CS 387
Applied Cryptography

David Evans

written by

Daniel Winter

16.04.2012

Contents

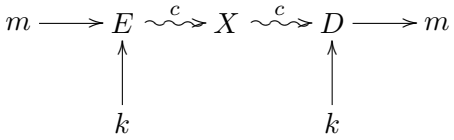
1 Unit 1

1.1 Cryptology, Symmetric Cryptography, Correctness Property

Definition *cryptography, cryptology*
cryptography comes from Greek with *crypto* means "hidden, secret" and *graphy* means "writing". A broader definition is *cryptology* with Greek "-logy" means "science".
Example 1:
These actions involve cryptology:

- Opening a door
- Playing poker
- Logging into an internet account

.
Definition *symmetric Cryptography*
Symmetric Cryptography menas all parties have the same key to do encryption and decryption.
Definition *symmetric Cryptosystem, decryption function, encryption function*
In this paper a *symmetric Cryptosystem* always looks as follows:



where m is a plaintext message from the set $\mathcal{M}$ of all possible messages, k is the key from the set $\mathcal{K}$ of all possible keys and c is a ciphertext from the set $\mathcal{C}$ of all possible Ciphertexts. E ist the *encryption function* and D is the *decryption function* with:

$$\begin{aligned} E &: M \times K \to C : (m,k) \mapsto c \\ D &: C \times K \to M : (c,k) \mapsto m \end{aligned}$$

X is a possible eavesdropper of the insecure channel.
Definition *correctness property*
In order we get the same message after decrypting the ciphertext we need the *correctness property*: $\forall m \in \mathcal{M}, \forall k \in \mathcal{K}$

$$D_k(E_k(m)) = m \Leftrightarrow E_k = D_k^{-1}$$

Example 2:
These functions satisfy the correctness property for a symmetric cryptosystem with $\mathcal{M} = \mathcal{K} = \mathbb{N}$

1. $E_k(m) = m + k, D_k(c) = c - k$
2. $E_k(m) = m, D_k(c) = c$

because

1. $D_k(E_k(m)) = D_k(m + k) = m + k - k = m$
2. $D_k(E_k(m)) = D_k(m) = m$

.

1.2 Kerchoff's Principle, *xor*-function

Theorem 1.2.1 (Kerchoff's Principle)
Even if the encryption function E and decryption function D are public, the message is still secure due to a usage of a key. Only the key has to be private. If the key gets public you have to use another key. So the keys must be kept secret in a cryptosystem.

Definition *xor function*
the *xor function* or *exclusive or* (symbol: $\oplus$) is giben by it's truth table:

A	B	$A \oplus B$
0	0	0
1	0	1
0	1	1
1	1	0

Example 3:
$\forall x, \forall y$:

- $x \oplus y \oplus x = y$
- $x \oplus x = 0$

and these two very important properties of the *xor function* with $\forall x, \forall y, \forall m \in \mathcal{M}, \forall k \in \mathcal{K}, \forall c \in \mathcal{C}$:

- $x \oplus y \oplus x = x \oplus x \oplus y$
- $m \oplus k = c$ and $c \oplus k = m$

show that the *xor function* is kommutative and assoziative and how to compute the ciphertext c with the message m and the key k and decrypt the ciphertext c with the same key k to get m.

1.3 One - Time Pad

Definition *One - Time Pad*
Let's assume that the set of all possible messages $\mathcal{M} := \{0,1\}^n$. That means every message is represented as a string of zeros and ones with fixed length $n \in \mathbb{N}$. Therefore n is the maximum length of a message $m \in \mathcal{M}$. Any key $k \in \mathcal{K}$ has to be as long as the message $m \in \mathcal{K}$. It follows that $\mathcal{K} := \{0,1\}^n$. The encryption function with $m = m_0 m_1 \dots m_{n-1}$ and $k = k_0 k_1 \dots k_{n-1}$ looks as follows:

