Lecture 11: Cryptography

Cryptography is the theory of **codes**. Important two aspects of the field are the **encryption** of information and **error correction**. Both aspects are crucial in daily life: when getting access to a computer, viewing a bank statement or when taking money from the ATM, encryption algorithms are used. When phoning, surfing the web, accessing data on a computer or listening to music, error correction algorithms are used. Since our lives have become more and more digital: music, movies, books, journals, finance, transportation, medicine, communication have all turned digital, we rely on strong error correction to avoid errors and encryption to assure things can not be tempered with. Without error correction, airplanes would crash: every small error in the memory of a computer would produce glitches in the navigation and control program. Without error correction music would sound like a 1920 gramophone record. Without encryption, everybody could enter the now mostly electronic banks and take money. Without encryption, all medical data and personal conversations would be public. Before the digital age, error correction was assured by extremely redundant information storage. Writing a letter on a piece of paper displaces billions of billions of molecules in ink. Now, changing any single bit gives the letter a completely different meaning. Before the digital age, information was kept in well guarded safes which were physically difficult to penetrate. Now, information is locked up on computers which are connected to other computers. Vaults, money or voting ballots are secured by mathematical algorithms which assure that information can only be entered accessed by the right users. Also life needs error correction: information in the genome is stored in a **genetic code**, where a error correction makes sure that life can survive. A cosmic ray hitting the skin changes the DNA of a cell, but in general this is harmless. Only a larger amount of radiation can render cells cancerous.

How can an encryption algorithm be safe? One possibility is to invent a method and keep it secret. An other is to use a well known encryption method and rely on the **difficulty of mathematical computation tasks** to assure that the method is safe. History has shown that the first method is unreliable. Systems which rely on "security through obfuscation" usually do not last long. The reason is that it is tough to keep a method secret if the encryption tool is distributed. Reverse engineering of the method is often possible, for example using plain text attacks. Given a map T, a third party can compute many pairs $x, T(x)$ and by choosing specific texts to find out what happens.

The **Caesar cypher** permutes letter of the alphabet. For example, replace every letter A with B, every letter B with C etc and every letter Z with A. The word "Mathematics" becomes encrypted as "Nbuifnbujdt". Caesar would shift the letters by 3. The right shift by one was used by his Nephew Augustus. **Rot13** shifts by 13, the **Atbash cypher** would reflect the alphabet and switch A with Z, B with Y etc. The alphabet does not have to be in the same order. Using one of the $26! = 403291461126605635584000000 \sim 10^{27}$ permutations makes it appear that a brute force attack is not possible. Cesar cyphers can be cracked quickly using statistical analysis however. We know the frequency with which letters appear. Matching the frequency of a text with this gives enough clues to connect the dots. The **Trithemius cypher** prevents this simple analysis by changing the permutation in each step. It is called a polyalphabetic substitution cypher. Instead of a simple permutation, there are many permutations. After transcoding a letter, we also change the key. Lets take a simple example. Rotate for the first letter the alphabet by 1, for the second letter, the alphabet by 2, for the third letter, the alphabet by 3 etc. The word "Mathematics" becomes now "Ncwljshbrmd". Note that the second "a" has been translated to something different than a. A simple frequency analysis is now no more possible. The **Viginaire cypher** adds additional complexity to this: instead of shifting the alphabet by 1, we can take a key like "BCNZ", then shift the first letter by 1, the second letter by 3 the third letter by 13, the fourth letter by 25 the shift the 5th letter by 1 again. Viginíre's cypher was unbroken for hundreds of years. But a more sophisticated frequency analysis which involves first finding the length of the key length makes the cypher breakable. With the emergence of computers, even more sophisticated versions like the German **enigma** had no chance.

Diffie-Hellman key exchange allows Ana and Bob want to agree on a secret key over a public channel. The two palindromic friends agree on a prime number p and a base a. Ana choses a secret number x and sends $X = a^x$ modulo p over the channel. Bob choses a secret number y and sends $Y = a^y$ modulo p to Ana. Ana can compute Y^x and Bob can compute X^y but both are equal to a^{xy}. This is their common secret. Eve, the eves dropper can not compute this number. The only information available to Eve are X and Y, the base a and p. Eve knows that $X = a^x$ but can not determine x. The key difficulty in this code is the **discrete log problem**: we have to get x from a^x is believed to be difficult.

The **Rivest-Shamir-Adleman public key system** uses a **RSA public key** (n, a) with an integer $n = pq$ and $a < (p-1)(q-1)$, where p, q are prime. The numbers n and a are known to everybody. Only the factorization of n is kept secret. Ana publishes this pair. Bob who wants to email Ana the message x, sends her $y = x^a \bmod n$. Ana, who has computed b with $ab = 1 \bmod (p-1)(q-1)$ can read the secrete email y because $y^b = x^{ab} = x^{(p-1)(q-1)} = x \bmod n$. But Eve, has no chance because the only thing Eve knows is y and (n, a). It is believed that without the **factorization** of n, it is not possible to determine x. The message has been transmitted securely. The core difficulty is that **taking roots** in the ring $Z_n = \{0, ..., n-1\}$ is difficult without knowing the factorization of n. With a factorization, we can quickly take arbitrary roots. If we can take square roots, then we can also factor: assume we have a product $n = pq$ and we know how to take square roots of 1. If x solves $x^2 = 1 \bmod n$ and x is different from 1, then $x^2 - 1 = (x-1)(x+1)$ is zero modulo n. This means that p divides $(x-1)$ or $(x+1)$. To find a factor, we can take the greatest common divisor of $n, x-1$. Take $n = 77$ for example. We are given the root 34 of 1. ($34^2 = 1156$ has reminder 1 when divided by 34). The greatest common divisor of $34 - 1$ and 77 is 11 is a factor of 77. Similarly, the greatest common divisor of $34 + 1$ and 77 is 7 divides 77. Finding roots modulo a composite number and factoring the number is equally difficult.

Cipher	Used for	Difficulty	Attack
Cesar	transmitting messages	many permutations	Statistics
Viginere	transmitting messages	many permutations	Statistics
Enigma	transmitting messages	no frequency analysis	Plain text
Diffie-Helleman	agreeing on secret key	discrete log mod p	Unsafe primes
RSA	electronic commerce	factoring integers	Factoring

The simplest **error correcting code** uses 3 copies of the same information. A single error can be corrected. With 3 watches for example, you know the time even if one of the watches fails. Cockpits of airplanes have three copies important instruments. This basic error correcting code is not very efficient. It can correct single errors by tripling the size. Its efficiency is 33 percent.
A cheap way to make it more efficient is to compress the data first and then make three copies. **Data compression** is a topic by itself. Here is a simple example, the **dictionary compression**. Take dictionary with $65'536 = 2^{16}$ words for example. Every word can be encode by two bytes. Assuming an average word length of 6, we can encode every word with 2 bytes instead of 6. There are better error correcting codes using linear algebra or algebraic geometry.