7 Lucifer

History and Relevance

Lucifer was the first published bitblock cipher. Horst Feistel at IBM developed it around 1970. It is in fact a Feistel cipher, and is a predecessor of the standard cipher DES that was developed shortly after. Compared with DES Lucifer seems stronger at first sight, but has some decisive weaknesses that became apparent in the meantime.

> Here we describe the variant that is called "Lucifer II" because of its publication date in 1975. First published (1973 in *Scientific American*) was a somewhat different variant called "Lucifer I".

These are the characteristics of Lucifer (compare the figures below):

- 128-bit key. This is a large enough key space even for today's requirements.
- 128-bit blocks. This is also considered sufficent down to the present day.
- For processing the 128-bit blocks (of keys and texts) are divided into 16 bytes. (From a historic point of we should say "octets" instead of "bytes" since in early computers a byte not necessarily consisted of exactly 8 bits.)
- The Feistel scheme consists of 16 rounds.
- In each round the 8 bytes of the right half R_i of a block (= 64 Bits) are processed quasi in parallel. In other words, every round processes 8 blocks à 1 byte, each one independently from the other ones.
- Each of the rounds consists of a substitution and a permutation. In between some key bits are added (XORed).
- Nonlinearity enters the algorithm only by the substitution. The newly added key bits are processed in a linear way in the actual round, but then are input to the nonlinear substitution of the next round.
- The substitution of a byte starts with a decomposition into two halves à 4 bit each of which is separately transformed by a substitution

  $$S_0, S_1 : \mathbb{F}_2^4 \longrightarrow \mathbb{F}_2^4.$$

  S_0 and S_1 are fixed substitutions used during the whole encryption process. Only the assignement of S_0 and S_1 to the 4-bit halves varies depending on a certain key bit. It is common use to call such nonlinear BOOLEan maps **"S-boxes"** when they occur as even more elementary building blocks of a kernel map.

The algorithm is very well suited for an implementation in hardware, in particular for 8-bit architectures. It is not so efficient in software for its many bit permutations.

Composing the kernel map of many small (identical or similar) S-boxes is a way followed often also today. A *rule of thumb*: the smaller the S-boxes, the more rounds are needed to achieve security.

The presentation of the algorithm follows the paper

- Arthur Sorkin: Lucifer, a cryptographic algorithm. Cryptologia 8 (1984), 22–41.

The Key Schedule

Denote the 16 bytes of the key $k \in \mathbb{F}_2^{128}$ by

$$k = (k_0, \ldots, k_{15}) \in (\mathbb{F}_2^8)^{16}$$

(IBM order but beginning with 0). Round i involves the selection

$$\alpha_i(k) = (\alpha_{ij}(k))_{0 \le j \le 7} \quad \text{of 8 bytes } \alpha_{ij}(k) = k_{7i+j-8 \bmod 16}.$$

This formula looks complicated but describes a quite simple selection scheme:

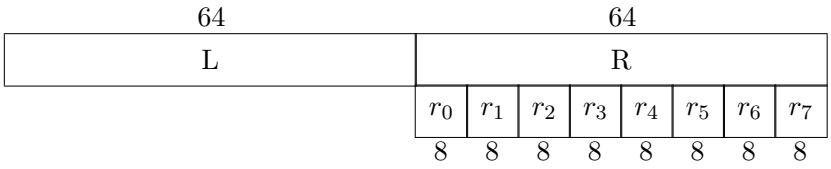
Round	Position							
	0	1	2	3	4	5	6	7
1	0	1	2	3	4	5	6	7
2	7	8	9	10	11	12	13	14
3	14	15	0	1	2	3	4	5
4	5	6	7	8	9	10	11	12
5	12	13	14	15	0	1	2	3
6	3	4	5	6	7	8	9	10
7	10	11	12	13	14	15	0	1
8	1	2	3	4	5	6	7	8
9	8	9	10	11	12	13	14	15
10	15	0	1	2	3	4	5	6
11	6	7	8	9	10	11	12	13
12	13	14	15	0	1	2	3	4
13	4	5	6	7	8	9	10	11
14	11	12	13	14	15	0	1	2
15	2	3	4	5	6	7	8	9
16	9	10	11	12	13	14	15	0

With each new round the selection is cyclically shifted by 7 positions. Note that each byte occurs at each position exactly once. The position defines to which byte of the actual 64-bit blocks the actual key byte applies. Furthermore the byte $\alpha_{i0}(k)$ in position 0 serves as “transformation control byte”.

