Cryptography

People have been sending and receiving secret messages for thousands of years. Ever since the beginning of the Information Age, the importance of secret transmissions has been growing very rapidly – while governments still want to hide their secret plans, every internet user relies on the secret transmission of messages in one form or another, like online transactions or sending personal information. This art of secret communication is what we now call cryptography.

In this section we will explore the role that Number Theory plays in cryptographic protocols. First, we will begin with some definitions. A **cipher** is a technique that transforms a message in order to conceal its original meaning, usually working fundamentally, replacing each letter with another letter. **Enciphering** transforms a **plaintext** message to a **ciphertext**, and **deciphering** is the reverse process.

Dating back to Ancient Greece, the **Caesar Cipher** is one of the earliest documented cipher processes. Historical records show that Julius Casear used this technique for military purposes. The cipher is quite simple – it replaces each letter of the alphabet with another one that is **k places down** the alphabet. For example, if k=3, then here is the corresponding substitution.

Plain alphabet	a b c d e f g h i j k l m n o p q r s t u v w x y z
Cipher alphabet	d e f g h i j k l m n o p q r s t u v w x y z a b c

So a message such as "veni, vidi, vici" will be encrypted as "yhql, ylgl, ylfl". The decryption scheme is very easy once you know k. However, with a modern day computer, this cipher is extremely weak, as we can simply try all possibilities for k (for English, $k \leq 26$), and for any reasonably sized text, only one of those will produce anything legible.

From the Caesar Cipher onwards, cryptography evolved rapidly. Most encryption schemes are based on scrambling a message in some way and substituting letters based on a key. The key is used to encrypt and decrypt a message, and must be sent along some private channels. Thus, the key of a cipher is very important. Indeed, much of the Allies success during the Second World War was based on the fact they they could figure out the key used by the Germans in their military communications.

The invention of the modern day computer made it possible to invent "perfect" scrambling algorithms that are so strong that the key can no longer be determined by analysing the encryption scheme and other relevant information. The best known method of decrypting a message encrypted using one of these schemes is to just try all possible keys. Thus, the strength of these cryptosystems is directly proportional to the number of keys available. The Data Encryption Standard (DES) was established in 1976, detailing a cryptosystem that is both strong and practical. The DES uses a 56-bit key, a size that makes it nearly impossible to break within the civilian community, but one that the government can just about be able to handle with their computing resources.

As strong as the DES is, the problem of transmitting keys safely still persists. The internet poses yet another problem – messages are being transmitted near instantaneously along wires that can easily be tapped. The problem of transmitting keys safely and efficiently is the **Key Distribution** problem. For a while, the feasibility of the key distribution seemed counter-intuitive; we want to be able to send a short message from one place to another, and yet it is virtually impossible to prevent anyone from eavesdropping on the data being sent. However, all doubts were casted away when Whitfield Diffie and Martin Hellman developed the nearly perfect solution: the **Diffie-Hellman key exchange** scheme.

Secret Key Exchange

Suppose Alice wants to send a key to Bob, and they know that someone named Eve is eavesdropping on the transmission line. The Diffie-Hellman key exchange scheme uses the idea of a **one-way** function to solve this problem. A one-way function is one that is easy to calculate, but whose inverse is very hard to calculate. The function that the scheme uses is

$$f(a,x) = a^x \pmod{p} \quad ,$$

where p is a prime number, and a is a **suitably** chosen value with respect to p. Knowing these two values, we can compute $f(a,x)$ easily for any x using the `powmod()` function. Reversing the function, however, is difficult when p is large, and is known as the **discrete logarithm problem.** It is widely believed that computing discrete logarithms is as hard as factoring integers.

The Diffie-Hellman Key Exchange Scheme

Step 1) Alice and Bob agree on some p and choose a suitable a. Eve eavesdrops and learns both values.

Step 2) Alice picks a number X and keeps it secret to herself, and Bob picks a number Y and keeps it secret to himself. Then Alice computes $a^X \pmod{p}$, and Bob computes $a^Y \pmod{p}$.

Step 3) They transmit the computed values from Step 2 to each other. Eve eavesdrops and now knows both $a^X \pmod{p}$ and $a^Y \pmod{p}$, but she **cannot** figure out what X and Y are without computing the discrete logarithm.

Step 4) Alice now has $a^Y \pmod{p}$, so she computes $(a^Y)^X \equiv a^{XY} \pmod{p}$ because she knows X. Bob, on the other hand, computes $(a^X)^Y \equiv a^{XY} \pmod{p}$. Notice that both Alice and Bob end up with the same value $key = a^{XY} \pmod{p}$.

With the information that Eve now has, she only needs either X or Y to determine the key. However, neither X nor Y are ever transmitted, and because there is no known efficient algorithm for solving the discrete logarithm problem when p is large, it is **very** difficult for Eve to determine the key (a lot more difficult than it is for Alice and Bob to transmit the key).

One major problem exists in this scheme – both Alice and Bob must be **active** for a transmission to take place, because Alice cannot run her encryption algorithm (e.g. DES) until she knows the key, and she doesn't know the key until Bob participates in the key exchange. In particular, this makes it hard to send secret emails, or leave secret messages.