$$E : M \times K \to C : (m,k) = m \oplus k = c_0 c_1 \dots c_{n-1} = c$$

with the value for eacht bit of the ciphertext is defined as: $\forall i \in \{0,1,\dots,n-1\}$

$$c_i = m_i \oplus k_i$$

Example 4:
Let assume our message is $m =$'CS'. We have to convert the string to an element of $\mathcal{M} := \{0,1\}^n$ where $n = 7$. That means every charakter in every message is represented by 7 bits. `Python` provides a built-in function `ord(<one charakter string>)` which maps every charakter to a specific dezimal number. The code for converting a string to a valid message looks as follows:

```
def convert_to_bits(n,pad):
   result = []
   while n > 0:
      if n % 2 == 0:
         result = [0] + result
      else:
         result = [1] + result
      n = n / 2
   while len(result) < pad:
      result = [0] + result
   return result

def string_to_bits(s):
   result = []
   for c in s:
      result = result + convert_to_bits(ord(c),7)
   return result

string_to_bits('CS') => [1,0,1,0,0,1,1,1,0,0,0,0,1,1]
```

and it follows with a random choosen key k:

$$\begin{aligned} m = \text{'CS'} &= \overbrace{1010011}^{\text{C}}\overbrace{1000011}^{\text{S}} \\ &\oplus \\ k &= 11001000100110 \\ &|| \\ c &= 01101111100101 \end{aligned}$$

If someone got the ciphertext but not the key - the person is not able to figure the original message out. Taking c and another key $k = 11001010100110$ and trying to get the message $\forall i \in \{0, 1, \ldots, n-1\}$:

$$c_i \oplus k_i = m_i \Rightarrow m = 10100101000011$$

if m is separated in 2 parts of length 7: 1010010 and 1000011, convert each to a decimal number and apply the built-in `Python` function `chr(<number>)=ascii charakter` the result is 'BS' instead of the correct string 'CS'.

1.4 Probability

Definition *Probability Space*
The *probability space* Ω is the set of all possible outcomes.
Example 5:

- Flipping a coin: $\Omega = \{(H)eads, (T)ail\}$
- Rolling a dice: $\Omega = \{1, 2, 3, 4, 5, 6\}$

.
Definition *Uniform Distribution*

If the probability space has a *uniform distribution* that menas each outcome has equal probability.

Definition *Probability Function*

The *Probability Function* is a function that maps each outcome to a non-negative value lower-equal than 1. That means

$$P_i : \Omega \to [0,1] : \omega \mapsto [0,1]$$

where $\omega \in A$ is an outcome. If $P_i(\omega) = 0$ that means that ω never happens (e.g. roll 7 on a dice). If $P(\omega) = 1$ that means that ω alyways happens (e.g. the outcome of a coinflip is in $\{H, T\}$)

Theorem 1.4.1:

Assume the probability space Ω and the probability function p. Then it is

$$\sum_{\omega \in \Omega} P(\omega) = 1$$

Example 6:

Let's assume $\Omega = \{Heads, Tails, Edge\}$ with the probabilities

$$\begin{aligned} P(H) &= 0.49999 \\ P(T) &= 0.49999 \end{aligned}$$

The probability for edge E is given by

$$1 = P(\Omega) = P(H) + P(T) + P(E) = 0.49999 + 0.49999 + P(E) \Rightarrow P(E) = 0.00002$$

Definition *Event*

An *Event* $\mathcal{A}$ is a subset of outcomes from a probability space.

