2 Bitblock-Chiffren und Feistel-Netze

In diesem Abschnitt werden grundlegende Überlegungen zu Bitblock-Chiffren sowie erste Konstruktions-Ansätze vorgestellt.

2.1 Bitblockchiffren – Einleitung

Beschreibung

Bitblock-Chiffren arbeiten über dem Alphabet $\Sigma = \mathbb{F}_2 = \{0, 1\}$ und verschlüsseln zunächst Blöcke fester Länge längentreu; sie sind also auf dem Vektorraum $\mathbb{F}_2^n$ definiert mit Schlüsselraum $\mathbb{F}_2^l$. Die Fortsetzung auf Bitketten beliebiger Länge ist Thema des Abschnitts „Betriebsarten“ und kümmert uns vorläufig nicht, ebensowenig wie die Frage, wie man zu kurze Blöcke auffüllt („Padding“).

Achtung: Der Buchstabe n bezeichnet jetzt bis auf weiteres nicht mehr die Größe des Alphabets, sondern die Länge der Bitblöcke.

Man kann eine Bitblock-Chiffre auch als monoalphabetisch über dem Alphabet $\Sigma' = \mathbb{F}_2^n$ ansehen. Zur Konstruktion und Beschreibung wird meist die algebraische Struktur als n-dimensionaler Vektorraum über dem Körper $\mathbb{F}_2$ verwendet, gelegentlich die als Körper $\mathbb{F}_{2^n}$, nur selten die als zyklische Gruppe der Ordnung 2^n.

Eine solche Chiffre wird also beschrieben durch eine Abbildung

$$F\colon \mathbb{F}_2^n \times \mathbb{F}_2^l \longrightarrow \mathbb{F}_2^n$$

bzw. als Familie $(F_k)_{k\in K}$ von Abbildungen

$$F_k\colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n \quad \text{für alle } k \in K = \mathbb{F}_2^l$$

mit $F_k(a) = F(a, k)$.

Wahl der Blocklänge

Die Blocklänge soll groß genug sein, um die Verfahren zum Brechen monoalphabetischer Substitutionen, insbesondere Muster- und Häufigkeitsanalysen, unmöglich zu machen; besser ist es, jede Art von Informationspreisgabe über den Klartext, z. B. jede Wiederholung im Geheimtext, zu vermeiden.

Bei einer **Codebuch-Attacke** würde der Angreifer bekannte Klartext-Geheimtextpaare bei festem Schlüssel sammeln, sich also sozusagen sein eigenes Codebuch anlegen. Ein vollständiges Codebuch führt, selbst wenn der Schlüssel sich daraus nicht bestimmen lässt, zu einer vollständigen Entschlüsselung. Konsequenzen:

- $\#\Sigma' = 2^n$ sollte größer als die Zahl der im ungünstigsten Fall verfügbaren Speicherplätze sein.
- Der Schlüssel sollte oft genug gewechselt werden.

Wegen des Geburtstagsphänomens sind allerdings strengere Kriterien sinnvoll: Falls der Gegner ca. $\sqrt{\#\Sigma'} = 2^{n/2}$ Klartext-Geheimtextpaare in seinem Codebuch gesammelt hat, ist die Wahrscheinlichkeit einer Kollision schon etwa $\frac{1}{2}$. Daher sollte diese Zahl, also $2^{n/2}$ bei einer Bitblock-Chiffre, schon die Zahl der verfügbaren Speicherplätze überschreiten; und auch Schlüssel sollten oft genug gewechselt werden – deutlich vor dieser Anzahl verschlüsselter Blöcke.

Die bisher meist verwendete Blocklänge 64 ist so gesehen also schon bedenklich; sie ist allenfalls noch bei häufigem Schlüsselwechsel zu rechtfertigen. Besser ist eine Blocklänge von 128 Bit, wie sie auch im neuen Standard AES vorgesehen ist.

Diese Überlegung ist ein typische Beispiel für die Sicherheitsabwägungen in der modernen Kryptographie: Es wird mit breiten Sicherheitsabständen gearbeitet; erkennbare Schwächen werden vermieden, selbst wenn sie noch weit von einer praktischen Auswertbarkeit für den Gegner entfernt sind. Da es effiziente Verfahren gibt, die diese Sicherheitsabstände einhalten, besteht überhaupt keine Notwendigkeit, weniger strenge Verfahren einzusetzen.

2.2 Polynome über endlichen Körpern

Satz 1 *Sei K ein endlicher Körper mit q Elementen und $n \in \mathbb{N}$. Dann wird jede Funktion $F : K^n \longrightarrow K$ durch ein Polynom $\varphi \in K[T_1, \ldots, T_n]$ vom partiellen Grad $\leq q-1$ in jedem T_i beschrieben.*

Beweis. (Skizze) folgt unten. $\diamond$

Korollar 1 *Seien $m, n \in \mathbb{N}$. Dann wird jede Abbildung $F : K^n \longrightarrow K^m$ durch ein m-Tupel $(\varphi_1, \ldots, \varphi_m)$ von Polynomen $\varphi_i \in K[T_1, \ldots, T_n]$ vom partiellen Grad $\leq q-1$ in jedem T_i beschrieben.*

