# 2 Frische symmetrische Verschlüsselung und Blockchiffren

**Szenarium 2** (frische symmetrische Verschlüsselung): Alice möchte Bob *mehrere verschiedene* Klartexte *vorher bekannter und begrenzter* Länge übermitteln. Sie verwendet dafür immer denselben Schlüssel. Eva hört die Chiffretexte mit und kann sich sogar einige Klartexte mit dem verwendeten Schlüssel verschlüsseln lassen (*chosen-plaintext attack, CPA*).

**Bemerkung** Das informationstheoretisch sichere Vernam-Kryptosystem aus Kapitel 1 ist nutzlos: Aus Kenntnis von $x \in \{0,1\}^\ell$ und $y = e(x,k)$ für ein einziges Paar $(x,k) \in X \times K$ kann Eva den Schlüssel $k = x \oplus_\ell y$ berechnen.

Gleiches gilt für das Cäsar-System und das Vigenère-System.

Mit dem nächsten Begriff erfassen wir folgende Situation: Eva kennt eine ganze Folge von Klartext-Chiffretext-Paaren bezüglich des (ihr unbekannten) Schlüssels $k$. Dabei kann sie sich die Klartexte sogar selbst herausgesucht haben. Wir wollen „possibilistische Sicherheit" so definieren, dass sie trotzdem bei beliebigem gegebenem weiteren Chiffretext $y$ keinen Klartext ausschließen kann.

**Definition 2.1** *Ein Kryptosystem $\mathcal{S} = (X, K, Y, e, d)$ ist* possibilistisch sicher bzgl. Szenarium 2*, wenn für jedes $1 \le r \le |X|$, jede Folge von paarweise verschiedenen Klartexten $x_1, x_2, \ldots, x_r \in X$, jeden Schlüssel $k \in K$ und jedes $y \in Y \setminus \{e(x_i,k) \mid 1 \le i < r\}$ ein Schlüssel $k' \in K$ existiert mit $e(x_i,k) = e(x_i,k')$ für alle $1 \le i < r$ und $e(x_r,k') = y$.*

Wenn man die Definition auf $r = 1$ anwendet, ergibt sich, dass $\mathcal{S}$ auch possibilistisch sicher im Sinn von Kapitel 1 ist.

**Proposition 2.2** *Für jede nichtleere Menge $X$ ist das Substitutionskryptosystem (Def. 1.9) auf $X$ possibilistisch sicher.*
(Erinnerung: $K = \mathcal{P}_X = \{\pi \mid \pi \text{ ist Permutation von } X\}$ und $e(\pi, x) = \pi(x)$.)

*Beweis*: Seien $x_1, x_2, \ldots, x_r \in X$ paarweise verschieden, $k \in K = \mathcal{P}_X$ und $y \in Y \setminus \{e(x_i,k) \mid 1 \le i < r\}$. Aufgrund der Dechiffrierbedingung sind die Chiffretexte $e(x_i,k)$ für $1 \le i < r$ paarweise verschieden. Also gilt

$$|Y \setminus (\{y\} \cup \{e(x_i,k) \mid 1 \le i < r\})| = |Y| - r = |X| - r = |X \setminus \{x_1, \ldots, x_r\}|,$$

und es existiert eine Bijektion $f\colon X \setminus \{x_1, \ldots, x_r\} \to Y \setminus (\{y\} \cup \{e(x_i,k) \mid 1 \le i < r\})$.

Definiere nun $\pi\colon X \to Y$ durch

$$\pi(x) = \begin{cases} e(x_i, k) & \text{falls } \exists i \in \{1, \ldots, r-1\} : x = x_i \\ y & \text{falls } x = x_r \\ f(x) & \text{sonst} \end{cases}$$

Dann ist $\pi$ eine Bijektion und damit eine Permutation von $X$ (da $X = Y$). □

**Proposition 2.3** *Sei $\mathcal{S} = (X, K, Y, e, d)$ ein Kryptosystem, das possibilistisch sicher ist bzgl. Szenarium 2. Dann ist $\{e(\,.\,, k) \mid k \in K\}$ die Menge aller injektiven Abbildungen von $X$ nach $Y$.*

*Beweis*: Aus der Dechiffrierbedingung folgt sofort, dass alle Abbildungen $e(\,.\,, k)$ injektiv sind.

Sei nun $\pi\colon X \to Y$ eine beliebige injektive Abbildung. Weiter gelte $X = \{x_1, \ldots, x_t\}$. Wir konstruieren induktiv Schlüssel $k_1, \ldots, k_t$ mit

$$\forall i \in \{1, \ldots, r\}\colon e(x_i, k_r) = \pi(x_i)\,.$$

Das heißt: Schlüssel $k_r$ realisiert $\pi$ eingeschränkt auf $\{x_1, \ldots, x_r\}$.

Da $\mathcal{S}$ auch possibilistisch sicher bzgl. Szenarium 1 ist, existiert der Schlüssel $k_1$ wie gewünscht. Die Schlüssel $k_r$ für $r = 2, \ldots, t$ ergeben sich nacheinander, indem man wiederholt Definition 2.1 auf $\{x_1, \ldots, x_r\}$, $k_{r-1}$ und $y = \pi(x_r)$ anwendet. Schließlich gilt $e(\,.\,, k_t) = \pi$. □

Im Fall $X = Y$ folgt, dass *nur* das Substitutionskryptosystem possibilistisch sicher bzgl. Szenarium 2 ist.

Ein Kryptosystem, das possibilistisch sicher bzgl. Szenarium 2 ist, hat also mindestens

$$|\{\pi \mid \pi\colon X \to Y \text{ ist injektiv}\}| = \frac{|Y|!}{(|Y| - |X|)!} \geq |X|!$$

Schlüssel. Mit $X = \{0,1\}^{128}$ (das ist durchaus realistisch!) gibt es also $\geq 2^{128}!$ viele Schlüssel. Für die Speicherung *eines* Schlüssels braucht man also durchschnittlich[19]

$$\log(2^{128}!) > 2^{128} \cdot (128 - 1.45) > 2^{134} > (10^3)^{13.4} > 10^{40}$$

viele Bits, eine völlig unpraktikable Zahl.

[19] Es gilt $\log_2(N!) > N(\log_2 N - \log_2 e) > N(\log_2 N - 1.45)$. Siehe dazu etwa Vorlesung „Algorithmen und Datenstrukturen“, Kapitel 6, Untere Schranke für Sortieren.

Also können wir in realen Situationen im Szenarium 2 keine possibilistische Sicherheit, geschweige denn informationstheoretische Sicherheit[20] erhalten.

Wir werden uns in diesem Kapitel also darauf verlassen müssen, dass Eva nur *beschränkte Rechenressourcen* hat, um einen praktikablen Angriff zu starten. Auf der anderen Seite wollen wir jetzt auch annehmen, dass Alice und Bob mit Computerhilfe verschlüsseln und entschlüsseln wollen, d. h. dass wir effiziente Algorithmen für die Ver- und Entschlüsselung benötigen. Damit wird es sinnvoll anzunehmen, dass Klar- und Chiffretexte und Schlüssel Bitvektoren sind.