Example 7:

An Event of tossing a coin woult be landing on heads, therefore $\mathcal{A} = \{H\}$. A valid coin toss is the set of outcomes $\mathcal{A} = \{H, T\}$

Theorem 1.4.2:

The probability of an event $\mathcal{A}$ is given by

$$P(\mathcal{A}) = \sum_{\omega \in \mathcal{A}} P(\omega)$$

Example 8:

The probability for a valid coin toss $\mathcal{A} = \{H, T\}$ is

$$P(\mathcal{A}) = P(H) + P(T) = 0.49999 + 0.49999 = 0.99998$$

Definition *Conditional Probability*

Given two events, $\mathcal{A}$ and $\mathcal{B}$, in the same probability space, the *conditional probability* of $\mathcal{B}$ given that $\mathcal{A}$ occured is:

$$P(\mathcal{B}|\mathcal{A}) = \frac{P(\mathcal{A} \cap \mathcal{B})}{P(\mathcal{A})}$$

Example 9:

Given that a coin toss is valid, the probability it is heads is given with $\mathcal{A} = \{H, T\}$ is the event that a coin toss is valid and $\mathcal{B} = \{H\}$ is the event that the coin toss is heads. It follows with $P(\mathcal{A}) = P(\{H, T\}) = P(\{H\}) + P(\{T\}) = 0.49999 + 0.49999 = 0.99998$ and $P(\mathcal{B}) = P(\{H\}) = 0.49999$:

$$P(\mathcal{B}|\mathcal{A}) = \frac{P(\mathcal{A} \cap \mathcal{B})}{P(\mathcal{A})} = \frac{P(\{H, T\} \cap \{H\})}{P(\{H, T\})} = \frac{P(\{H\})}{P(\{H, T\})} = \frac{0.49999}{0.99998} = \frac{1}{2}$$

1.5 Perfect Cipher

Definition *perfect cipher*
The ciphertext provides an attacker with **no** additional information about the plaintext. Assume $m, m^* \in \mathcal{M}, k \in \mathcal{K}, c \in \mathcal{C}$. The property for a *perfect cipher* is given by

$$P[m = m^*|E_k(m) = c] = P[m = m^*]$$

That means: for an attacker/eavesdropper the probability that $m = m^*$ without knowing the ciphertext is equal $m = m^*$ with knowing the ciphertext. The property

$$P[m = m^*|E_k(m) = c] = \frac{1}{|\mathcal{M}|}$$

where $|\mathcal{M}|$ is the cardinality of $\mathcal{M}$ (number possible messages). This woult be correct if, a priori, the attacker knew nothing about the messages, therefore all messages are equally likely (whats obviously not correct - not all sentences make sense).
Theorem 1.5.1:
The One-Time Pad is a perfect cipher.

Proof: Remember the perfect cipher property:

$$P[m = m^*|E_k(m) = c] = P[m = m^*]$$

and the definition of the conditional probability:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

It follows with $A = (m = m^*)$ and $B = (E_k(m) = c)$

$$P(B) = P(E_k(m) = c) = \sum_{m_i \in \mathcal{M}} \sum_{k_i \in \mathcal{K}} \frac{P(E_{k_i}(m_i))}{|\mathcal{M}| \cdot |\mathcal{K}|}$$

For any message-ciphertext pair, there is only one key that maps that message to that ciphertext, therefore

$$\sum_{k_i \in \mathcal{K}} P(E_{k_i}(m) = c) = 1$$

and summing over all messages the value of 1 leads to

$$P(B) = P(E_k(m) = c) = \sum_{m_i \in \mathcal{M}} \sum_{k_i \in \mathcal{K}} \frac{P(E_{k_i}(m_i))}{|\mathcal{M}| \cdot |\mathcal{K}|} = \frac{|\mathcal{M}| \cdot 1}{|\mathcal{M}| \cdot |\mathcal{K}|} = \frac{1}{|\mathcal{K}|}$$

That's the probability of event B, wich is the probability that some message encrypts to some key (computet over all the messages).
Then