Korollar 2 *Jede Abbildung $F : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m$ wird durch ein m-Tupel von Polynomen $(\varphi_1, \ldots, \varphi_m)$ von Polynomen $\varphi_i \in \mathbb{F}_2[T_1, \ldots, T_n]$ beschrieben, deren sämtliche partiellen Grade ≤ 1 sind.*

Damit erhalten wir auch die **algebraische Normalform** einer Boolschen Funktion $F : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2$: Für eine Teilmenge $I = \{i_1, \ldots, i_r\} \subseteq \{1, \ldots, n\}$ sei x^I das Monom

$$x^I = x_{i_1} \cdots x_{i_r}.$$

Dann lässt sich F eindeutig schreiben als

$$F(x_1, \ldots, x_n) = \prod_I a_I x^I \quad \text{mit } a_I = 0 \text{ oder } 1.$$

Insbesondere bilden die 2^n Monome x^I eine Basis des $\mathbb{F}_2$-Vektorraums $\mathrm{Abb}(\mathbb{F}_2^n, \mathbb{F}_2)$, und es gibt – wie auch anderweitig klar ist – 2^{2^n} solche Funktionen.

Beweisskizze für den Satz – ein etwas elementarerer, vollständiger Beweis im Fall $K = \mathbb{F}_2$ wird im Abschnitt „Die algebraische Normalform Boolescher Funktionen“ gegeben.

Sei zunächst K ein beliebiger (kommutativer) Körper. Dann ist $A := \mathrm{Abb}(K^n, K)$ eine K-Algebra. Sei $K[T]$ der Polynomring in dem n-Tupel $T = (T_1, \ldots, T_n)$ von Unbestimmten. Dann ist

$$\begin{array}{rccl} \alpha \colon K[T] & \longrightarrow & A, & \\ \varphi & \mapsto & \alpha(\varphi) & \text{mit } \alpha(\varphi)(x_1, \ldots, x_n) := \varphi(x_1, \ldots, x_n) \end{array}$$

ein K-Algebra-Homorphismus, der „Einsetzungs-Homomorphismus“. Sein Bild, $\mathrm{Bild}\,\alpha \subseteq A$, ist die Algebra der Polynomfunktionen. Es gibt zwei grundsätzlich unterschiedliche Fälle:

Fall 1: K ist unendlich. Dann ist α

- injektiv, d. h., unterschiedliche Polynome definieren unterschiedliche Funktionen – der Beweis ist der Eindeutigkeitsbeweis von Interpolationsformeln –,
- nicht surjektiv, denn $K[T]$ hat die gleiche Mächtigkeit wie K, dagegen hat A eine echt größere – der Beweis ist elementare Mengenlehre –.

Fall 2: K ist endlich. Dann ist α

- nicht injektiv, denn $K[T]$ ist unendlich, aber $\#A = q^{q^n}$,
- surjektiv, denn $F \in A$ wird vollständig beschrieben durch die q^n Paare $(x, F(x))$, $x \in K^n$, also durch den Graphen; mit Interpolation findet man ein Polynom $\varphi \in K[T]$ mit $\varphi(x) = F(x)$ für alle $x \in K^n$, d. h., $\alpha(\varphi) = F$.

Damit ist der erste Teil des Satzes bewiesen: *Jede Funktion $K^n \to K$ ist Polynomfunktion.*

Für den weiteren Beweisgang bestimmt man am besten $\operatorname{Kern}\alpha$. Sei $\mathfrak{a}$ das von den Polynomen $T_i^q - T_i$ erzeugte Ideal:

$$\mathfrak{a} = (T_1^q - T_1, \ldots, T_n^q - T_n) \trianglelefteq K[T].$$

Da die multiplikative Gruppe $K^\times$ die Ordnung $q-1$ hat, ist $a^q = a$ für alle $a \in K$. Also ist $\mathfrak{a} \subseteq \operatorname{Kern}\alpha$.

Nun hat der Restklassenring $K[T]/\mathfrak{a}$ offensichtlich ein vollständiges Repräsentantensystem aus den Restklassen derjenigen Polynome, die in allen T_i den Grad $\leq q-1$ haben, und die bilden einen K-Vektorraum der Dimension $\leq q^n$, da die Monome ihn aufspannen. Also ist

$$q^{q^n} = \#A = \#(K[T]/\operatorname{Kern}\alpha) \leq \#(K[T]/\mathfrak{a}) \leq q^{q^n}.$$

Da folglich überall in dieser Kette die Gleichheit gilt, ist $\operatorname{Kern}\alpha = \mathfrak{a}$, und $A \cong K[T]/\mathfrak{a}$ wird daher von den Polynomen mit allen partiellen Graden $\leq q-1$ ausgeschöpft. $\diamond$

2.3 Algebraische Kryptoanalyse

Der Angriff mit bekanntem Klartext

Sei (wie hier üblich) eine Bitblock-Chiffre durch eine Abbildung

$$F\colon \mathbb{F}_2^n \times \mathbb{F}_2^l \longrightarrow \mathbb{F}_2^n$$

beschrieben. Dann ist F ein n-Tupel $F = (F_1, \dots F_n)$ von Polynomfunktionen in $n + l$ Unbestimmten, deren sämtliche partiellen Grade ≤ 1 sind.

