2.7 Lucifer

Geschichte und Bedeutung

Lucifer war die erste öffentlich bekannte und weit diskutierte Bitblock-Chiffre. Sie wurde um 1970 von Horst Feistel bei der IBM entwickelt und ist auch wirklich eine Feistel-Chiffre. Sie gilt als Vorläufer der kurz darauf entwickelten Standard-Chiffre DES, der gegenüber sie einige ins Auge fallende Stärken, aber, wie sich inzwischen herausgestellt hat, auch entscheidende Schwächen hat.

> Es handelt sich bei der Beschreibung hier um die Variante, die wegen der etwas späteren Veröffentlichung als „Lucifer II“ bezeichnet wird; die zuvor veröffentlichte Variante (1973 im *Scientific American*) weicht davon etwas ab.

Folgendes sind die Charakteristika von Lucifer:

- 128-Bit-Schlüssel, d. h., ein auch für heutige Begriffe bei weitem ausreichender Schlüsselraum.
- 128-Bit-Blöcke, auch dies – wie gesehen – heutigen Ansprüchen genügend.
- Für die Verarbeitung werden die 128 Bit von Schlüsseln und Blöcken in 16 Oktette (fälschlicherweise, aber inzwischen eingebürgert, auch als Bytes bezeichnet) unterteilt.
- Es werden 16 Runden durchgeführt.
- In jeder Runde werden die 8 Oktette der rechten Blockhälfte (= 64 Bits) quasi parallel bearbeitet – anders ausgedrückt, besteht jede Runde aus 8 gleichartigen Verarbeitungsblöcken, in der jeweils 1 Oktett abgearbeitet wird.
- Jeder dieser Verarbeitungsblöcke besteht aus einer Substitution und einer Permutation, dazwischen werden Schlüsselbits binär addiert.
- Nichtlinearität wird also einzig und allein durch die Substitution eingeführt.
- Für die Substitution eines Bytes wird dieses in zwei 4-Bit-Hälften zerlegt und jede davon getrennt mit einer Substitution

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

  behandelt. Das sind immer die gleichen beiden Substitutionen; einzig die Zuordnung – welche 4-Bit-Hälfte mit S_0 und welche mit S_1

behandelt wird – variiert; sie wird aufgrund eines Bits des Schlüssels entschieden. Es hat sich eingebürgert, solche elementaren nichtlinearen BOOLEschen Abbildungen wie diese S_0 und S_1 als **„S-Boxen“** zu bezeichnen.

- Das Verfahren ist sehr gut für eine Hardware-Implementierung geeignet – auch auf 8-Bit-Architekturen. In Software ist es wegen vieler Bit-Permutationen weniger effizient.
- Das Prinzip, die Kernfunktion aus vielen kleinen, evtl. gleichartigen, S-Boxen zusammenzusetzen, wird auch heute noch gelegentlich verwendet. *Faustregel*: Je kleiner die S-Boxen, desto mehr Runden sind nötig.

Die Darstellung des Verfahren folgt dem Artikel

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

Die Schlüsselauswahl

Die 16 Oktette des Schlüssels $k \in \mathbb{F}_2^{128}$ werden mit

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

bezeichnet – IBM-Nummerierung, aber mit 0 beginnend. In der i-ten Runde werden davon die Oktette

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

benützt. Diese kompliziert aussehende Formel beschreibt in Wirklichkeit etwas ganz einfaches, nämlich folgendes Auswahl-Schema:

Runde	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

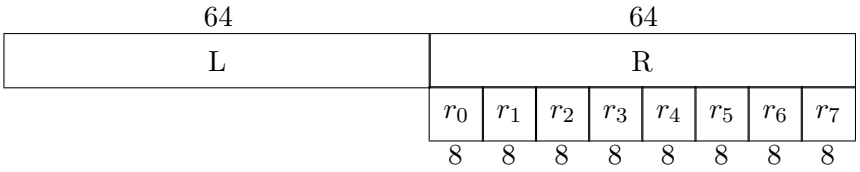
Von Runde zu Runde wird die Auswahl also zyklisch um 7 Plätze weitergeschoben. Bemerkenswert ist, dass jedes Oktett genau einmal in jeder Position vorkommt. Die Position gibt an, für welches Oktett des aktuellen 64-Bitblocks das jeweilige Schlüsseloktett verwendet wird. Das jeweils in Position 0 stehende Oktett $\alpha_{i0}(k)$ wird außerdem in der zugehörigen Runde als „Transformations-Steuerbyte“ eingesetzt.

Die Runden-Abbildung

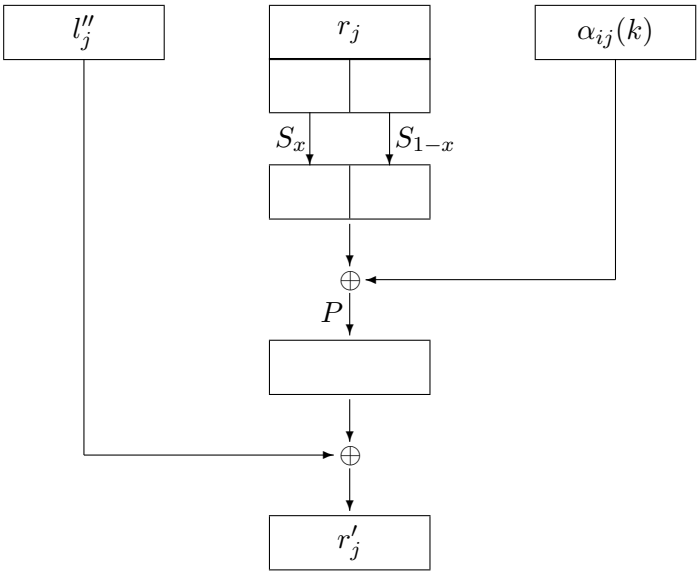
In der i-ten Runde sei der Input, die rechte 64-Bit-Hälfte des aktuellen Blocks, in die acht Oktette

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

eingeteilt:



Für das j-te davon sieht die Transformation in dieser Runde so aus:



Dabei ist l''_j eine feststehende Auswahl von acht Bits aus der linken Hälfte des aktuellen Blocks. Das Transformations-Steuerbyte

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

wird von rechts nach links abgearbeitet; es ist $x = b_{7-j}$.

Aufgrund dieser Beschreibung ist die eigentliche Kern-Abbildung f nicht kompakt explizit beschreibbar.

Die S-Boxen

Die S-Boxen

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

werden durch ihre Wertetabellen beschrieben. Dabei werden die 4-Bitblöcke hexadezimal notiert, also z. b. $1011 = B$.

$x =$	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

Die Permutationen

Die Permutation P vertauscht die Bits eines Oktetts wie folgt:

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

Die bitweise Addition der linken auf die transformierte rechte Hälfte geschieht nach einer Permutation, die so beschrieben wird: Die linke Hälfte des aktuellen Blocks besteht aus acht Oktetten:

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

diese werden nach jedem Schritt zyklisch rotiert, d. h., für r_j sind die in der Lage

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

Dann wird das aufzuaddierende Oktett l''_j so gebildet:

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

usw. in der Reihenfolge $(7, 6, 2, 1, 5, 0, 3, 4)$.