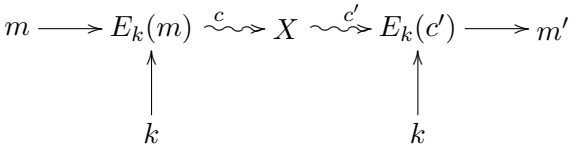
$$P(A \cap B) = P(m = m^* \cap E_k(m) = c) = P(m = m^*) \cdot P(k = k^*) = P(m = m^*) \cdot \frac{1}{|\mathcal{K}|} = \frac{P(m = m^*)}{|\mathcal{K}|}$$

to see this consider $k^* \in \mathcal{K}, m^* \in \mathcal{M}$ and the distribution of $\mathcal{M}$ is not normal (not all messages are equally likely) and every key maps each message to only one ciphertext and the keys are equally likely (the distribution of the keys is normal), therefore $P(k = k^*) = \frac{1}{|\mathcal{K}|}$. Plug all together in the conditional probability formula gives

$$P[m = m^* | E_k(m) = c] = \frac{P(m = m^* \cap E_k(m) = c)}{P(E_k(m) = c)} = \frac{\frac{P(m=m^*)}{|\mathcal{K}|}}{\frac{1}{|\mathcal{K}|}} = P(m = m^*)$$

Which is the definition of the perfect cipher. It follows the One - Time Pad is a perfect cipher
□ **Definition** *malleable cipher, impractical cipher*

- *malleable*:
  The encryptet message $E_k(m) = c$ can be modified by an active attacker X, that means



- *impractical*:
  The One - Time Pad is very impractical, because the keys have to be as long as the messages, and a key can never be reused. That means

$$|\mathcal{K}| = |\mathcal{M}|$$

  In general, a cipher is impractical iff (if and only if)

$$|\mathcal{K}| \geq |\mathcal{M}|$$

  Unfortunately, Claude Shannon proved that finding a practical perfect cipher is impossible.

.
Theorem 1.5.2 (Shannon's Theorem)
Every perfect cipher is impractical.

Proof: Proof by contradiction: Assume to have a perfect cipher that does not satisfy the impractical property. That's equal to:
Suppose E is a perfect cipher where $|\mathcal{M}| > |\mathcal{K}|$.
Let $c_0 \in \mathcal{C}$ with $P(E_k(m) = c) > 0$. That means there is some key that encrypts some message c_0.
Decrypt c_0 with all $k \in \mathcal{K}$ with the decryption function D (not necessarily the same as E).
Since the cipher is correct - in order to be perfect it has to both be correct and perfectly secure. That means the decryption function must have the property

$$D_k(E_k(m)) = m$$

Let

$$\mathcal{M}_0 = \bigcup_{k \in \mathcal{K}} D_k(c_0)$$

Therefore $\mathcal{M}_0$ is the set of all possible messages decrypting c_0 with every key $k \in \mathcal{K}$ (brute-force attack). It follows

(a) $|\mathcal{M}_0| \leq |\mathcal{K}|$
Because of construction of $\mathcal{M}_0$ (union over all keys)

(b) $|\mathcal{M}_0| < |\mathcal{M}|$
Because of the assumption $|\mathcal{M}| > |\mathcal{K}|$ and combined with $(a) : |\mathcal{M}_0| \leq |\mathcal{K}|$.

(c) $\exists m^* \in \mathcal{M} : m^* \notin \mathcal{M}_0$
Follows exactly from $(b) : |\mathcal{M}_0| < |\mathcal{M}|$