Ein Angriff mit bekanntem Klartext $a \in \mathbb{F}_2^n$ und Geheimtext $c \in \mathbb{F}_2^n$ ergibt ein Gleichungssystem

$$F(a, x) = c$$

von n Polynomgleichungen für den unbekannten Schlüssel $x \in \mathbb{F}_2^l$.

Solche Gleichungssysteme (über beliebigen Körpern) sind Gegenstand der Algebraischen Geometrie. Eine Faustregel besagt

Die Lösungsmenge für x ist „im allgemeinen klein“, wenn $n \geq l$.

(Andernfalls braucht man mehrere bekannte Klartextblöcke.)

Die allgemeine Theorie hierzu ist hochkompliziert, insbesondere, wenn man konkrete Lösungsverfahren haben will. Aber vielleicht hilft die Beobachtung, dass man nur partielle Grade ≤ 1 benötigt?

Beispiele

Beispiel 1, Linearität: Ist F eine *lineare* Abbildung, so ist das Gleichungssystem mit den Methoden der Linearen Algebra effizient lösbar (n lineare Gleichungen in l Unbekannten). Es reicht dazu schon, wenn F linear in x ist.

Beispiel 2: Sei $n = l = 2$, $F(T_1, T_2, X_1, X_2) = (T_1 + T_2X_1, T_2 + T_1X_2 + X_1X_2)$, $a = (0, 1)$, $c = (1, 1) \in \mathbb{F}_2^2$. Dann sieht das Gleichungssystem für den Schlüssel $(x_1, x_2) \in \mathbb{F}_2^2$ so aus:

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 + x_1 \\ 1 + 0 + x_1x_2 \end{pmatrix},$$

die Lösung ist offensichtlich $x_1 = 1$, $x_2 = 0$.

Substitution: Dass man Polynomgleichungen nicht immer auf den ersten Blick ihre Komplexität ansieht, zeigt das Beispiel (über $\mathbb{F}_2$)

$$x_1x_2x_3 + x_1x_2 + x_1x_3 + x_2x_3 + x_2 + x_3 = 0.$$

Es geht durch die Substitutionen $x_i = u_i + 1$ über in

$$u_1u_2u_3 + u_1 = 0$$

(umgekehrt sieht man das leichter) mit der Lösungsmenge

$$u_1 = 0,\ u_2, u_3 \text{ beliebig } \textit{oder } u_1 = u_2 = u_3 = 1.$$

Die vollständige Lösung der ursprünglichen Gleichung ist also

$$x_1 = 1,\ x_2, x_3 \text{ beliebig } \textit{oder } x_1 = x_2 = x_3 = 0.$$

Die Komplexität des Angriffs

Was in den Beispielen so einfach war, ist im allgemeinen zu komplex:

Satz 2 (Garey/Johnson) *Das Problem, eine gemeinsame Nullstelle eines Polynomsystems $f_1, \ldots, f_r \in \mathbb{F}_2[T_1, \ldots T_n]$ zu finden, ist* **NP***-vollständig.*

Beweis. Siehe das Buch von Garey/Johnson. ◇

Der Begriff „**NP**-vollständig“ wird später in der Vorlesung erklärt (siehe III.7).

Deutung: Bei günstig gewählter Blockverschlüsselungsfunktion $F\colon \mathbb{F}_2^n \times \mathbb{F}_2^l \longrightarrow \mathbb{F}_2^n$ ist der Angriff mit bekanntem Klartext nicht effizient durchführbar.

Offene Probleme

Im Grunde besagt der Satz für die praktische Anwendung *gar nichts*:

1. Er bezieht sich nur auf den Fall eines Algorithmus für *beliebige* Polynome. Er macht keine Aussage für ein bestimmtes Polynomsystem.
2. Selbst wenn er das machen würde, wäre immer noch kein konkretes Beispiel eines „schwierigen“ Polynomsystems bekannt.
3. Und selbst dann würde der Satz nichts darüber sagen, ob nur einzelne, wenige Instanzen des Problems schwierig sind oder – was der Kryptologe eigentlich braucht – fast alle.

Weitere Hinweise auf die Schwierigkeit, Polynomgleichungen zu lösen, gibt der Artikel

- D. Castro, M. Giusti, J. Heintz, G. Matera, L. M. Pardo: The hardness of polynomial equation solving. Found. Comput. Math. 3 (2003), 347–420. (Für Uni-Mainz online über die EZB zugänglich.)

Interpolationsangriff

Eine Variante der algebraischen Kryptoanalyse mit bekanntem Klartext ist der Interpolationsangriff, der in

- Thomas Jakobsen, Lars R. Knudsen: The interpolation attack on block ciphers, FSE 1997,

vorgestellt wurde. Die Idee ist ganz einfach: Der Vektorraum $\mathbb{F}_2^n$ kann bei Wahl einer geeigneten Multiplikation als endlicher Körper $K = \mathbb{F}_{2^n}$ der Charakteristik 2 interpretiert werden. Bei festem Schlüssel $k \in \mathbb{F}_2^l$ ist die Bitblock-Chiffre dann einfach eine Funktion $F_k \colon K \longrightarrow K$, also ein Polynom. Ist d sein Grad, so kann es nach der Interpolationsformel aus $d+1$ bekannten Klartext-Blöcken bestimmt werden; gleiches gilt für die Umkehrfunktion. Damit kann dann ver- bzw. entschlüsselt werden, ohne dass der Schlüssel explizit bestimmt wurde.