**Definition 2.4** *Sei $\ell > 0$. Ein* $\ell$-Block-Kryptosystem *ist ein Kryptosystem $\mathcal{S} = (\{0,1\}^\ell, K, \{0,1\}^\ell, e, d)$ mit $K \subseteq \{0,1\}^s$ für ein $s > 0$.*

*Beispiele* für $\ell$-Block-Kryptosysteme sind das Vernam-System der Länge $\ell$ und das Substitutionskryptosystem mit $X = \{0,1\}^\ell$, falls Permutationen als Bitvektoren kodiert sind. Wir sprechen dann vom *Substitutionskryptosystem mit Parameter $\ell$*.

Eine weitere Klasse von Verfahren wird im nächsten Abschnitt vorgestellt.

## 2.1 Substitutions-Permutations-Kryptosysteme (SPKS)

Substitutions-Permutations-Kryptosysteme bilden eine große Familie von praktisch relevanten Kryptosystemen, zu denen auch die Verfahren des Data Encryption Standard (DES) und des Advanced Encryption Standard (AES) gehören.

Grundsätzlich handelt es sich um $\ell$-Block-Kryptosysteme für $\ell = mn$. Dabei werden die Klartexte $x = (x_0, \ldots, x_{mn-1}) \in \{0,1\}^{mn}$ als $m$-Tupel $(x^{(0)}, x^{(1)}, \ldots, x^{(m-1)})$ von Bitvektoren der Länge $n$ betrachtet, wobei gilt $x^{(i)} = (x_{in}, x_{in+1}, \ldots, x_{(i+1)n-1})$, für $0 \leq i < m$.

Ein Baustein der Verschlüsselungs- und Entschlüsselungsfunktion bei diesen Systemen sind *Bitpermutationen* auf Bitstrings der Länge $\ell$: Sei $\beta$ eine Permutation von $\{0, \ldots, \ell - 1\}$ und $x \in \{0,1\}^\ell$. Dann bezeichnet $x^\beta$ den Bitvektor

$$(x_{\beta(0)}, x_{\beta(1)}, \ldots, x_{\beta(\ell-1)}) .$$

Kurz: $x^\beta(i) = x_{\beta(i)}$, für $0 \leq i < \ell$.

Sehr oft werden wir annehmen, dass $\beta$ *selbstinvers* ist, dass also $\beta^{-1} = \beta$ ist (oder $\beta(\beta(i)) = i$ für $0 \leq i < \ell$). Dann erhält man $x^\beta$ aus $x$, indem man $x(i)$ an die Stelle $\beta(i)$ des neuen Vektors schreibt.

[20] Dieses Konzept lässt sich leicht analog zur informationstheoretischen Sicherheit in Szenarium 1 definieren. Man muss nur die wahrscheinlichkeitstheoretische Aussage auf jede beliebige gegebene Zuordnung von $x_1, \ldots, x_{k-1}$ auf Chiffretexte $y_1, \ldots, y_{k-1}$ bedingen.

**Definition 2.5** *Ein* Substitutions-Permutations-Netzwerk (SPN) *ist ein Tupel*

$$N = (m, n, r, s, S, \beta, \kappa)$$

*wobei*

- *die positiven ganzen Zahlen* $m$*,* $n$*,* $r$ *und* $s$ *die* Wortanzahl $m$*, die* Wortlänge $n$*, die* Rundenzahl $r$ *und die* Schlüssellänge $s$ *angeben,*
- $S\colon \{0,1\}^n \to \{0,1\}^n$ *eine bijektive Funktion ist („Substitution“, die* S-Box*),*
- $\beta\colon \{0,\dots,mn-1\} \to \{0,\dots,mn-1\}$ *eine* selbstinverse *Permutation (die* Bitpermutation*) ist und*
- $\kappa\colon \{0,1\}^s \times \{0,\dots,r\} \to \{0,1\}^{mn}$ *die* Rundenschlüsselfunktion[21] *ist.*

*Das zu* $N$ *gehörende* Substitutions-Permutations-Kryptosystem (SPKS) *ist das* $mn$*-Block-Kryptosystem*

$$\mathcal{B}(N) = (\{0,1\}^{mn}, \{0,1\}^s, \{0,1\}^{mn}, e, d)\,,$$

*das durch folgende Operationen beschrieben ist:*

Chiffrierung: $e(x,k)$ wird für $x \in \{0,1\}^{mn}$ und $k \in \{0,1\}^s$ wie folgt berechnet:

1. Initialisierung („Weißschritt“): Berechne $u = x \oplus_{mn} \kappa(k,0)$.
2. Verschlüsselung in Runden: für $i = 1,\dots,r-1$ berechne
   (a) $v^{(j)} = S(u^{(j)})$ für $0 \le j < m$ (jedes Wort einzeln durch die $S$-Box)
   (b) $w = v^\beta$ (Bitpermutation auf dem Gesamtwort der Länge $mn$)
   (c) $u = w \oplus_{mn} \kappa(k,i)$ (XOR mit dem Rundenschlüssel)
3. Schlussrunde: $v^{(j)} = S(u^{(j)})$ für $0 \le j < m$
   Ausgabe: $y = v \oplus \kappa(k,r)$ (wie gewöhnliche Runde, ohne Bitpermutation)

Dechiffrierung: $d(y,k)$ wird aus $y$ und $k$ *nach demselben Verfahren* berechnet, wobei jedoch

1. die $S$-Box (die eine Permutation ist) durch ihre Inverse $S^{-1}$ ersetzt wird und
2. die Rundenschlüsselfunktion $\kappa$ ersetzt wird durch die Funktion

$$\kappa' : \{0,1\}^s \times \{0,\dots,r\} \to \{0,1\}^{mn}, (k,i) \mapsto \begin{cases} \kappa(k, r-i) & \text{für } i \in \{0,r\} \\ \kappa(k,r-i)^\beta & \text{für } 0 < i < r \end{cases}$$

[21] $\kappa$ berechnet aus dem eigentlichen Schlüssel $k \in K$ und der Rundennummer $i$ einen *Rundenschlüssel* $\kappa(k,i) \in \{0,1\}^{mn}$.

**[Blockschaltbild aus Buch v. Küsters/Wilke, Seite 50. Handout in Vorlesung.]**

Vorteil der Struktur: Man kann dieselbe Hardware für Verschlüsselung und Entschlüsselung benutzen. Bei der Entschlüsselung läuft der Vorgang rückwärts ab, wobei die Runden anders gruppiert sind. Anstelle von $S$ verwendet man $S^{-1}$. Da $\beta$ selbstinvers ist, kann man für die Permutation auch bei der Dekodierung $\beta$ verwenden. Da Permutation und Schlüsselanwendung in den Runden vertauscht sind, muss man auf die Rundenschlüssel der „inneren" Runden bei der Dekodierung auch noch $\beta$ anwenden.

