguments that have several steps. On the other hand, all of the steps are elementary (that is, they probably do not involve any ideas you do not know). Also, all of the general results can be illustrated by specific numerical examples. If the going gets tough, then I urge you to plug in numerical values into general formulas and see how the formulas work.

2 Mathematical Preliminaries

Some of the ideas are related to "modular" arithmetic. We say that $x \equiv a \pmod{m}$ ("x is congruent to a modulo m") if $x - a$ is divisible by m. (Throughout this section, numbers are whole numbers. When I say that y is divisible by m, I mean that there is a whole number r such that $y = r \cdot m$.) We use modular arithmetic every day when we tell time. 3:00 PM today is congruent to 3:00 PM yesterday modulo 24; it is congruent to 3:00 AM today modulo 12. Minutes and seconds are naturally organized modulo 60.

You'll need to recall three other definitions. A **prime number** is a number that is divisible only by one and itself. The **greatest common division** d of two numbers m and n, written $gcd(m, n)$ is the largest number that is a divisor of both m and n. Two numbers m and n are **relatively prime** (or **co-prime**) if $gcd(m, n) = 1$.

Here is the first result.

Proposition 1 (Chinese Remainder Theorem.) *Let m and n be relatively prime and let a and b be arbitrary. The pair of equations $x \equiv a \pmod{m}$ and*

$x \equiv b$ (mod n) have a unique solution for x (mod $m \cdot n$).

Proof Uniqueness: Suppose that x and x' are two solutions. From the first equation it follows that

$$x - x' = K_1 m \tag{1}$$

(that is, the difference between the solutions must be a multiple of m). From the second equation it follows that

$$x - x' = K_2 n. \tag{2}$$

From equations (1) and (2) it must be that $x - x'$ is a multiple of both n and m. Since $gcd(m, n) = 1$ by assumption, the least common multiple of m and n is $m \cdot n$, which means that $x \equiv x'$ (mod $m \cdot n$). This proves uniqueness.

To prove existence, consider the m numbers x_k of the form $b + kn$ for $k = 0, \ldots, m-1$. Plainly $x_k \equiv b$ (mod n) for all x. Also if $k \neq k'$ (and $0 \leq k, k' < m$), then $x_k - x'_k = (k - k')n$ is not a multiple of m (because $gcd(m, n) = 1$). It follows that x_k are m distinct numbers mod m and hence, for any a, precisely one of them (say x^*_k) satisfies $x^*_k \equiv a$ (mod m). This proves existence. *Q.E.D.*

Proposition 2 (Fermat's Little Theorem.) *Let p be prime. Then $x^{p-1} \equiv 1$ (mod p) for all x satisfying $gcd(x, p) = 1$.*

Proof List the first $p-1$ positive multiples of x: $x, 2x, 3x, \ldots, (p-1)x$. Suppose that rx and sx are the same modulo p, for $0 \leq r, s < p$. Then we have $x(r-s) \equiv 0$ (mod p). Since $gcd(x, p) = 1$, this can only happen if $r = s$. Therefore, the $p-1$ multiples of x above are distinct and nonzero; that is, they must be congruent to $1, 2, 3, \ldots, p-1$ in some order. Multiply all these congruences together and we find:

$$(p-1)!x^{p-1} = x \cdot 2x \cdot \ldots \cdot (p-1)x \equiv 1 \cdot 2 \cdot \ldots \cdot (p-1) = (p-1)! \pmod{p}.$$

Divide both side by $(p-1)!$ to complete the proof. *Q.E.D.*

Proposition 3 (Generalization of Fermat's Little Theorem) *Let p, q be two distinct primes. Let $n = p \cdot q$. Then $x^{(p-1)(q-1)} \equiv 1$ (mod n) for all x satisfying $gcd(x, n) = 1$.*

Proof

$$x^{(p-1)(q-1)} \equiv (x^{p-1})^{q-1} \equiv (1)^{q-1} \equiv 1 \pmod{p},$$

where $x^{p-1} \equiv 1$ (mod p) from Fermat's Little Theorem. Similarly, it must be that $x^{(p-1)(q-1)} \equiv 1$ (mod q).