Um diesen Angriff zu verhindern, muss man also darauf achten, dass Ver- und Entschlüsselungsfunktion auf K bei festem Schlüssel stets einen hohen Grad besitzen; das ist machbar, da ja Grade bis $2^n - 1$ möglich sind.

Allerdings funktioniert der Angriff auch bei hohem Grad, wenn das jeweilige Polynom nur wenige Koeffizienten $\neq 0$ hat, also „dünn besetzt“ ist. Daher muss auch dieses vermieden werden.

Linearisierung überbestimmter Gleichungssysteme

Gleichungssysteme höherer Ordnung kann man manchmal brechen, wenn sie so weit überbestimmt sind, dass man die Monome (oder einige davon) als eigene Unbekannte ansehen kann. Dies wird durch das folgende einfache Beispiel illustriert:

$$\begin{aligned} x^3 + xy + y^5 &= 1, \\ 2x^3 - xy &= 0, \\ xy + 3y^5 &= 3. \end{aligned}$$

Hier substituiert man alle vorkommenden Monome: $u := x^3$, $v := xy$, $w := y^5$, und erhält das lineare Gleichungssystem

$$\begin{aligned} u + v + w &= 1, \\ 2u - v &= 0, \\ v + 3w &= 3. \end{aligned}$$

aus drei Gleichungen mit drei Unbekannten. Die (in diesem Fall sehr einfach auch manuell zu erhaltende) Lösung ist $u = 0$, $v = 0$, $w = 1$; sie ist eindeutig, wenn wir einen Körper der Charakteristik $\neq 7$ annehmen. Daraus ergibt sich als vollständige Lösung des ursprünglichen Systems: $x = 0$, $y = 1$ oder irgendeine im Körper enthaltene 5. Einheitswurzel.

Dieser Angriff wurde im Jahre 2002 populär, als das Gerücht um die Welt lief, der neue AES sei für diesen Angriff anfällig. Bei genauem Hinsehen ließ sich das aber nicht bestätigen; die einzelnen Gleichungen des konstruierten (riesigen) linearen Gleichungssystems waren bei weitem nicht unabhängig.

2.4 SP-Netze

Shannons Konstruktionsprinzipien

Nach Shannon sollen Blockchiffren folgendes leisten:

Diffusion (Durchmischung): Die Bits des Klartextblocks werden über den gesamten Block „verschmiert“. Quantitativ ausdrücken kann man das durch den „Lawinen-Effekt“ (englisch: avalanche effect):

- Jedes Bit des Geheimtextblocks hängt von jedem Bit des Klartextblocks ab.
- Bei Änderung eines Klartextbits ändern sich ca. 50% der Geheimtextbits.

Grundbausteine zur Erreichung von Diffusion sind Transpositionen.

Konfusion (Komplexität des Zusammenhangs): Die Beziehung zwischen Klartextblock und Geheimtextblock soll möglichst kompliziert sein (insbesondere hochgradig nichtlinear). Ähnlich komplex soll auch die Abhängigkeit des Geheimtextblocks vom Schlüssel sein, insbesondere sollen sich bei Änderung eines Schlüsselbits möglichst viele Geheimtextbits ändern, und zwar möglichst unvorhersagbar. Es soll für den Angreifer unmöglich sein zu erkennen, dass er einen Schlüssel „fast“ richtig geraten hat.

Grundbausteine hierfür sind vor allem Substitutionen.

Produktchiffren nach Shannon

Shannon schlug als Konstruktionsprinzip für starke Blockchiffren vor, Produktchiffren aus einer wechselnden Folge von **S**ubstitutionen und Transpositionen (= **P**ermutationen) zu bilden – sogenannte **SP-Netze**. Im einfachsten Fall sieht das so aus:

$$\begin{aligned} \mathbb{F}_2^n \xrightarrow{S_1(\bullet,k)} \mathbb{F}_2^n \xrightarrow{P_1(\bullet,k)} \mathbb{F}_2^n \longrightarrow \ldots \\ \ldots \longrightarrow \mathbb{F}_2^n \xrightarrow{S_r(\bullet,k)} \mathbb{F}_2^n \xrightarrow{P_r(\bullet,k)} \mathbb{F}_2^n \end{aligned}$$

abhängig von einem Schlüssel $k \in \mathbb{F}_2^l$. Dabei ist

$$\begin{aligned} S_i &= i\text{-te Substitution}, \\ P_i &= i\text{-te Permutation}, \\ P_i \circ S_i &= i\text{-te } \textbf{Runde} \text{ –} \end{aligned}$$

wobei insgesamt r Runden nacheinander ausgeführt werden.

Beispiel: Lucifer I (Feistel 1973).