Ein typischer Vertreter dieser Art Kryptosystem ist DES[22] (bzw. sind die Chiffren der DES-Familie). Die DES-Version mit Blocklänge 64 und effektiver Schlüsselbreite 56 (das heißt, dass der Schlüsselraum Größe $2^{56}$ hat) wurde im 1977 als Standard für nichtgeheime Dokumente in USA publiziert. „Lineare Kryptanalyse" ist eine Methode, solche Verfahren zu attackieren. Nach Weiterentwicklung der Methode gelang 1999 die Entschlüsselung einer DES-verschlüsselten Nachricht in weniger als einem Tag, mit einem Verbund von 100 000 Arbeitsplatzrechnern, die im Wesentlichen alle Schlüssel durchprobierten. Dabei wurden über 240 Milliarden Schlüssel pro Sekunde getestet. Stärkere Varianten wie Triple-DES werden auch heute noch eingesetzt.

**Beispiel 2.6** Die Wortanzahl ist $m = 3$, die Wortlänge $n = 4$, die Rundenzahl $r = 3$, die Schlüssellänge $s = 24 = 6 \cdot n$, und $\kappa(k, i) = k^{(i)}k^{(i+1)}k^{(i+2)}$. Die Bitpermutation $\beta$ vertauscht die Bits $(0, 4)$, $(1, 5)$, $(2, 8)$, $(3, 9)$, $(6, 10)$ und $(7, 11)$ und ist damit selbstinvers. Die S-Box ist gegeben durch die folgende zweizeilige Tabelle:

| $b$ | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 |
|---|---|---|---|---|---|---|---|---|
| $S(b)$ | 0101 | 0100 | 1101 | 0001 | 0011 | 1100 | 1011 | 1000 |
| $b$ | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
| $S(b)$ | 1010 | 0010 | 0110 | 1111 | 1001 | 1110 | 0000 | 0111 |

Für $x = 0000\,1111\,0000$ und $k = 0000\,0001\,0010\,0011\,0100\,0101$ ergeben sich nacheinander

1. $\kappa(k, 0) = 0000\,0001\,0010$ und damit $u = 0000\,1110\,0010$
2. $i = 1$ $v = 0101\,0000\,1101$, $w = 0011\,0101\,0100$, $\kappa(k, 1) = 0001\,0010\,0011$ und $u = 0010\,0111\,0111$

   $i = 2$ $v = 1101\,1000\,1000$, $w = 1010\,1100\,0100$, $\kappa(k, 2) = 0010\,0011\,0100$ und damit $u = 1000\,1111\,0000$
3. $v = 1010\,0111\,0101$, $\kappa(k, 3) = 0011\,0100\,0101$ und damit $y = 1001\,0011\,0000$

[22] DES ist Abkürzung für „Data Encryption Standard".

## Einschub: Endliche Körper

Zur Vorbereitung der Beschreibung von AES und für die spätere Verwendung in asymmetrischen Systemen diskutieren wir kurz die Konstruktion von endlichen Körpern mit $2^k$ Elementen. Dies ist eine Spezialisierung einer allgemeinen Konstruktion, die es erlaubt, aus einem Körper mit $q$ Elementen einen mit $q^k$ Elementen zu konstruieren. Eine detaillierte Darstellung der Konstruktion findet man z. B. im Buch „Diskrete Strukturen 1. Kombinatorik, Graphentheorie, Algebra“ von Angelika Steger (2. Auflage Springer, 2007).

Die Struktur $\mathbb{Z}_2 = (\{0,1\}, \oplus, \odot, 0, 1)$ ist ein Körper, wobei $\oplus$ für die XOR-Operation und $\odot$ für $\wedge$ (AND) steht. (Körper: Addition, Multiplikation mit den üblichen Gesetzen, d.h. $(\mathbb{F}, \oplus, 0)$ ist kommutative Gruppe, $\odot$ ist assoziativ und hat 1 als neutrales Element, die Distributivgesetze gelten, und Elemente $\neq 0$ haben ein multiplikatives Inverses.)[23] Man beachte, dass in $\mathbb{Z}_2$ die Subtraktion dasselbe ist wie die Addition: $a \oplus a = 0$ für $a = 0, 1$.

Zu einem endlichen Körper $\mathbb{F}$ (zentrales Beispiel: $\mathbb{Z}_2$) bildet man den *Polynomring* $\mathbb{F}[X]$. Vorstellen sollte man sich diesen als die Menge aller „formalen Ausdrücke“

$$a_n \cdot X^n + \cdots + a_2 \cdot X^2 + a_1 \cdot X + a_0,$$

für $n \geq 0$ und $a_n, \ldots, a_1, a_0 \in \mathbb{F}$. Ein solcher Ausdruck wird mit seiner Koeffizientenfolge $(\ldots, 0, 0, a_n, \ldots, a_1, a_0)$ (gegebenenfalls künstlich als unendliche Folge geschrieben) identifiziert. Normalerweise lässt man Terme mit Koeffizient 0 weg. Über $\mathbb{Z}_2$ ist es üblich, nur den Bitstring $a_n \ldots a_1 a_0$ ohne Klammern und Kommas zu schreiben.

*Beispiel*: In $\mathbb{Z}_2[X]$ liegen die Polynome $g = 01011 = (\ldots, 0, 1, 0, 1, 1) = X^3 + X + 1$ und $h = 0110 = (\ldots, 0, 1, 1, 0) = X^2 + X$.

Der Grad eines solchen Polynoms ist $n$, wenn $n$ maximal mit $a_n \neq 0$ ist. Falls alle Koeffizienten gleich 0 sind ($f$ ist das „Nullpolynom“, man schreibt 0), ist der Grad $-\infty$. Man addiert und subtrahiert solche Polynome wie üblich (d. h. komponentenweise) und multipliziert sie wie üblich (ausmultiplizieren, Koeffizienten bei derselben $X$-Potenz aufsammeln). Als geschlossene Formel: Das Produkt von $(\ldots, 0, 0, a_n, \ldots, a_1, a_0)$ und $(\ldots, 0, 0, b_m, \ldots, b_1, b_0)$ ist $(\ldots, 0, 0, c_{n+m}, \ldots, c_1, c_0)$ mit

$$c_k = \sum_{\substack{0 \leq i \leq n \\ 0 \leq j \leq m \\ i+j=k}} a_i \cdot b_j.$$

*Beispiel*: Die Summe von $g$ und $h$ ist $g + h = 01101 = (\ldots, 0, 1, 1, 0, 1)$, ihr Produkt ist $g \cdot h = X^5 + X^4 + X^3 + X = (\ldots, 0, 0, 1, 1, 1, 0, 1, 0) = 111010$.

[23] Andere unmittelbar verfügbare endliche Körper sind alle Strukturen $\mathbb{Z}_p = (\{0, 1, \ldots, p-1\}, +_p, \cdot_p, 0, 1)$, für Primzahlen $p$, wo $+_p$ und $\cdot_p$ für die Addition und die Multiplikation modulo $p$ stehen.

Mit diesen Operationen erhält $\mathbb{F}[X]$ die Struktur eines *Rings*: Die Addition ist Gruppe mit neutralem Element 0 (Nullpolynom), die Multiplikation ist assoziativ mit neutralem Element $1 = (\ldots, 0, 0, 0, 1)$, die Distributivgesetze gelten.