To complete the proof, note that the Chinese Remainder Theorem implies that the solution to the two equations is unique mod $p \cdot q$. Since $x^{(p-1)(q-1)} \equiv 1$ (mod $p \cdot q$) is one solution, the theorem follows. *Q.E.D.*

The other bit of mathematics involves an algorithm for finding the greatest common divisor of two numbers.

Proposition 4 *Suppose m and n are integers with $m \geq n \geq 1$. Suppose that $m = qn + r$ for integers q and r, $0 \leq r < n$. Then $gcd(m, n) = gcd(n, r)$.*

Proof The proposition states that in order to find the greatest common divisor of two numbers, m and n, you can look instead for the greatest common divisor of n and a smaller number – the remainder after you divide m by n.

To prove the proposition you establish two things. First, you show that anything that divides both n and r must divide both n and m. This implies that $gcd(m, n) \geq gcd(n, r)$. Second, you show that anything that divides both n and m must divide both n and r. This implies that $gcd(m, n) \leq gcd(n, r)$.

To prove the first thing, let u be a divisor of both n and r. Since $m = q \cdot n + r$, u must be a divisor of m also. For the second thing, you argue in the same way: if v divides evenly into n and m, then it must also divide evenly into $r = m - q \cdot n$. *Q.E.D.*

This proposition guarantees that the following algorithm works to compute greatest common divisors.

Procedure 1 [Euclidean Algorithm.] Suppose that a_1 and a_2 are integers with $a_1 > a_2 \geq 1$. Define the decreasing sequence of integers $a_1, a_2, \ldots, a_{N+2} = 0$ by letting a_{j+2} be the remainder when we divide a_j by a_{j+1}. [That is, $a_j = q_{j+1}a_{j+1} + a_{j+2}$ with q_{j+1} and a_{j+2} integers and $a_{j+1} > a_{j+2} \geq 0$.] a_{N+1} is the first a_{k+1} with the property that a_{k+1} is a divisor of a_k. The integer a_{N+1} is equal to $gcd(a_1, a_2)$.

I need to say three things about the algorithm. First, I must show that it is well defined. The procedure that defines the elements of the sequence $\{a_j\}$ is straightforward division. So the sequence is well defined. Furthermore, it is guaranteed that $a_j > a_{j+1} \geq 0$. Therefore we must eventually get to a point at which $a_{k+2} = 0$. In this case a_{k+1} divides a_k. This means that the algorithm will stop after a finite number of steps.

Second, I must show that a_{N+1} is equal to the greatest common divisor of a_1 and a_2. By construction, $gcd(a_N, a_{N+1}) = a_{N+1}$ (because a_{N+1} is a divisor of a_N). By the previous proposition, we have $gcd(a_1, a_2) = gcd(a_2, a_3) = \ldots = gcd(a_N, a_{N+1})$, which establishes the claim.

Third, I want to assert that the algorithm is computationally efficient. One way to do this is to notice that the sequence constructed in the algorithm has the property that $\frac{a_k}{2} > a_{k+2}$ for all k. That is, the bigger number is (at least) cut in half in two steps. This means that the sequence $\{a_j\}$ shrinks exponentially: If $a_1 < 2^N$ for some N, then the algorithm returns the gcd in no more than $2N$ steps. To prove the claim that $\frac{a_k}{2} > a_{k+2}$ for all k, note that:

$$a_k = q_{k+1}a_{k+1} + a_{k+2} \geq a_{k+1} + a_{k+2} > 2a_{k+2},$$

where the equation is a definition, the first inequality follows because $q_{k+1} \geq 1$ and the second inequality follows because $a_{k+1} > a_{k+2}$.

We'll need the next proposition for our application.