2.5 FEISTEL-Chiffren

Die Kernabbildung

Die Blockgröße $n = 2s$ wird als gerade vorausgesetzt. Blöcke $a \in \mathbb{F}_2^n$ werden in ihre linke und rechte Hälfte zerlegt:

$$a = (L, R) \in \mathbb{F}_2^s \times \mathbb{F}_2^s$$

(groß geschrieben, um die Verwechslung mit der Dimension l des Schlüsselraums zu vermeiden). Hierfür muss man sich auf eine Nummerierung der Bits in einem Block einigen:

- In der **natürlichen Nummerierung**, die auch hier meistens verwendet wird, steht das LSB (Least Significant Bit) immer rechts und trägt die Nummer 0, das MSB (Most Significant Bit) steht links und trägt die Nummer $n - 1$:

  $$b = (b_{n-1}, \ldots, b_0) \in \mathbb{F}_2^n.$$

  Dies entspricht der Darstellung natürlicher Zahlen im ganzzahligen Intervall $[0 \ldots 2^n[$ zur Basis 2:

  $$b_{n-1} \cdot 2^{n-1} + \cdots b_1 \cdot 2 + b_0 \in \mathbb{N}.$$

- In der **IBM-Nummerierung** stehen die Bits umgekehrt und werden von 1 bis n nummeriert:

  $$a = (a_1, \ldots, a_n) \in \mathbb{F}_2^n.$$

  Dies entspricht der üblichen Nummerierung der Komponenten von Vektoren eines Vektorraums. Manchmal wird auch 0 bis $n - 1$ nummeriert.

Eine FEISTEL-Chiffre beruht auf einer **Kernabbildung**

$$f \colon \mathbb{F}_2^s \times \mathbb{F}_2^q \longrightarrow \mathbb{F}_2^s,$$

an die formal keine weiteren Anforderungen gestellt werden; insbesondere brauchen die $f(\bullet, k)$ nicht bijektiv zu sein.

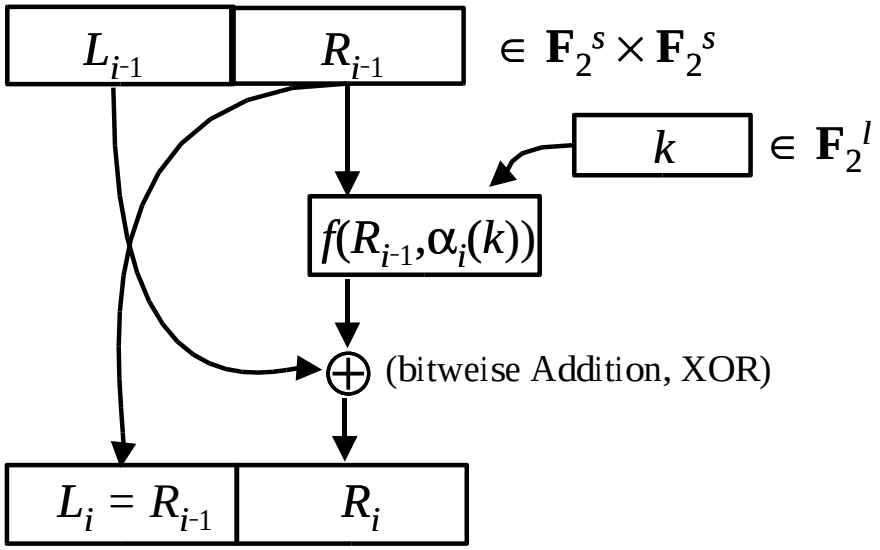
> Um ein brauchbares Verschlüsselungsverfahren zu erhalten, fordert man jedoch, dass f möglichst gute Konfusion und Diffusion bietet, etwa bereits aus Substitutionen und Transpositionen zusammengesetzt und hochgradig nichtlinear ist.

Beschreibung der Runden

Eine FEISTEL-Chiffre besteht aus r Runden, wobei jeweils aus dem Schlüssel $k \in \mathbb{F}_2^l$ ein q-Bit-Rundenschlüssel gebildet wird mit Hilfe der i-ten **Schlüsselauswahl**

$$\alpha_i \colon \mathbb{F}_2^l \longrightarrow \mathbb{F}_2^q \quad \text{für } i = 1, \ldots, r.$$

Die i-te Runde soll dann so aussehen:



Man erkennt in der Addition der linken Hälfte auf die transformierte rechte eine Spur des Autokey-Prinzips.

Algorithmische Beschreibung

Aus der graphischen Beschreibung leitet man leicht eine algorithmische ab:

$$\begin{array}{lrcl}
\textbf{Input} \longrightarrow & a & = & (a_0, a_1) \in \mathbb{F}_2^s \times \mathbb{F}_2^s \\
& a_2 & := & a_0 + f(a_1, \alpha_1(k)) \\
& & & \quad - \text{1. Runde, Ergebnis } (a_1, a_2) \\
& \vdots & & \vdots \\
& a_{i+1} & := & a_{i-1} + f(a_i, \alpha_i(k)) \\
& & & \quad - i\text{-te Runde, Ergebnis } (a_i, a_{i+1}) \\
& & & \quad - [a_i = R_{i-1} = L_i,\ a_{i+1} = R_i] \\
& \vdots & & \vdots \\
\textbf{Output} \longleftarrow & c & = & (a_r, a_{r+1}) =: F(a, k)
\end{array}$$

Die Entschlüsselung

Entschlüsselt wird nach der Formel

$$a_{i-1} = a_{i+1} + f(a_i, \alpha_i(k)) \quad \text{für } i = 1, \ldots, r.$$

Das entspricht dem gleichen Algorithmus, nur werden die Runden in umgekehrter Reihenfolge durchlaufen – mit anderen Worten: Die Schlüsselauswahl wird in umgekehrter Reihenfolge ausgeführt.