Als Grundmenge des zu konstruierenden Körpers verwenden wir $\mathbb{F}^k$, die Menge aller $k$-Tupel über $\mathbb{F}$. Dies entspricht der Menge aller Polynome vom Grad bis zu $k-1$. Diese Menge hat genau $|\mathbb{F}|^k$ Elemente. Im Fall $\mathbb{F} = \mathbb{Z}_2$ erhalten wir $\{0,1\}^k$, die Menge aller Bitstrings der Länge $k$.

Als Addition $\oplus$ benutzen wir die gewöhnliche Addition von Polynomen, also die komponentenweise Addition der Tupel. Bei $\mathbb{F} = \mathbb{Z}_2^k$ ist dies einfach das bitweise XOR. Wir geben die Additionstafel für $k = 4$ an. Die Elemente des Körpers $\mathrm{GF}(2^4)$ werden dabei wir üblich als Hexadezimalziffern geschrieben. Achtung: Man muss die Einträge richtig interpretieren, nämlich als Bitstrings, nicht als Zahlen: $\texttt{5} \oplus \texttt{C} = 0101 \oplus 1100 = 1001 = \texttt{9}$.

| $\oplus$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 1 | 1 | 0 | 3 | 2 | 5 | 4 | 7 | 6 | 9 | 8 | B | A | D | C | F | E |
| 2 | 2 | 3 | 0 | 1 | 6 | 7 | 4 | 5 | A | B | 8 | 9 | E | F | C | D |
| 3 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | B | A | 9 | 8 | F | E | D | C |
| 4 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | C | D | E | F | 8 | 9 | A | B |
| 5 | 5 | 4 | 7 | 6 | 1 | 0 | 3 | 2 | D | C | F | E | 9 | 8 | B | A |
| 6 | 6 | 7 | 4 | 5 | 2 | 3 | 0 | 1 | E | F | C | D | A | B | 8 | 9 |
| 7 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | F | E | D | C | B | A | 9 | 8 |
| 8 | 8 | 9 | A | B | C | D | E | F | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 9 | 9 | 8 | B | A | D | C | F | E | 1 | 0 | 3 | 2 | 5 | 4 | 7 | 6 |
| A | A | B | 8 | 9 | E | F | C | D | 2 | 3 | 0 | 1 | 6 | 7 | 4 | 5 |
| B | B | A | 9 | 8 | F | E | D | C | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 |
| C | C | D | E | F | 8 | 9 | A | B | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 |
| D | D | C | F | E | 9 | 8 | B | A | 5 | 4 | 7 | 6 | 1 | 0 | 3 | 2 |
| E | E | F | C | D | A | B | 8 | 9 | 6 | 7 | 4 | 5 | 2 | 3 | 0 | 1 |
| F | F | E | D | C | B | A | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Die Multiplikation $\odot$ ist etwas komplizierter.

Es wird ein *irreduzibles Polynom* $f = X^k + a_{k-1} \cdot X^{k-1} + \cdots + a_1 \cdot X + a_0$ vom Grad $k$ (mit „Leitkoeffizient“ $a_k = 1$) benötigt, also ein Koeffiziententupel $(1, a_{k-1}, \ldots, a_1, a_0)$ mit der Eigenschaft, dass man *nicht* $f = f_1 \cdot f_2$ schreiben kann, für Polynome $f_1$ und $f_2$, die Grad $\geq 1$ haben. Man kann zeigen, dass es für jedes $k \geq 2$ stets solche Polynome gibt. Sie können mit randomisierten Algorithmen effizient gefunden werden. Hier einige Beispiele für $\mathbb{F} = \mathbb{Z}_2$. Wie üblich schreibt man die Koeffizientenfolge als Bitstring. Dieser Bitstring kann dann auch als natürliche Zahl (dezimal geschrieben) interpretiert werden (letzte Spalte).

| $k$ | irreduzibles Polynom vom Grad $k$ über $\mathbb{Z}_2$ | Kurzform | Zahl |
|---|---|---|---|
| 1 | $X+1$ | 11 | 3 |
| 2 | $X^2+X+1$ | 111 | 7 |
| 3 | $X^3+X+1$ | 1011 | 11 |
| 4 | $X^4+X+1$ | 10011 | 19 |
| 5 | $X^5+X^2+1$ | 100101 | 37 |
| 6 | $X^6+X+1$ | 1000011 | 67 |
| 7 | $X^7+X^3+1$ | 10001001 | 137 |
| 8 | $X^8+X^4+X^3+X+1$ | 100011011 | 283 |

Das Polynom $X^2+1=(1,0,1)=101$ ist nicht irreduzibel, da über $\mathbb{Z}_2$ die Gleichung $(X+1)\cdot(X+1)=X^2+1$ gilt.

Die Multiplikation funktioniert nun wie folgt: Wenn $g$ und $h$ gegeben sind, berechnet man das Produkt $g\cdot h$ (Grad maximal $2k-2$) und bestimmt den Rest $r$ von $g\cdot h$ bei der Division durch $f$. Dieser Rest (genannt „$g\cdot h \bmod f$“) ist das eindeutig bestimmte Polynom $r$ vom Grad höchstens $k-1$, das

$$g\cdot h = q\cdot f + r$$

erfüllt, für ein „Quotientenpolynom“ $q$.

*Beispiel:* Sei $k=4$ und $f=(1,0,0,1,1)=10011$. Die Faktoren seien $g=(1,0,1,1)=1011$ und $h=(0,1,1,0)=110$. Das Produkt ist $g\cdot h=(1,1,1,0,1,0)=111010$. Wenn wir hiervon $f\cdot(X+1)=(1,1,0,1,0,1)=110101$ subtrahieren, erhalten wir das Produkt $(1,1,1,1)=1111$ im Körper.

Man kann durch geduldiges Multiplizieren und Bilden von Resten (also Dividieren von Polynomen) in dieser Weise eine komplette Multiplikationstabelle aufstellen. Geschickter ist Folgendes: Man berechnet alle Produkte $g\cdot h \bmod f$, für die in $g$ und in $h$ jeweils die ersten beiden Bits oder die letzten beiden Bits 0 sind. (Wenn man beobachtet, dass $(0,0,0,1)=0001$ das neutrale Element ist, und die Kommutativität berücksichtigt, muss man nur 15 Produkte berechnen.) Es ergibt sich die folgende Tabelle.

| $\odot$ | 0000 | 0001 | 0010 | 0011 | 0100 | 1000 | 1100 |
|---|---|---|---|---|---|---|---|
| 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 | 0000 |
| 0001 | 0000 | 0001 | 0010 | 0011 | 0100 | 1000 | 1100 |
| 0010 | 0000 | 0010 | 0100 | 0110 | 1000 | 0011 | 1011 |
| 0011 | 0000 | 0011 | 0110 | 0101 | 1100 | 1011 | 0111 |
| 0100 | 0000 | 0100 | 1000 | 1100 | 0011 | 0110 | 0101 |
| 1000 | 0000 | 1000 | 0011 | 1011 | 0110 | 1100 | 1010 |
| 1100 | 0000 | 1100 | 1011 | 0111 | 0101 | 1010 | 1111 |

Wir notieren weiterhin Bitstrings $(a_3, a_2, a_1, a_0) = a_3a_2a_1a_0$ hexadezimal. Das Polynom $g = (1, 0, 1, 1) = 1011$ ist also $\mathtt{B} = 1000 \oplus 0011 = \mathtt{8} \oplus \mathtt{3}$, das Polynom $h = (0, 1, 1, 0) = 0110$ ist $\mathtt{6} = 0100 \oplus 0010 = \mathtt{4} \oplus \mathtt{2}$.