Proposition 5 *If u and v are non-zero integers with $gcd(u,v) = w$, there exist integers R and S such that*

$$w = Ru + Sv.$$

Proof Let $a_1 = u$ and $a_2 = v$. Consider a sequence of equations of the form:

$$a_1 = q_2 a_2 + a_3$$

$$a_2 = q_3 a_3 + a_4$$

$$a_j = q_{j+1} a_{j+1} + a_{j+2}$$

$$a_{n-1} = q_n a_n + a_{n+1}$$

$$a_n = q_{n+1} a_{n+1}$$

The Euclidean Algorithm implies that $w = a_{n+1}$. It follows from the second-to-last of the equations that $w = a_{n-1} - q_n a_n$. We can express this relationship more grandly as: There exist integers R_{n-1} and S_{n-1} for which

$$w = R_{n-1} a_{n-1} + S_{n-1} a_n.$$

Suppose that we could find integers R_{j+1} and S_{j+1} for which

$$w = R_{j+1} a_{j+1} + S_{j+1} a_{j+2}.$$

Then we could use $a_j = q_{j+1} a_{j+1} + a_{j+2}$ to write

$$w = R_{j+1} a_{j+1} + S_{j+1}(a_j - q_{j+1} a_{j+1}) = S_{j+1} a_j + (R_{j+1} - q_{j+1} S_{j+1}) a_{j+1}.$$

Therefore, we can write

$$w = R_j a_j + S_j a_{j+1}$$

with $R_j = S_{j+1}$ and $S_j = R_{j+1} - q_{j+1} S_{j+1}$. This means that we can continue until we have

$$w = R_1 a_1 + S_1 a_2,$$

which is the desired result. *Q.E.D.*

3 Public-Key Cryptography

Cryptographers always begin explanations with quaint stories involving three people. I will not deviate from the convention.

Alice and Bob are friends. Alice wants to send Bob a secret message. Unfortunately for Alice and Bob, a third party, Eve, can hear anything Alice says. Alice wants Bob to learn the secret, but keep Eve from eavesdropping.

Abstractly, we think of Alice's secret as x (you can think of x as a number, which is perfectly general, or a message like: "Attack the west front at dawn."). What Alice does is send a function of x, call it $m = e(x)$, over a public channel. Bob can hear m, but so can Eve. Cryptography works when Bob has access to a key that allows him to figure out x from m, but Eve doesn't have the key. Mathematically, the key is a function $d(\cdot)$ that inverts the function $e(\cdot)$ so that

$$d(e(x)) = x.$$

An easy to explain, classical cryptographic method is letter substitution. In this method, Alice and Bob agree on a permutation of the letters of the alphabet (say A replaces C, B replaces K, and so on; permutation means that all letters are replaced by exactly one letter). For example, if you replace a letter by the letter three later in the alphabet (and X is replaced by A; Y by B; and Z by C), then $JOEL$ becomes $MRHO$). If Bob and Alice agreed privately on a particular letter substitution method, then one might expect that Alice could send a message to Bob that Bob could decode, but Eve wouldn't understand. If the message is long enough, however, Eve can use properties of the English language (for example, that the letter e is more common than the letter z) to crack the code efficiently. This type of code creates amusing puzzles that can be done with pencil and paper, but would be a foolish way to safeguard state secrets.

The Data Encryption Standard (DES) is a widely used system in which longer strings of letters are encrypted. Doing the encoding and decoding is more computationally intensive (that is, it is harder), but the method promises to break the power of "frequency analysis" to figure out the code. Fast computers make it possible to encode and decode messages efficiently. Unfortunately, it also makes it possible to crack codes by enumeration. (There are 26! different substitutions, for example. If Eve has a fast computer, she can program it to compute all 26! possible meanings of Alice's message and then cross check these with a dictionary to see which make sense.)

In this section, I will outline the ideas behind what is called public-key cryptography. In this system, Bob has his own secret key for decoding (k_d) and a public key for encoding (k_e). k_e is public – so it is known to everyone. k_d is known only to Bob. If Alice wants to sent the information x to Bob, she uses the public key to compute $m = k_e(x)$; Bob then figures out x by computing $k_d(m) = x$. This system is a feasible method of sending codes if it is the case that $k_d(k_e(x)) = x$, so that Bob can always decode Alice's message. It is a secure method if Bob can be confident that no one can figure out $k_d(\cdot)$ knowing

$k_e(\cdot)$. This last step is certainly not true for traditional substitution codes (or the more complicated DES method).