Considering the perfect cipher property

$$P[m = m^* | E_k(m) = c] = P[m = m^*]$$

Due to $(b) : P(m = m^*) = 0$ but due to $(c) : m^* \in \mathcal{M} \Rightarrow P(m = m^*) \neq 0$
We have contradicted the requirement for the pefect cipher. Therefore the assumption $|\mathcal{M}| > |\mathcal{K}|$. Thus:
There exists **no** perfect ciphers where

$$|\mathcal{M}| > |\mathcal{K}|$$

Therefore every cipher that is perfect must be impractical. □

1.6 Lorenz Cipher Machine

Theorem 1.6.1:
Given two cipertexts $m \oplus k = c = c_0 c_1 \dots c_{n-1}, m' \oplus k = c' = c'_0 c'_1 \dots, c'_{n-1}$ with $c, c' \in \mathcal{C}$ and $\exists j \in I := \{0, 1, 2, \dots, n-1\}$:

$$c_j \neq c'_j$$

then by xor'ing $c \oplus c' = m \oplus k \oplus m' \oplus k = m \oplus m'$. If there is only a slightly difference between c and c' it is possible by guessing $m^* \sim m$ and getting a possible message via (xor'ing with the intercepted cipertexts)

$$m^* \oplus c \oplus c'$$

which should give back the other message m'.
Once the two messages m, m' are given, it's easy to get the key with

$$k = m \oplus c$$

Definition *Lorenz Cipher Machine*

Each letter of the message m would be divided into 5 bits $m_0 m_1 m_2 m_3 m_4$, and those would be *xor'd* with the value coming from the corresponding and different sized k-wheels which had at each position a 0 or a 1. The result would also be *xor'd* with the value of the s-wheels, which worked similarly. The k-wheels turned every charakter, the s-whells turned conditionally on the result of 2 other wheels, which were the m_1-wheel (which turned every time) and the m_2-wheel (which would rotate depending on the value of the m_1-wheel) and depending on the $m_1 \oplus m_2$ either all the s-wheels would rotate by 1 or none of them would. The result of all these *xors* is

the cipher text $c = c_0c_1c_2c_3c_4$.
The Lorenz Cipher works similarly to an One - Time Pad: *xor'ing* a message with a key leads to a ciphertext.
Knowing the structure of the machine is not enough to break the cipher. It's necessary to know the initial configuration.

Example 10:
Let $z = z_0z_1z_2z_3z_4z_5z_6z_7 \dots z_{n-1}$ be the interceptet message. The ciphertext z is broken into 5 channels c where each bit on position i with $i \in \{0, 1, 2, \dots, n-1\}$ is transmitted over channel $(i+1) \mod 5$. Thus for 5 channels c_1, c_2, c_3, c_4, c_5:

$$\begin{aligned} c_1 &\rightarrow z_0z_5z_{10}z_{15}\dots \\ c_2 &\rightarrow z_1z_6z_{11}z_{16}\dots \\ c_3 &\rightarrow z_2z_7z_{12}z_{17}\dots \\ c_4 &\rightarrow z_3z_8z_{13}z_{18}\dots \\ c_5 &\rightarrow z_4z_9z_{14}z_{19}\dots \end{aligned}$$

So channel 1 transmit the first part of the first letter z_0, the first part of the second letter z_5, channel 2 transmit the second part of the first letter z_1, the second part of the second letter z_6, and so on.
Now subscripting z by th channel and the letter for that channel $z_{c,i}$. It follows

$$\begin{aligned} z_{0,i} &= z_0, z_5, z_{10}, \dots \\ z_{1,i} &= z_1, z_6, z_{11}, \dots \end{aligned}$$

The subscripts break up the ciphertext into channels and therefore, with the weakness of the cipher (all s-wheels move in turn). Thus

$$z_{c,i} = m_{c,i} \oplus k_{c,i} \oplus s_{c,i}$$

and by separating the ciphertext into these 3 pieces, it's possible to take advantage of properties that they have. The importance is that the s-wheels don't always turn. Looking at subsequent characters, there is a good chance that the s-wheels have not changed. Let's define

$$\Delta z_{c,i} := z_{c,i} \oplus z_{c,i+1}$$

notice that $z_{c,i}, z_{c,i+1}$ are 5 characters apart in the interceptet ciphertext, but they are adjacent for that channel. It follows