Die Tabelle sieht dann so aus:

| $\odot$ | 0 | 1 | 2 | 3 | 4 | 8 | C |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 2 | 3 | 4 | 8 | C |
| 2 | 0 | 2 | 4 | 6 | 8 | 3 | B |
| 3 | 0 | 3 | 6 | 5 | C | B | 7 |
| 4 | 0 | 4 | 8 | C | 3 | 6 | 5 |
| 8 | 0 | 8 | 3 | B | 6 | C | A |
| C | 0 | C | B | 7 | 5 | A | F |

Nun wollen wir beliebige Polynome multiplizieren. *Beispiel*: Um $g = (1, 0, 1, 1) = 1011 = \mathtt{B}$ und $h = (0, 1, 1, 0) = 0110 = \mathtt{6}$ zu multiplizieren, rechnet man unter Benutzung der Tabelle und des Distributivgesetzes wie folgt (Multiplikation modulo $f$):

$$\begin{aligned}\mathtt{B} \odot \mathtt{6} &= (\mathtt{8} \oplus \mathtt{3}) \odot (\mathtt{4} \oplus \mathtt{2}) = (\mathtt{8} \odot \mathtt{4}) \oplus (\mathtt{8} \odot \mathtt{2}) \oplus (\mathtt{3} \odot \mathtt{4}) \oplus (\mathtt{3} \odot \mathtt{2}) \\ &= \mathtt{6} \oplus \mathtt{3} \oplus \mathtt{C} \oplus \mathtt{6} \\ &= 0110 \oplus 0011 \oplus 1100 \oplus 0110 = 1111 = \mathtt{F}.\end{aligned}$$

Mit etwas Geduld und unter Ausnutzung des Distributivgesetzes lässt sich auf diese Weise die folgende Multiplikationstabelle für die Elemente von $\mathbb{Z}_2^4$ finden.

| $\odot$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 2 | 0 | 2 | 4 | 6 | 8 | A | C | E | 3 | 1 | 7 | 5 | B | 9 | F | D |
| 3 | 0 | 3 | 6 | 5 | C | F | A | 9 | B | 8 | D | E | 7 | 4 | 1 | 2 |
| 4 | 0 | 4 | 8 | C | 3 | 7 | B | F | 6 | 2 | E | A | 5 | 1 | D | 9 |
| 5 | 0 | 5 | A | F | 7 | 2 | D | 8 | E | B | 4 | 1 | 9 | C | 3 | 6 |
| 6 | 0 | 6 | C | A | B | D | 7 | 1 | 5 | 3 | 9 | F | E | 8 | 2 | 4 |
| 7 | 0 | 7 | E | 9 | F | 8 | 1 | 6 | D | A | 3 | 4 | 2 | 5 | C | B |
| 8 | 0 | 8 | 3 | B | 6 | E | 5 | D | C | 4 | F | 7 | A | 2 | 9 | 1 |
| 9 | 0 | 9 | 1 | 8 | 2 | B | 3 | A | 4 | D | 5 | C | 6 | F | 7 | E |
| A | 0 | A | 7 | D | E | 4 | 9 | 3 | F | 5 | 8 | 2 | 1 | B | 6 | C |
| B | 0 | B | 5 | E | A | 1 | F | 4 | 7 | C | 2 | 9 | D | 6 | 8 | 3 |
| C | 0 | C | B | 7 | 5 | 9 | E | 2 | A | 6 | 1 | D | F | 3 | 4 | 8 |
| D | 0 | D | 9 | 4 | 1 | C | 8 | 5 | 2 | F | B | 6 | 3 | E | A | 7 |
| E | 0 | E | F | 1 | D | 3 | 2 | C | 9 | 7 | 6 | 8 | 4 | A | B | 5 |
| F | 0 | F | D | 2 | 9 | 6 | 4 | B | 1 | E | C | 3 | 8 | 7 | 5 | A |

Das neutrale Element der Multiplikation ist das Polynom $\mathtt{1} = (0, \ldots, 0, 1) = 0 \ldots 01$. Man beobachtet, dass in jeder Zeile (und ebenso in jeder Spalte) der Tabelle, außer der für das Nullpolynom, alle Elemente genau einmal vorkommen. Dies ist für jeden Körper $\mathbb{F}$ und jedes beliebige $k$ so, und es folgt daraus, dass die Elemente der Menge $\mathbb{F}^k - \{0\}$ ein Inverses haben.

**Fakt:** Wenn man die Multiplikation wie beschrieben definiert, mit einem irreduziblen Polynom $f$ mit Leitkoeffizient $a_k = 1$, dann gibt es für jedes Polynom $g \neq (0, \ldots, 0)$ vom Grad $< k$ (genau) ein Polynom $h$ mit $g \cdot h \bmod f = (0, \ldots, 0, 1) = \mathtt{1}$. Dieses Polynom $h$ ist also $g^{-1}$, das multiplikative Inverse von $g$.

(*Beweisidee*: Man zeigt, dass die Abbildung $h \mapsto g \cdot h \bmod f$ in der Menge dieser Polynome injektiv ist. Ist $h_1 \cdot g \bmod f = h_2 \cdot g \bmod f$, so ist $(h_1 - h_2) \cdot g \bmod f = 0$, also ist $(h_1 - h_2) \cdot g$ durch $f$ teilbar. Nach einem Lemma über irreduzible Polynome (analog zur Situation bei Primzahlen) folgt, dass $f$ Teiler von $h_1 - h_2$ oder Teiler von $g$ sein muss. Weil $g$ nicht das Nullpolynom ist und Grad $< k$ hat, kann $f$ kein Teiler von $g$ sein. Also teilt $f$ das Polynom $h_1 - h_2$. Da auch dieses Grad $< k$ hat, muss $h_1 - h_2$ das Nullpolynom sein, also $h_1 = h_2$ gelten. – Aus der Injektivität von $h \mapsto g \cdot h \bmod f$ folgt die Surjektivität, also gibt es ein $h$ mit $g \cdot h \bmod f = 1$.)

Weil die üblichen Rechenregeln in der Struktur $(\mathbb{F}^k, \oplus, \odot, 0, 1)$ mit den angegebenen Operationen leicht nachzuweisen sind, bedeutet dies, dass wir einen Körper mit $|\mathbb{F}|^k$ Elementen erhalten haben. Wir nennen ihn $\mathbb{F}[X]/(f)$, für das gewählte irreduzible Polynom $f$ vom Grad $k$. Wenn $|\mathbb{F}|^k = q$ ist, nennen wir diesen Körper $\mathrm{GF}(q)$.[24] Im Fall $\mathbb{F} = \mathbb{Z}_2$ erhalten wir so Körper $\mathrm{GF}(2^k)$, deren zugrundeliegende Menge einfach $\{0,1\}^k$ ist, die Menge der Bitstrings der Länge $k$.