Here is one way to look at the public-key system. Bob has a box in front of his house. On the box is a lock. Alice (or anyone else) can go to Bob's house, drop a message in the box, and then click the lock shut. The lock acts as the public key. Now the only one who can read the message is the person with the key to the lock. This key is the private key, assumed to be available only to Bob. It is easy to imagine a lock that is easy to close but hard to open. This section provides a mathematical construction of such a system (due to Ron Rivest, Adi Shamir, and Len Adleman and called the RSA system).

The RSA method depends on numbers p, q, n, e, and d that Bob constructs. Bob selects p and q and then computes the other numbers from p and q. I assert (and will provide limited support for the assertion) that the steps are all computationally simple. First, Bob finds two "large" prime numbers p and q. The numbers must be large to make sure that it is hard to figure out $k_d(\cdot)$ knowing $k_e(\cdot)$. I will have more to say about what "large" and "hard" mean. Second, Bob computes $n = p \cdot q$. Third, Bob picks an integer $e < n$ with the property that e and $p-1$ and e and $q-1$ are co-prime. (Two numbers are co-prime if their greatest common factor is equal to one.) The pair (n, e) describes Bob's public key. Bob's secret key is (n, d), where $d = e^{-1} \bmod ((p-1)\cdot(q-1))$.

Now I must describe how the code operates. Suppose Alice wants to send a message to Bob.

1. Alice breaks the message into segments of length $\log n$. (If the large primes have 100 digits, then $\log n$ is roughly two hundred.)[1] Call a representative segment x.

2. Alice computes $x^e \pmod n$. This is the encoded message $k_e(x)$. Note that the encoding function $k_e(x) = x^e \pmod n$ depends only on the pair (n, e).

3. Bob receives the message $m = k_e(x)$. He computes $k_d(m)$ and decodes the message. I claim that $k_d(m) = m^d \pmod n$. I will return to this claim.

I have described the system. Several aspects need to be evaluated. First, is it possible to construct the key? In practice, one can take $e = 3$ in which case the only real issue is whether it is possible to come up with a pair of large primes p and q with the property than neither $p-1$ nor $q-1$ are divisible by three. There are a lot of primes and efficient methods for finding primes exist (although the algorithm depends on it being hard to factor really big numbers).

[1]Here is a pedantic procedure. Start with a message, written in English (for convenience, ignore punctuation and breaks between words). Break it up into groups of K letters. Associate each letter with its numerical position in the alphabet ($A = 1, B = 2, \ldots Z = 26$). Translate a group of K letters $k_1 k_2 \ldots k_K$ into the number $x = \sum_{i=1}^{K} 26^{i-1}(k_i - 1)$. This procedure assigns a unique integer between 0 and $26^K - 1$ to every sequence of K letters. For example, $JOEL$ becomes $9 + 14 \cdot 26 + 4 \cdot 26^2 + 11 \cdot 26^3$.

To use the proposition to compute d, we set $u = e$, $v = (p-1)\cdot(q-1)$, and use $gcd(e,(p-1)\cdot(q-1)) = 1$ to conclude that there exist k and l such that

$$1 = Re + S(p-1)\cdot(q-1). \tag{3}$$

¿From equation (3) we see that R is the inverse of e modulo $(p-1)\cdot(q-1)$. Therefore, computing d from e, p, q is not hard. Second, I must verify that the formula given for $k_d(\cdot)$ above is actually the inverse of $k_e(\cdot)$. Consider the following equations:

$$k_d(x)^d = x^{d\cdot e} = x^{1+K\cdot(p-1)\cdot(q-1)} \equiv x \pmod{n}. \tag{4}$$

Here, the first equation is the definition of $k_d(\cdot)$. By definition, $d = e^{-1}$ (mod $(p-1)\cdot(q-1)$). This means that $d = e^{-1} + c\cdot(p-1)\cdot(q-1)$, where c is a constant. Hence $d\cdot e = 1 + c\cdot e\cdot(p-1)\cdot(q-1)$, so $K = c\cdot e$ in equation (4). Finally, the generalization of Fermat's Little Theorem implies that $x^{(p-1)\cdot(q-1)} \equiv 1 \pmod{n}$, so the last equation in (4) follows (provided that x is not a multiple of p or q).