$$\begin{aligned} \Delta z_{0,i} \oplus \Delta z_{1,i} &= z_{0,i} \oplus z_{0,i+1} \oplus z_{1,i} \oplus z_{1,i+1} \\ &= m_{0,i} \oplus k_{0,i} \oplus s_{0,i} \oplus m_{0,i+1} \oplus k_{0,i+1} \oplus s_{0,i+1} \oplus m_{1,i} \oplus k_{1,i} \oplus s_{1,i} \oplus m_{1,i+1} \oplus k_{1,i+1} \oplus s_{1,i+1} \\ &= \underbrace{m_{0,i} \oplus m_{0,i+1} \oplus m_{1,i} \oplus m_{1,i+1}}_{=:\Delta m} \oplus \underbrace{k_{0,i} \oplus k_{0,i+1} \oplus k_{1,i} \oplus k_{1,i+1}}_{=:\Delta k} \oplus \underbrace{s_{0,i} \oplus s_{0,i+1} \oplus s_{1,i} \oplus s_{1,i+1}}_{=:\Delta s} \\ &= \Delta m \oplus \Delta k \oplus \Delta s \end{aligned}$$

Theorem 1.6.2:
With the example above follows

(a) $P(\Delta m = 0) > \frac{1}{2}$

(b) $P(\Delta s = 0) > \frac{1}{2}$

Proof:

(a) $P(\Delta m = 0) > \frac{1}{2}$ depends on subsequent message letters:
If adjacent letters in the message are the same, that ensures that $\Delta m = 0$ (repeated letter: 'wheels', 'letters', for German 0.61)

(b) $P(\Delta s = 0) > \frac{1}{2}$ follows by the structure of the machine:
When the s-wheel advance this probability is about $\frac{1}{2}$ but when they don't advance, Δs is always 0. This means, the probability that $\Delta s = 0$ is significanlty greater than $\frac{1}{2}$ (for the structure of the Lorenz Cipher Maschine it's about 0.73)

□

Example 11:
Assume $P(\Delta k = 0) = \frac{1}{2}$ and $P(\Delta m = 0) > \frac{1}{2}$ and $P(\Delta s = 0) > \frac{1}{2}$ with $\Delta z_{c,i} = \Delta m_{c,i} \oplus \Delta k_{c,i} \oplus \Delta s_{c,i}$. It's possible to break the cipher knowing more about the key k. If key is uniformly distributed, whatever patterns m and s have are lost when they get *xor'ed* with k in $\Delta z_{c,i} = \Delta m_{c,i} \oplus \Delta k_{c,i} \oplus \Delta s_{c,i}$. The k-wheels in the Lorenz Cipher machine produce key. Looking at Δz for two channels, i.e. only at the first two k-wheels (size 41 and size 31). Then there are $41 \cdot 31 = 1271$ different configurations for k_0 and k_1. That means that every 1271 letters those wheels woult repeat, and there are only 1271 different possible settings for the k-wheels. Trying all 1271 possible setting and for 1 of those possible settings we are goint to know all the key bits and if we guess the right setting then $\Delta k = 0$. If we guess right then $P(\Delta k = 0) = 1$ otherwise (false guess) $P(\Delta k = 0) = \frac{1}{2}$. With $\Delta z = \Delta m \oplus \Delta k \oplus \Delta s$ it follws $P(\Delta z = 0) = 0.55$ because:

$$\begin{aligned} \Delta z = \Delta m \oplus \underbrace{\Delta k}_{=0} \oplus \Delta s \Rightarrow P(\Delta z = 0) &= P(\Delta m = 0) \wedge P(\Delta s = 0) + P(\Delta m = 1) \wedge P(\Delta s = 1) \\ &= 0.61 \cdot 0.73 + (1 - 0.63) \cdot (1 - 0.73) \\ &= 0.55 \end{aligned}$$