Many researchers were amazed at the Diffie-Hellman key exchange scheme because it seemed so counter-intuitive, and yet it worked so well. They immediately began to tackle the remaining puzzle – to develop an encryption protocol that does not involve simultaneous participation. One year later, the idea of **public-key** cryptography was invented, creating the now famous **RSA cryptosystem**.

Public Key Cryptography

So far, all encryption schemes that we have discussed use the same key for encryption and decryption. The key is the only extra information we use to encrypt a message, and it seems natural that we must know the key in order to reverse the process of encryption. However, if we have separate keys for encryption and decryption, many problems can be resolved. For example, when Alice wants to send a message to Bob, Bob just creates a pair of keys, (e, d), used for encryption and decryption respectively, and sends Alice the encryption key. As long as we cannot determine d from e easily, this encryption scheme solves the key distribution problem, and it doesn't even require Alice and Bob to be transmitting simultaneously beucause Bob can generate the pair of keys offline and publish e on some public website.

A cryptosystem as described above is a **public-key cryptosystem**. It has a **public** key, e, that everyone can know and use, and a **private** key, d, that Bob never publishes. The strength of such a cryptosystem is depended on how hard it is to determine d from e. The first practical public-key cryptosystem was invented in 1977 by three researchers from MIT – Ronald Rivest, Adi Shamir, and Leonard Adleman – and was subsequently called the **RSA cryptosystem**.

The RSA cryptosystem works by first chopping up the message evenly into **binary letters**, and then encrypts each letter separately. After we decide how to chop up the message, we know how big the letters can get, so we need to pick a modulus n that is bigger than any of these letters.

The RSA Algorithm

Setup
Choose $n=pq$ where p and q are both prime numbers. Make sure n is big enough to accommodate all the letters. Figure out that $\phi(n)=(p-1)(q-1)$.

Finding keys
Bob first chooses a **private** key, d , and figures out the **public** key, e , by solving

$$de\equiv 1(mod\,\phi(n))$$.

This equation is solvable if and only if $gcd(d,\phi(n))=1$, so Bob must pick d that is relatively prime to $\phi(n)$. Bob now publishes e .

Encryption
For each letter, a , Alice encrypts it to another letter, b , using

$$b\equiv a^e(mod\,n)$$.

Decryption
For each letter, b , that Bob receives from Alice, Bob can figure out the original letter, a , from

$$a\equiv b^d(mod\,n)$$.

Proof of Correctness
Recall from last class the following Theorem:

If $x\equiv y(mod\,\phi(n))$, and $gcd(a,n)=1$, then $a^x\equiv a^y(mod\,n)$.

This theorem ties everything together in the RSA algorithm. We only need to show that the decryption scheme does reverse the encryption scheme. Indeed, whenever $gcd(a,n)=1$, we have

$$b^d\equiv(a^e)^d\equiv a^{de}(mod\,n)$$,

and because $de\equiv 1(mod\,\phi(n))$ from the F*inding keys* step, we get

$$b^d\equiv a^{de}\equiv a(mod\,n)$$.

There are several things that we still need to discuss regarding the RSA algorithm. First, how can we be certain that $gcd(a,n)=1$? Well, since $n=pq$ is just a product of two primes, and we chose n big enough so that $a<n$, we know that the only bad cases are when a is either p or q . We can deal with these cases easily by substituting a for some other letter that we know does not exist in any message. This is the benefit of making n simple – only very few letters are "bad" letters.

Given the public key, e , and the modulus n , how hard is it to determine the decryption key d ? It turns out that once we know $\phi(n)$, we can determine d easily, since we just need to compute

$$de \equiv 1 (mod \phi(n)) ,$$

only this time we start with e to obtain d . The **hard** part, however, is determining $\phi(n)$.

We want n to be simple (having few factors) because we don't want too many letters a to share a factor with n . Then, why don't we choose n to be a prime? This is because fast algorithms exist to test whether a number is prime, and once n is determined to be a prime, we know immediately $\phi(n)=n-1$. Consequently, d can be determined from e when n is a prime.

But, making n just a bit more complex by setting it as a product of two primes is enough to make computations of $\phi(n)$ a lot more difficult. Try doing it yourself – try solving for $\phi(n)=pq-p-q-1$ when you only know that $n=pq$. The only known way of solving this is to factor n completely to determine the two prime factors, and once you factor n , you can compute $\phi(n)$ with equations we discussed last time.

Hence, the RSA cryptosystem is safe as long as p,q are big enough that we cannot determine d from e easily. This means that the safety of the RSA cryptosystem depends on how fast we can compute $\phi(n)$, and computing $\phi(n)$ requires us to **factor** n completely. Many algorithms have been developed for solving the factoring problem quickly, but none of them are polynomial time. In the current state, a 2048-bit number for n is fairly safe and efficient for general purposes.

I recommend the reader consult [1] for a general overview of the history of cryptography, and [2] for a rigid treatment of the mathematics behind all this, and the internet for online resources regarding RSA and public-key cryptography in general.

Resources

[1] Singh, Simon. The Code Book. New York: Anchor Books, 1999.
[2] Niven, Ivan, Herbert S. Zuckerman, and Hugh L. Montgomery. An Introduction to the Theory of Numbers. John Wiley & Sons, Inc., 1991.