Insbesondere ist damit bewiesen:

Satz 3 (FEISTEL) *Sei* $F\colon \mathbb{F}_2^{2s} \times \mathbb{F}_2^l \longrightarrow \mathbb{F}_2^{2s}$ *die Blockabbildung zur Kernabbildung* $f : \mathbb{F}_2^s \times \mathbb{F}_2^q \longrightarrow \mathbb{F}_2^s$ *und zur Schlüsselauswahl* $\alpha = (\alpha_1, \ldots, \alpha_r)$, $\alpha_i\colon \mathbb{F}_2^l \longrightarrow \mathbb{F}_2^q$.

Dann ist die Verschlüsselungsfunktion $F(\bullet, k) : \mathbb{F}_2^{2s} \longrightarrow \mathbb{F}_2^{2s}$ *für jeden Schlüssel* $k \in \mathbb{F}_2^l$ *bijektiv.*

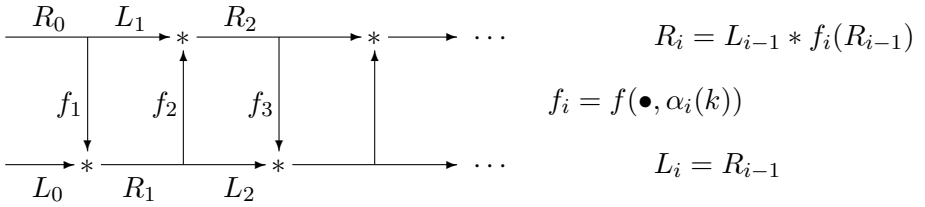
Zusatz. *Die Entschlüsselung ist die Blockabbildung zur gleichen Kernabbildung* f *und zur umgekehrten Schlüsselauswahl* $(\alpha_r, \ldots, \alpha_1)$.

Achtung: Beginnt man die Entschlüsselung mit $c = (a_r, a_{r+1})$, so sind zuerst *die Seiten zu vertauschen*, denn der Algorithmus beginnt mit (a_{r+1}, a_r). Daher wird bei der letzten Runde einer FEISTEL-Chiffre meist die Seitenvertauschung weggelassen.

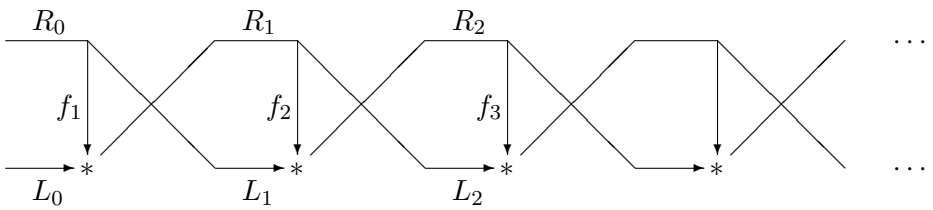
Anmerkungen

- Sind f und die α_i linear, so auch F.
- Für die α_i nimmt man meist nur eine Bitauswahl, also eine Projektion $\mathbb{F}_2^l \longrightarrow \mathbb{F}_2^q$.
- Alternative graphische Beschreibungen:

a) Leiter



b) Verdrehte Leiter



Verallgemeinerungen

1. Die Gruppe $(\mathbb{F}_2^s, \oplus)$ wird durch eine beliebige Gruppe $(G, *)$ ersetzt. Die Formeln für Verschlüsselung und Entschlüsselung werden dann zu:

$$\begin{aligned} a_{i+1} &= a_{i-1} * f(a_i, \alpha_i(k))), \\ a_{i-1} &= a_{i+1} * f(a_i, \alpha_i(k)))^{-1}. \end{aligned}$$

2. Unbalancierte Feistel-Chiffren (Schneier/Kelsey): Hier werden die Blöcke in zwei ungleich große Hälften zerteilt: $\mathbb{F}_2^n = \mathbb{F}_2^s \times \mathbb{F}_2^t$, $x = (\lambda(x), \rho(x))$. Die Formel für die Verschlüsselung wird dann zu:

$$\begin{aligned} L_i &= \rho(L_{i-1}, R_{i-1}) && \in \mathbb{F}_2^s, \\ R_i &= \lambda(L_{i-1}, R_{i-1}) \oplus f(L_i, \alpha_i(k))) && \in \mathbb{F}_2^t. \end{aligned}$$

Beispiele

1. Lucifer II (von Feistel 1971 entwickelt, 1975 veröffentlicht),
2. DES (von Coppersmith u. a. bei der IBM 1974 entwickelt, 1977 als US-Norm veröffentlicht),
3. u. v. a. neuere Bitblock-Chiffren.

