Chapter 5

Block ciphers

5.1 Introduction to block ciphers

In the previous chapter we have been introduced to stream ciphers. We now present the second class of symmetric ciphers, block ciphers.

A block cipher encrypts a block of plaintext bits x to a block of ciphertext bits y. The transformation is controlled by a secret key K, and it is written $E_K(x) = y$. In fact, each key defines a fixed mapping from the plaintext block to the ciphertext block. If the block size is b bits the number of different input values to the block ciphers is 2^b. This means that a block cipher can be viewed as a substitution cipher of alphabet size 2^b. However, only a very small fraction of all possible permutations on alphabet size 2^b will correspond to a certain key in the block cipher.

A block cipher needs an efficient implementation. Most block ciphers are *iterated ciphers*. In an iterated cipher we apply a simple encryption function iteratively a number of times (*rounds*), say N. The simple function applied is called the *round function*. It often uses a *round key*, which is derived from the key K. The way the round keys are derived from K is called the *key schedule*.

Let w_i, $i = 0..N$, denote intermediate values in the implementation process of the block cipher, where $w_0 = x$ is the plaintext block. Then the process of obtaining the ciphertext is described as

$$\begin{array}{rcc} w_0 & = & x \\ w_1 & = & h(w_0, K_1) \\ w_2 & = & h(w_1, K_2) \\ \vdots & & \\ w_{N-1} & = & h(w_{N-2}, K_{N-1}) \\ w_N & = & h(w_{N-1}, K_N) \end{array},$$

and $w_N = y$ is the ciphertext block. Here $h(w_{i-1}, K_i)$ denotes the round function and K_i is the round key used in round i.

A vital property for any encryption function $E_K(x)$ is that we need to have a procedure of efficient decryption $D_K(y)$. For an iterated cipher this property can be reduced to the

property of efficiently being able to invert the round function $h(w_{i-1}, K_i)$, ie., we need a function $h^{-1}()$ such that

$$h^{-1}(h(w_{i-1}, K_i), K_i) = w_{i-1}.$$

Then the decryption function $D_K(y)$ for an iterated cipher can be implemented in the same iterated style as the encryption with the difference that the round keys are used in reverse order,

$$\begin{array}{rcc} w_N & = & y \\ w_{N-1} & = & h^{-1}(w_N, K_N) \\ w_{N-2} & = & h^{-1}(w_{N-1}, K_{N-1}) \\ \vdots & & \\ w_1 & = & h^{-1}(w_2, K_2) \\ w_0 & = & h^{-1}(w_1, K_1) \end{array},$$

and $w_0 = x$.

5.2 Two different types of iterated ciphers

There are two common ways of building the round functions in a block cipher. The first one is called a *Feistel cipher*. In a Festel cipher we split the intermediate values w_i in a left half and a right half, denoted

$$w_i = (L_i, R_i).$$

Then the round function $(L_i, R_i) = h(L_{i-1}, R_{i-1}, K_i)$ is implemented as

$$\begin{array}{rcc} L_i & = & R_{i-1}, \\ R_i & = & L_{i-1} \oplus f(R_{i-1}, K_i), \end{array}$$

where $f(R_{i-1}, K_i)$ can be any function.

The motivation for choosing the round function in this way is the simplicity of the decryption function $h^{-1}()$, implemented as

$$\begin{array}{rcc} R_{i-1} & = & L_i, \\ L_{i-1} & = & R_i \oplus f(R_{i-1}, K_i), \end{array}$$

and (L_0, R_0) gives back x.

The second common way to construct the round function is to design it as an *SP network* (Substitution Permutation network). An SP network consists of a mix between many substitutions (substitution ciphers) and permutations or linear transformations (like a transposition cipher or affine cipher). The substitutions, also called S-boxes, are taken over a small alphabet, typically 4, 6, or 8 bits input. S-boxes of this size can be efficiently inverted, a requirement for efficient decryption. The mappings are often fixed and independent of the key in their description. Instead, the round key is added to the input to the round function in some simple way, usually bitwise xor. This makes the round function key dependent even if the S-boxes are not.

The permutation part permutes the different bits in the vector it is applied on.

5.3 DES - an example of a Feistel cipher

see project 4

5.4 AES - an example of an SP network

not included

5.5 Linear cryptanalysis

One of the most powerful general ideas for cryptanalysis of block ciphers is called *linear cryptanalysis*. The general idea is the following. If the cipher would consist only of linear operations, then there would be a linear relationship between the plaintext bits and the ciphertext bits. This means that for each ciphertext bit c_i we would be able to write one equation

$$c_i = \sum_{j=1}^{N} d_{ij} x_j + e_i(K), \tag{5.1}$$

where $e(K)$ denotes the direct contribution from the key K.

The problem is that the cipher contains nonlinear operations and the above is not true. The idea is to introduce linear approximations of nonlinear parts of the cipher. A linear approximation is when we replace a nonlinear function with a linear one. When we do so, we will not always get the correct value. The probability p that the linear function gives the same value as the original nonlinear one is a measure of the effectiveness of a linear approximation. If $p = 0.5$ the approximation is useless. But when $p \neq 0.5$ it can be useful.

Our goal in linear cryptanalysis is to find a linear expression through the whole cipher, i.e., a sum of expressions of the form (5.1) such that the probability p is as far away from 0.5 as possible. If such an expression has been established we can consider a known-ciphertext attack and write the key contribution $e(K)$ as a sum of known values,

$$e(K) = \sum_{j=1}^{N} d_j c_j \oplus \sum_{j=1}^{N} d'_j x_j.$$

Note that this expression holds with probability p. It is clear that if we evaluate the right hand side above for different plaintext-ciphertext pairs, we will eventually know the value of $e(K)$.

The hard problem in linear cryptanalysis is to find the best linear approximation and to prove its effectiveness. This is done by introducing many, but still as few as possible, linear approximations along a *trail* in the cipher. A trail corresponds to a chain of linear expressions (using intermediate values) through the cipher connecting the plaintext bits and the ciphertext bits.

For a detailed example, see project 4.

5.6 Modes of Operation

When we want to encrypt a sequence of plaintext bits, we need to decide how this should be done. The most obvious way would be to split the plaintext sequence

$$x_1, x_2, \ldots, x_N, x_{N+1}, \ldots$$

in blocks and encrypt each block. Let $X_1 = (x_1, x_2, \ldots, x_N)$ be the first block consisting of the first N bits, $X_2 = (x_{N+1}, x_{N+2}, \ldots, x_{2N})$ be the second block consisting of the next N bits, etc. Encryption is done blockwise, i.e.,

$$C_i = E_K(X_i), i = 1, 2, \ldots.$$

This way of encrypting is called the ECB (*Electronic codebook*) mode.

There is however a problem with the ECB mode. It is easily verified that if two plaintext blocks are the same, then the two corresponding ciphertext blocks are also the same.

There are instead other modes of operation that we can use. One common mode of operation is CBC mode (Cipher block chaining mode). It follows the rule

$$Y_i = E_K(Y_{i-1} \oplus X_i),$$

where Y_0 is some fixed and known initial value. This will produce the output sequence $Y_1, Y_2, \ldots$.

Another mode of operation that will turn the block cipher into a stream cipher is *counter mode*. The keystream sequence is given by

$$E_K(0), E_K(1), E_K(2), \ldots.$$