**Bemerkung:** In der Algebra zeigt man folgende Fakten über endliche Körper:

- Es gibt einen endlichen Körper $\mathrm{GF}(q)$ mit $q$ Elementen genau dann wenn $q = p^r$ für eine Primzahl $p$ und einen Exponenten $r \geq 1$.
- Es ist gleichgültig, auf welchem Konstruktionsweg man zu einem Körper mit $q$ Elementen gelangt: alle diese Körper sind *isomorph*, also strukturell identisch. (Insbesondere ist der Körper $\mathrm{GF}(2^k)$ immer „der gleiche“, ganz egal welches irreduzible Polynom $f$ man benutzt.) Die Tabellen für die Operationen können eventuell unterschiedlich aussehen, aber nur aufgrund von Umbenennungen von Objekten.

Ab hier schreiben wir $+$ für die Körperaddition und $\cdot$ für die Körpermultiplikation.

**Bemerkungen zur Zeit- und Platzeffizienz:** Allgemein, und für beliebig große $k$, kann

[24] „GF“ steht für „*Galois field*“, nach Évariste Galois (1811–1832), einem französischen Mathematiker. „Körper“ heißt auf englisch „*field*“.

man für ein festes Polynom $f$ den Rest der Division durch $f$ stets als Produkt des Zeilenvektors, der $g \cdot h$ darstellt, mit einer festen $((2k-1) \times k)$-Matrix $M_f$ erhalten.

Damit ist die Komplexität der Multiplikation in einem solchen Körper definiert: Es muss das Produkt $g \cdot h$ berechnet werden („Konvolution“ von zwei Bitvektoren) und dann eine Vektor-Matrix-Multiplikation ausgeführt werden. Über dem Körper $\mathbb{Z}_2$ lässt sich die Vektor-Matrix-Multiplikation mit Hilfe der wortweisen XOR-Operation sehr effizient durchführen. Wenn zusätzlich $f$ nur wenige Terme hat, hat $M_f$ nur wenige 1-Einträge und die Matrix-Vektor-Multiplikation läuft auf einer Addition weniger Shifts eines Vektors hinaus. Für die Konvolution ist effiziente Ausführung etwas schwieriger; es gibt aber moderne Prozessoren, die diese Operation für konstante Wortlängen bereitstellen, was diese Operation auch für beliebige Wortlängen beschleunigt.

Wenn die Bitlänge $k$ kurz und bekannt ist, wird man Tabellen benutzen. (Zum Beispiel benutzt AES einen Körper der Größe 256.) Für die Multiplikation verwendet man auch gerne Potenz- und Logarithmentabellen.

**Fakt 2.7** *Sei $\mathbb{F}$ ein endlicher Körper mit $q$ Elementen. Dann gibt es in $\mathbb{F}$ ein „primitives Element“ $g$, d. h., $g$ erfüllt*

$$\{g^i \mid 0 \leq i < q-1\} = \mathbb{F} - \{0\}.$$

(*$g$ ist erzeugendes Element der multiplikativen Gruppe von $\mathbb{F}$.*)

*Beispiel*: Wir hatten oben eine Multiplikationstabelle für $\mathrm{GF}(2^4)$ aufgestellt. Mit ihrer Hilfe findet man leicht die ersten 15 Potenzen von `2`:

Potenztabelle:

| $i$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\exp_2(i) = 2^i$ | 1 | 2 | 4 | 8 | 3 | 6 | C | B | 5 | A | 7 | E | F | D | 9 |

Diese Potenzen sind alle verschieden, also ist $g =$ `2` primitives Element. Die entsprechende Logarithmentabelle sieht so aus:

| $x$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\log_2(x)$ | 0 | 1 | 4 | 2 | 8 | 5 | 10 | 3 | 14 | 9 | 7 | 6 | 13 | 11 | 12 |

Achtung: Der Index der Exponential- und der Logarithmenfunktion ist nicht die natürliche Zahl 2, sondern `2`, das ist das Element $X = 0010$ von $\mathrm{GF}(2^4)$.

Allgemein: Wenn $g$ primitives Element ist, dann betrachtet man die *Exponentialfunktion* zur Basis $g$:

$$\exp_g \colon \mathbb{Z} \ni i \mapsto g^i \in \mathbb{F} - \{0\}.$$

Diese ist periodisch: Wenn $i-j$ durch $q-1$ teilbar ist, dann ist $g^i = g^j$. Es genügt also, die Werte $g^i$ für $0 \leq i < q-1$ zu kennen. Man erstellt eine Tabelle mit diesen $q-1$ Werten.

Die Umkehrfunktion ist die *Logarithmusfunktion* $\log_g$ zur Basis $g$. Sie ordnet jedem $x \in \mathbb{F} - \{0\}$ den (eindeutig bestimmten) Wert $i \in \{0, 1, \ldots, q-2\}$ mit $x = g^i$ zu, genannt $\log_g(x)$. Es gilt $\log_g(g^i) = i$ für $i \in \{0, 1, \ldots, q-2\}$ und $g^{\log_g(x)} = x$ für alle $x \in \mathbb{F} - \{0\}$. Man erstellt ebenfalls eine Tabelle mit diesen Werten.

Nun kann man leicht multiplizieren.

Aufgabe: Berechne $z = x \cdot y$.

Falls $x = 0$ oder $y = 0$ in $\mathbb{F}$, ist $z = 0$. Andernfalls schaue in die log-Tabelle, um $i = \log_g(x)$ und $j = \log_g(y)$ zu finden. Berechne $k = (i + j) \bmod (q-1)$. Das Ergebnis $z = g^k$ liest man nun in der exp-Tabelle an Stelle $k$ ab.

*Beispiel*: In $\mathrm{GF}(2^4)$ seien $x = \mathtt{9}$ und $y = \mathtt{C}$ gegeben. Die $\log_{\mathtt{2}}$-Tabelle liefert $i = \log_{\mathtt{2}}(\mathtt{9}) = 14$ und $j = \log_{\mathtt{2}}(\mathtt{C}) = 6$. Wir erhalten $k = (14 + 6) \bmod 15 = 5$ und $z = g^5 = \mathtt{6}$ mit der $\exp_{\mathtt{2}}$-Tabelle. Man kontrolliert mit der Multiplikationstabelle für $\mathrm{GF}(2^4)$, dass dort tatsächlich $\mathtt{9} \cdot \mathtt{C} = \mathtt{6}$ gilt.

Aufwand für eine Multiplikation: Drei Zugriffe auf Tabellen, eine modulare *Addition*!

Der Rechenaufwand für die Erstellung der Tabellen und auch ihr Platzbedarf ist $O(q)$ und damit akzeptabel, wenn $q$ nicht zu groß ist. Im Fall von $q = 256$ hat man zwei Tabellen mit je 255 Einträgen und kann damit sehr leicht in konstanter Zeit multiplizieren. Auch Tabellen für $q = 2^{16} = 65536$ stellen weiter kein Problem dar. Für noch größere $q$ benötigt man weitere Tricks zum Ermitteln diskreter Logarithmen mit Hilfe von Tabellen der Größe etwa $\sqrt{q}$. (Wir werden später darauf zurückkommen.) Sollte $q$ sehr groß werden (Hunderte oder Tausende von Bits), wird man, wie oben skizziert, Konvolution und Matrix-Vektor-Multiplikation benutzen.