Die **Bedeutung** der Feistel-Netze liegt in den empirischen Beobachtungen:

- Die „(s, q)-Bit-Sicherheit“ der Kernfunktion f wird durch die Mehrfach-Ausführung im Runden-Schema zu einer „(n, l)-Bit-Sicherheit“ der Feistel-Chiffre F erweitert.
- Die gesamte Chiffre ist lässt sich aus handhabbaren, auf Sicherheit optimierbaren Stücken zusammensetzen.

Die erste dieser Beobachtungen lässt sich auch theoretisch untermauern: Nach einem Ergebnis von Luby/Rackoff ist eine Feistel-Chiffre mit mindestens vier Runden nicht mehr effizient von einer zufälligen Permutation zu unterscheiden, wenn die Kernfunktion zufällig ist. D. h., eine an sich gute, aber zu kurze zufällige Funktion wird durch diese Konstruktion zu einer ausreichend langen erweitert.

2.6 Algebraische Angriffe bei kleiner Rundenzahl

Formeln für kleine Rundenzahlen

Die Rekursionsformel für eine FEISTEL-Chiffre kann man in der Form

$$(L_i, R_i) = (R_{i-1}, L_{i-1} + f(R_{i-1}, k_i))$$

schreiben mit dem Rundenschlüssel $k_i = \alpha_i(k)$.

Satz 4 *Die Ergebnisse einer* FEISTEL*-Chiffre nach* 2, 3 *und* 4 *Runden erfüllen die Gleichungen*

$$\begin{aligned} L_2 - L_0 &= f(R_0, k_1), \\ R_2 - R_0 &= f(L_2, k_2); \end{aligned}$$

$$\begin{aligned} L_3 - R_0 &= f(L_0 + f(R_0, k_1), k_2), \\ R_3 - L_0 &= f(L_3, k_3) + f(R_0, k_1); \end{aligned}$$

$$\begin{aligned} L_4 - L_0 &= f(R_0, k_1) + f(R_4 - f(L_4, k_4), k_3), \\ R_4 - R_0 &= f(L_4, k_4) + f(L_0 + f(R_0, k_1), k_2). \end{aligned}$$

Die Minuszeichen stehen hier, damit die Formeln auch noch für die Verallgemeinerung auf abelsche Gruppen gut sind, wo nicht, wie „im Binären“, Plus und Minus zusammenfallen. Der Sinn dieser Formeln ist, dass außer den Rundenschlüsseln k_i jeweils nur der Klartext (L_0, R_0) und der Geheimtext (L_r, R_r) vorkommen, also Größen, die bei der algebraischen Kryptoanalyse als bekannt angenommen werden.

Beweis. Im Fall von zwei Runden sind die Gleichungen

$$\begin{aligned} L_1 &= R_0, \\ R_1 &= L_0 + f(R_0, k_1), \\ L_2 &= R_1 = L_0 + f(R_0, k_1), \\ R_2 &= L_1 + f(R_1, k_2) = R_0 + f(L_2, k_2); \end{aligned}$$

und daraus folgt die Behauptung.

Im Fall von drei Runden ist analog

$$\begin{aligned} L_1 &= R_0, \\ R_1 &= L_0 + f(R_0, k_1), \\ L_2 &= R_1 = L_0 + f(R_0, k_1), \\ R_2 &= L_1 + f(R_1, k_2) = R_0 + f(L_2, k_2), \\ L_3 &= R_2 = R_0 + f(L_0 + f(R_0, k_1), k_2), \\ R_3 &= L_2 + f(R_2, k_3) = L_0 + f(R_0, k_1) + f(L_3, k_3). \end{aligned}$$

Die Berechnung für vier Runden bleibt dem Leser überlassen. ◇

Zweirunden-Chiffren

Bei einem Angriff mit einem bekannten Klartextblock sind L_0, R_0, L_2 und R_2 gegeben. Aufzulösen sind die Gleichungen

$$\begin{aligned} L_2 - L_0 &= f(R_0, k_1), \\ R_2 - R_0 &= f(L_2, k_2); \end{aligned}$$

nach k_1 und k_2. Die Sicherheit der Chiffre hängt also ganz von der Kernfunktion f ab. Da q, die Bitlänge der Teilschlüssel, allerdings in der Regel wesentlich kleiner als die Gesamtschlüssellänge l ist, sind die für einen Exhaustionsangriff nötigen 2^{q+1} Auswertungen von f eventuell im Bereich des Möglichen. Bemerkenswert ist, dass diese Überlegung unabhängig von der Schlüsselauswahl α ist – es werden einfach die tatsächlich verwendeten Schlüsselbits (k_1, k_2) bestimmt.

Beispiel: Auf $\mathbb{F}_2^s$ wird die Multiplikation „$\cdot$“ des Körpers $\mathbb{F}_t$ mit $t = 2^s$ verwendet [diese wird in einem späteren Abschnitt genauer erklärt] und

$$f(x, y) = x \cdot y$$

gesetzt. Die Schlüsselauswahl sei durch $l = 2q$ und $k_i =$ linke oder rechte Hälfte von k, je nachdem ob i ungerade oder gerade ist, gegeben. Dann werden die Gleichungen zu

$$\begin{aligned} L_2 - L_0 &= R_0 \cdot k_1, \\ R_2 - R_0 &= L_2 \cdot k_2, \end{aligned}$$

sind also leicht zu lösen. (Falls einer der Faktoren R_0 oder L_2 Null ist, braucht man natürlich einen anderen bekannten Klartextblock.)