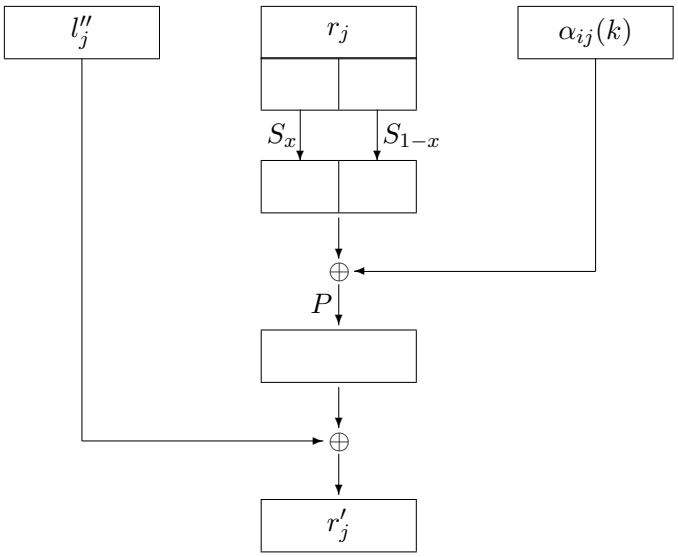
The Round Map

In round i the input, that is the right 64-bit part of the actual block, is divided into eight bytes

$$r = (r_0, \ldots, r_7).$$



Byte number j is transformed as follows by this round:



Here l''_j is a fixed selection of eight bits from the left part of the actual block. The transformation control byte

$$\alpha_{i1}(k) = (b_0, \ldots, b_7)$$

is taken from right to left, and $x = b_{7-j}$.

The definition of the kernel map f is clear from this picture. The explicit formula is not quite compact, therefore we omit it.

The S-Boxes

The S-boxes

$$S_0, S_1 \colon \mathbb{F}_2^4 \longrightarrow \mathbb{F}_2^4$$

are given by their value tables. Here the 4-bit blocks are written in hexadecimal notation, for example $1011 = B$.

$x =$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_0(x) =$	C	F	7	A	E	D	B	0	2	6	3	1	9	4	5	8

$x =$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_1(x) =$	7	2	E	9	3	B	0	4	C	D	1	A	6	F	8	5

The Permutations

The permutation P permutes the bits of a byte as follows:

$$\begin{aligned} P\colon \mathbb{F}_2^8 &\longrightarrow \mathbb{F}_2^8, \\ (z_0, z_1, z_2, z_3, z_4, z_5, z_6, z_7) &\mapsto (z_2, z_5, z_4, z_0, z_3, z_1, z_7, z_6). \end{aligned}$$

The bitwise addition of the left half to the transformed right half follows a permutation whose description is: Divide the left half of the actual block into eight bytes:

$$L = (l_0, l_1, l_2, l_3, l_4, l_5, l_6, l_7).$$

Rotate these cyclically after each step. Then for r_j they are in the positions

$$(l'_0, \ldots, l'_7) = (l_j, \ldots, l_{7+j \bmod 8}).$$

Finally construct the byte l''_j as

$$l''_j = (\text{Bit 0 of } l'_7, \text{Bit 1 of } l'_6, \text{Bit 2 of } l'_2, \ldots)$$

etc. in the order $(7, 6, 2, 1, 5, 0, 3, 4)$, and add it to r_j.