## 2.2 AES – Advanced Encryption Standard

Der „Advanced Encryption Standard (AES)“ ist ein symmetrisches Verschlüsselungsverfahren (d. h., beide Kommunikationspartner benutzen denselben Schlüssel). AES wird in vielen Standardverfahren bei der Kommunikation im Internet benutzt, zum Beispiel bei *Secure Socket Layer* (SSL)/*Transport Layer Security* (TLS) und bei *Secure Shell* (SSH). (Mehr Details hierzu in der Vorlesung „Network Security“.)

Die AES-Chiffren gehören in die Klasse der Substitutions-Permutations-Kryptosysteme, auch wenn es in Details Abweichungen von der in Abschnitt 2.1 vorgestellten Struktur gibt. Bei der standardisierten Version AES ist die Klartextlänge und Chiffretextlänge stets $\ell = 128$, die Schlüssellänge 128, 192 oder 256 Bits. Wir konzentrieren uns auf die Variante mit Klartextlänge 128, Schlüssellänge 128.[25]

In AES wird Arithmetik im Körper $\mathrm{GF}(2^8)$ benutzt, dessen Elemente 8-Bit-Vektoren, also Bytes, entsprechen.

Klartext und alle Zwischenergebnisse bei der Ver- und Entschlüsselung sind Strings aus 128 Bits, die als 16 Wörter von 8 Bit Länge aufgefasst werden (1 Wort = 1 Byte = 2 Hexziffern). Die Bytes werden als Elemente von $\mathrm{GF}(2^8)$ aufgefasst, konstruiert aus $\mathbb{Z}_2$ mit irreduziblem Polynom

$$f(X) = X^8 + X^4 + X^3 + X + 1 \qquad (= (1,0,0,0,1,1,0,1,1)\ ).$$

Die 16 Bytes werden als $4\times4$-Matrix (mit Einträgen aus $\mathrm{GF}(2^8) = \mathrm{GF}(256)$) aufgefasst. Die Leseanordnung ist dabei spaltenweise von links nach rechts, wobei Spalten von oben nach unten gelesen werden. Ein Bitstring $z \in \{0,1\}^{128}$ wird also in 16 Bytes $z^{(0)}, z^{(1)}, \ldots, z^{(15)}$ mit $z = z^{(0)}z^{(1)}\ldots z^{(15)}$ aufgeteilt und wie folgt als Matrix $A(z) \in (\{0,1\}^8)^{4\times4}$ geschrieben:

$$A(z) = \begin{pmatrix} z^{(0)} & z^{(4)} & z^{(8)} & z^{(12)} \\ z^{(1)} & z^{(5)} & z^{(9)} & z^{(13)} \\ z^{(2)} & z^{(6)} & z^{(10)} & z^{(14)} \\ z^{(3)} & z^{(7)} & z^{(11)} & z^{(15)} \end{pmatrix}.$$

Die Einträge einer solchen Matrix sind mit $0 \le i, j \le 3$ indiziert, wie folgt:

$$A = \begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{pmatrix}.$$

[25] AES umfasst einige Spezialfälle der *Rijndael*-Familie von Chiffren, mit der Klartexte der Länge 128, 160, 192, 224 oder 256 Bits ver- und entschlüsselt werden können. Für die Varianten ist die hier angegebene Beschreibung zu ändern. Hierfür wird auf die Literatur verwiesen, insbesondere auf: Joan Daemen, Vincent Rijmen: The Design of Rijndael: AES – The Advanced Encryption Standard. Information Security and Cryptography, Springer 2002.

Aus jeder solchen Matrix ergibt sich durch spaltenweises Auslesen wieder ein Bitstring

$$A_{00}A_{10}A_{20}A_{30}A_{01}A_{11}A_{21}A_{31}A_{02}A_{12}A_{22}A_{32}A_{03}A_{13}A_{23}A_{33}.$$

Abkürzungen für Zeilen und Spalten einer Matrix $A$:

Zeile $i$: $\mathrm{row}_i(A) = (A_{i0}, A_{i1}, A_{i2}, A_{i3})$, für $i = 0, 1, 2, 3$.

Spalte $j$: $\mathrm{col}_j(A) = \begin{pmatrix} A_{0j} \\ A_{1j} \\ A_{2j} \\ A_{3j} \end{pmatrix}$, für $j = 0, 1, 2, 3$.

**Elementare Operationen auf Matrizen:**

- Für Matrizen $A$ und $B$ in $\mathrm{GF}(2^8)^{4\times4}$ bedeutet $A \oplus B$ die komponentenweise Anwendung der Addition im Körper $\mathrm{GF}(2^8)$.
  (Diese ist wiederum $\oplus_8$, die bitweise XOR-Operation auf Bytes.)

- Zyklischer Linksshift von Zeile $i$ um $h = 1, 2, 3$ Positionen:
  $\mathrm{rotLeft}_1((A_{i0}, A_{i1}, A_{i2}, A_{i3})) = (A_{i1}, A_{i2}, A_{i3}, A_{i0})$ usw.

  *Beispiel*: $\mathrm{rotLeft}_2((\mathtt{A1}, \mathtt{06}, \mathtt{4B}, \mathtt{EF})) = (\mathtt{4B}, \mathtt{EF}, \mathtt{A1}, \mathtt{06})$.

  In AES wird Zeile $i$ um $i$ Positionen nach links rotiert. Die Abbildung ist also folgende:

$$\begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{pmatrix} \quad \mapsto \quad \begin{pmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{11} & A_{12} & A_{13} & A_{10} \\ A_{22} & A_{23} & A_{20} & A_{21} \\ A_{33} & A_{30} & A_{31} & A_{32} \end{pmatrix}$$

  Diese Zeilenrotation ist eine Bitpermutation. Ihre Inverse besteht offenbar darin, Zeile $i$ um $i$ Positionen nach *rechts* zu rotieren. (Sie ist also nicht selbstinvers, anders als im SPKS-Schema in Abschnitt 2.1.)

- Zyklischer Shift von Spalte $j$ um $h = 1, 2, 3$ Positionen nach oben:
  $\mathrm{rotUp}_1((A_{0j}, A_{1j}, A_{2j}, A_{3j})^\mathsf{T}) = (A_{1j}, A_{2j}, A_{3j}, A_{0j})^\mathsf{T}$ usw.

  (Benötigt bei der Rundenschlüsselerzeugung.)

- **„Lineare Spaltendurchmischung“**

  $\mathrm{col}_j(A) \mapsto M \cdot \mathrm{col}_j(A)$, für $0 \le j \le 3$, für die feste Matrix

$$M = \begin{pmatrix} \mathtt{02} & \mathtt{03} & \mathtt{01} & \mathtt{01} \\ \mathtt{01} & \mathtt{02} & \mathtt{03} & \mathtt{01} \\ \mathtt{01} & \mathtt{01} & \mathtt{02} & \mathtt{03} \\ \mathtt{03} & \mathtt{01} & \mathtt{01} & \mathtt{02} \end{pmatrix} \in \mathrm{GF}(2^8)^{4\times4}\,.$$

Achtung: Gerechnet wird in $\mathrm{GF}(2^8)$!