Dreirunden-Chiffren

Hier sind die zu lösenden Gleichungen deutlich komplexer, da f bereits iteriert wird. Allerdings ist mit einem bekanntem Klartextblock ein Treffpunktangriff (Meet in the Middle) durchführbar, wenn die Bitlänge q der Teilschlüssel nicht zu groß ist: Man berechnet für alle möglichen Teilschlüssel k_1 das Zwischenergebnis (L_1, R_1) nach der ersten Runde und speichert es in einer Tabelle. Über die letzten beiden Runden macht man eine Exhaustion, wie für die Zweirunden-Chiffre beschrieben. Der Gesamtaufwand beträgt also $3 \cdot 2^q$ Auswertungen von f und 2^q Speicherplätze.

Daraus resultiert die Faustregel: Feistel-*Chiffren sollten stets mindestens vier Runden haben.* Das ist eine passende Ergänzung zu dem zitierten Ergebnis von Luby/Rackoff. Man sieht, wie mit wachsender Rundenzahl, wenn nur f genügend komplex ist, die Resistenz des gesamten Schemas gegen den algebraischen Angriff wächst.

In der **Beispiel**chiffre mit Kernfunktion = Multiplikation im Körper mit 2^s Elementen werden die Gleichungen zu:

$$\begin{aligned} L_3 - R_0 &= [L_0 + R_0 \cdot k_1)] \cdot k_2, \\ R_3 - L_0 &= [R_0 + R_3] \cdot k_1. \end{aligned}$$

Sie sind offensichtlich leicht zu lösen.

Vierrunden-Chiffren

Hier sind die Gleichungen noch komplexer, und selbst im **Beispiel**

$$\begin{aligned} L_4 - L_0 &= [R_0 + R_4 + L_4 \cdot k_2] \cdot k_1, \\ R_4 - R_0 &= [L_4 + L_0 + R_0 \cdot k_1] \cdot k_2, \end{aligned}$$

sind sie schon quadratisch in zwei Unbekannten (wenn auch in diesem Trivialbeispiel noch leicht zu lösen – die Elimination von k_1 führt auf eine quadratische Gleichung für k_2 – **Übungsaufgabe**).

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 denen 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 in den Algorithmus eingeführt.
- Für die Substitution eines Bytes wird dieses in zwei 4-Bit-Hälften zerlegt und jede davon getrennt mit einer Substitution

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

  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, um Sicherheit zu erreichen.

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

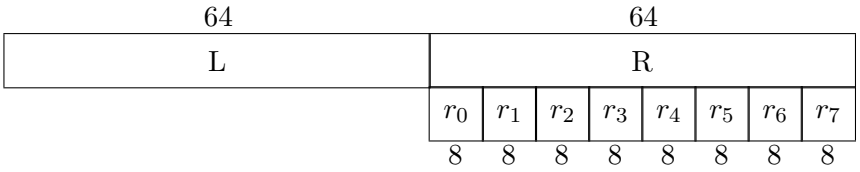
Von Runde zu Runde wird die Auswahl also zyklisch um 7 Plätze weitergeschoben. Bemerkenswert ist, dass jedes Oktett genau einmal in jeder Position vorkommt. Die Position gibt an, für welches Oktett des aktuellen 64-Bitblocks das jeweilige Schlüsseloktett verwendet wird. Das jeweils in Position 0 stehende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

Die Runden-Abbildung

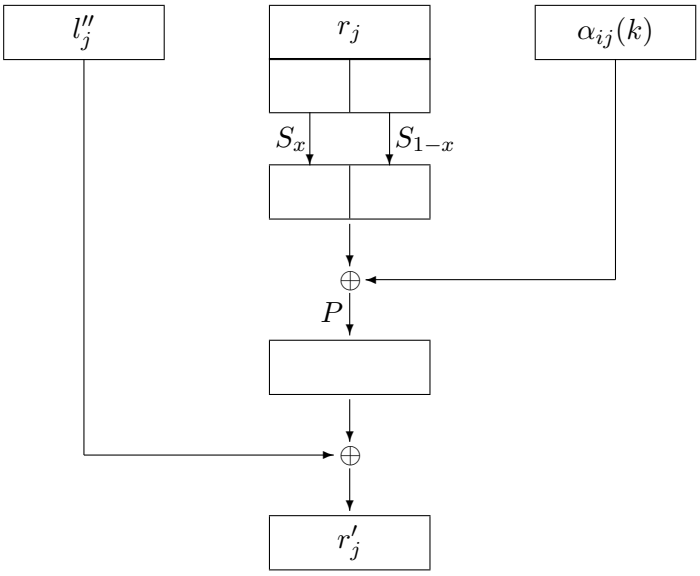
In der i-ten Runde wird der Input, die rechte 64-Bit-Hälfte des aktuellen Blocks, in die acht Oktette

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

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 $x = b_{7-j}$.

Diese grafische Beschreibung der eigentlichen Kern-Abbildung f ist nicht durch eine kompakte explizite Formel wiederzugeben.

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 $(7, 6, 2, 1, 5, 0, 3, 4)$.