Third, I must argue that even if Eve learns m she cannot be expected to figure out x. I can't do this. I can only assert that very smart computer scientists and mathematicians believe that the only way for Eve to recover x is for her to somehow compute p and q and that the problem of inferring p and q from $p\cdot q$, that is the problem of factoring a very large number, is considered to be analytically intractable. (It would take a very fast computer using the best available methods a very long time to factor a product of two large primes. One can quantify what it means for the computer to be fast: this is the number of elementary operations it can perform in a unit of time. One can estimate the number of steps it takes the best known algorithm to factor a number of a given length. This generates an operational definition of "large." Given the current computational speed of computers, you select the length of p and q to guarantee that factoring will take a long time. As computers get faster, the definition of large changes. This discussion ignores one thing. It is conceivable that someone will discover a more efficient algorithm for factoring large numbers. It is currently believed that the number of computations needed to factor a number must increase exponentially with the length of the number. If this turns out to be false, then the foundation of the RSA procedure is destroyed.)

Finally, why did I bother to break the message into segments (Step 1)? The answer is that since the coding and decoding is done modulo n, if messages could be large relative to n, then there will be multiple interpretations when they are decoded modulo n. As it stands, there is still a tiny problem – if x is a multiple of p or q, then we cannot conduct the inversion in equation (4).[2] This is not a problem in practice.

More on Eve: Eve certainly cannot break the code by enumeration (take all possible x; compute $k_e(x)$ and compare the results to m). This would take too long. Alternatively, she could compute $d = e^{-1}$ (mod $(p-1)\cdot(q-1)$) and then

[2] Because the generalization of Fermat's Little Theorem requires that $gcd(x, p\cdot q) = 1$ in order to conclude that $x^{(p-1)\cdot(q-1)} \equiv 1 \pmod{n}$.

follow Bob's procedure for computing x from m. Doing this requires that Eve figure out $(p-1)\cdot(q-1)$. But if Eve can figure out $(p-1)\cdot(q-1)$ and she knows $n = p \cdot q$, then she can compute p and q by solving a simple quadratic equation. Again, if there is an (as yet undiscovered) efficient algorithm for computing d from the information available to Eve then the RSA method is not secure.

The value of the RSA system depends on two assertions that I have not justified (and, in fact, are still conjectures rather than mathematically established): Any method of decoding the enciphered message will give the factors of n. There is no computationally efficient way to factor n.

It is also possible to use the ideas of public-key cryptography to send a signed message. Here is one way to do this. Imagine that everyone has their own private key. Now suppose that Alice wants to send a message to Bob and "sign" it in a way that convinces Bob that the message really came from her. Here is what she can do. She takes the text that she wants to send, x, and encodes it as above. This creates the message $m = k_e^B(x)$ (where $k_e^B(\cdot)$ is Bob's public key). Next she adds a signature, $s(m)$. She signature is computed by the following formula: $s(m) = k_d^A(m)$, where $k_d^A(\cdot)$ is Alice's private key. The premise of the RSA procedure is that only Alice can compute $s(m)$ from m (that is, only Alice knows $k_d^A(\cdot)$). Now Bob receives m, which he decodes using his private key. This is the secret message. To confirm that the message came from Alice, he "decodes" the signature $s(m)$ using Alice's public key: $k_e^A(s(m)) = k_e^A(k_d^A(m)) = m$. If the decoded signature agrees with the message, then Bob can be confident that the sender had access to Alice's private key.

There are several other remarkable things that the RSA procedure make possible. I'll supply two "key words" and you can search for more information if interested. "Mental Poker" permits you to play poker – and be confident that your opponent is not cheating – at a distance, say over the internet. "Zero-knowledge Proofs" allow someone to demonstrate that a mathematical proposition is true without actually supplying the details of the argument.