Computing the sum of all Δz. If the output is nearly $\frac{|z|}{2}$ is was a bad guess otherwise if the result is about $0.55 \cdot |z|$ is was a good guess.
Assume having a 5000 letters message with all 1271 configurations of k_0 and k_1 and for all configuration its necessary to compute the summation of the Δz. Guessing that the $\Delta s = 0$ therefore

$$\Delta z_{0,i} \oplus \Delta z_{1,i} = m_{0,i} \oplus m_{0,i+1} \oplus m_{1,i} \oplus m_{1,i+1} \oplus k_{0,i} \oplus k_{0,i+1} \oplus k_{1,i} \oplus k_{1,i+1} \oplus \underbrace{s_{0,i} \oplus s_{0,i+1} \oplus s_{1,i} \oplus s_{1,i+1}}_{=:\Delta s=0}$$

Thus computin 7 *xors* for each character and counting the number of times that's equal to 0. It follows the number of *xors* are $5000 \cdot 1272 \cdot 7 = 44485000$ whats the maximum number of *xors* we have to do (expect about half of 44485000 operations to find the correct configuration of k_0 and k_1 and then do similar thinks with the other k-wheels and then we can decrypt the whole message). With a 2 GHz processor we need a fraction of a millisecond
Theorem 1.6.3:
Goal of cipher: Hide statistical properties of message and key (which should be perfectly random).

Goal of cryptoanalyst: find statistical properties in ciphertext and use those to break the key and/or message (Lorenz Cipher Machine has statistical properties when you looked acrosss channels at subsequent letters which was not hidden by the cipher and because of a mechanical weakness that all the s-wheel either all moved or didn't move and matehmatical weakness - only 1272 different positions of the first two k-wheels-

1.7 Modern Symmetric Ciphers

Definition *modern symmetric ciphers, stream ciphers, block ciphers*
There are two main types of *modern symmetric ciphers*:

- stream cipher:
  consists of a stream of data and the cipher can encrypt small chunks at a time (usually 1 byte at a time)

- block cipher:
  our data is separated in larger chunks and the cipher encrypts a block at a time (usually a block size is at least 64 bits and can be longer up to 128 or 256 bits)

The only differences is changing the block size. The different ciphers are designed for different purposes.
Definition *Advanced Encryption Standard (AES), Data Encryption Standard (DES)*
Advanced Encryption Standard or *AES* is the most important block cipher nowadays (since 1997) and works on blocks on 128 bits and displaced the *Data Encryption Standard* or *DES*, which had been a standard for the previous decades. *AES* was the winner of a competition that was run by the United States. The main criteria for the submissed ciphers where

- **security** (as provable security is only achievable for the One - Time Pad) computed with

$$\text{security} \sim \frac{\text{actual \# round}}{\text{minimal \# of rounds}}$$

  where breakable means anything that showed you could reduce the search space even a little bit woult be enough

- **speed**: implementing it both in hardware and in software and

- **simplicity** which is usually against security.

The winner of the *AES* competition was a cipher known as *Rijndael* (developed by two belgian cryptographers). A brute force attack with a 128 bit key would require on average $\frac{2^{128}}{2} = 2^{127}$ attempts. The best known attack needs 2^{126} attempts.
In *AES* works with *xor* and has two main operations

- **shift** (permuting bits - moving bits around)

- **s-boxes** (non-linearity: mixes up data in way that is not linear):
  This is done by lookup-tables:
  A *s-box* takes 8 bits and have a lookup table (with 256 entries) mapping each set of 8 bits to some other set of 8 bits. Designing the lookup table is a challenge. The lookup table has to be as nonlinear as possible and make sure there is no patterns in the data in this table.

The way *AES* works is combining *shifts* and *s-boxes* with *xor* to scramble up the data and do this multiple rounds and put them back through a series of *shifts* and *s-boxes* with *xor*. The number of rounds depens on the key size: for the smallest key size for *AES* (128 bits) we would do 10 rounds going through the cycle, getting the output cipher text for that block.