*Beispiel*:

$$M \cdot \begin{pmatrix} \texttt{4F} \\ \texttt{B0} \\ \texttt{3E} \\ \texttt{A0} \end{pmatrix} = \begin{pmatrix} \texttt{02} \cdot \texttt{4F} \oplus \texttt{03} \cdot \texttt{B0} \oplus \texttt{01} \cdot \texttt{3E} \oplus \texttt{01} \cdot \texttt{A0} \\ \vdots \end{pmatrix} = \begin{pmatrix} \texttt{9E} \oplus \texttt{CB} \oplus \texttt{3E} \oplus \texttt{A0} \\ \vdots \end{pmatrix} = \begin{pmatrix} \texttt{CB} \\ \vdots \end{pmatrix}.$$

In AES werden einmal alle vier Spalten auf diese Weise transformiert. Dies kann man auch zu einer Matrixmultiplikation zusammenfassen:

$$A \mapsto M \cdot A.$$

Diese Operation „vermischt“ die Einträge, ist aber keine Bitpermutation mehr, da Bits nicht nur vertauscht werden, sondern mit Körperoperationen verrechnet. Allerdings ist $M$ invertierbar, so dass man die Operation auch wieder rückgängig machen kann, indem man mit

$$M^{-1} = \begin{pmatrix} \texttt{0E} & \texttt{0B} & \texttt{0D} & \texttt{09} \\ \texttt{09} & \texttt{0E} & \texttt{0B} & \texttt{0D} \\ \texttt{0D} & \texttt{09} & \texttt{0E} & \texttt{0B} \\ \texttt{0B} & \texttt{0D} & \texttt{09} & \texttt{0E} \end{pmatrix}$$

multipliziert.

- Die ***S*-Box** von AES ist eine feste bijektive Abbildung $\{0,1\}^8 \to \{0,1\}^8$, die durch Abb. 3 gegeben ist. Der mathematische Hintergrund der $S$-Box wird weiter unten diskutiert.
- Rundenschlüssel: Aus Schlüssel $k \in \{0,1\}^{128}$ und Rundennummer $r$ wird Rundenschlüssel $\kappa(k,r) \in \{0,1\}^{128} = (\{0,1\}^8)^{4\times 4}$ berechnet. Details hierzu folgen weiter unten.

**AES-Chiffrieralgorithmus**($X$: 128-Bit-Klartext, $k$: 128-Bit-Schlüssel)

// Umarrangiert: $X \in \mathrm{GF}(2^8)^{4\times 4}$: Klartext*matrix*; $k \in \{0,1\}^{128}$: Schlüssel.

1. Initialschritt („Weißschritt“):

   $U \leftarrow X \oplus \kappa(k, 0)$ // addiere Rundenschlüssel für Runde $r = 0$, als 128-Bit-String.

2. Für $r = 1, \ldots, 9$ tue

   (a) **Substitution** // Anwendung der $S$-Box auf jedes Byte in $U$

   $V_{ij} \leftarrow S(U_{ij})$, für $0 \le i, j \le 3$;

   (b) **Zeilenrotation** // $i$-te Zeile um $i$ Positionen nach links

   $\mathrm{row}_i(W) \leftarrow \mathrm{rotLeft}_i(\mathrm{row}_i(V))$, für $0 \le i \le 3$;

   (c) **Lineare Spaltendurchmischung** // $M$ wie oben beschrieben

   $Z \leftarrow M \cdot W$, für $0 \le i \le 3$;

   (d) **Schlüsseladdition** // Rundenschlüssel für Runde $r$

   $U \leftarrow Z \oplus \kappa(k, r)$ // komponentenweise Addition

3. Verkürzte Schlussrunde, $r = 10$, keine Spaltendurchmischung:

   (a) **Substitution**

   $V_{ij} \leftarrow S(U_{ij})$, für $0 \le i, j \le 3$;

   (b) **Zeilenrotation**

   $\mathrm{row}_i(Z) \leftarrow \mathrm{rotLeft}_i(\mathrm{row}_i(V))$, für $0 \le i \le 3$;

   (c) **Schlüsseladdition**

   $Y \leftarrow Z \oplus \kappa(k, 10)$

**Ausgabe:** $Y \in \mathrm{GF}(2^8)^{4\times 4}$, geschrieben als 128-Bit-Wort.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 63 | 7C | 77 | 7B | F2 | 6B | 6F | C5 | 30 | 01 | 67 | 2B | FE | D7 | AB | 76 |
| 1 | CA | 82 | C9 | 7D | FA | 59 | 47 | F0 | AD | D4 | A2 | AF | 9C | A4 | 72 | C0 |
| 2 | B7 | FD | 93 | 26 | 36 | 3F | F7 | CC | 34 | A5 | E5 | F1 | 71 | D8 | 31 | 15 |
| 3 | 04 | C7 | 23 | C3 | 18 | 96 | 05 | 9A | 07 | 12 | 80 | E2 | EB | 27 | B2 | 75 |
| 4 | 09 | 83 | 2C | 1A | 1B | 6E | 5A | A0 | 52 | 3B | D6 | B3 | 29 | E3 | 2F | 84 |
| 5 | 53 | D1 | 00 | ED | 20 | FC | B1 | 5B | 6A | CB | BE | 39 | 4A | 4C | 58 | CF |
| 6 | D0 | EF | AA | FB | 43 | 4D | 33 | 85 | 45 | F9 | 02 | 7F | 50 | 3C | 9F | A8 |
| 7 | 51 | A3 | 40 | 8F | 92 | 9D | 38 | F5 | BC | B6 | DA | 21 | 10 | FF | F3 | D2 |
| 8 | CD | 0C | 13 | EC | 5F | 97 | 44 | 17 | C4 | A7 | 7E | 3D | 64 | 5D | 19 | 73 |
| 9 | 60 | 81 | 4F | DC | 22 | 2A | 90 | 88 | 46 | EE | B8 | 14 | DE | 5E | 0B | DB |
| A | E0 | 32 | 3A | 0A | 49 | 06 | 24 | 5C | C2 | D3 | AC | 62 | 91 | 95 | E4 | 79 |
| B | E7 | C8 | 37 | 6D | 8D | D5 | 4E | A9 | 6C | 56 | F4 | EA | 65 | 7A | AE | 08 |
| C | BA | 78 | 25 | 2E | 1C | A6 | B4 | C6 | E8 | DD | 74 | 1F | 4B | BD | 8B | 8A |
| D | 70 | 3E | B5 | 66 | 48 | 03 | F6 | 0E | 61 | 35 | 57 | B9 | 86 | C1 | 1D | 9E |
| E | E1 | F8 | 98 | 11 | 69 | D9 | 8E | 94 | 9B | 1E | 87 | E9 | CE | 55 | 28 | DF |
| F | 8C | A1 | 89 | 0D | BF | E6 | 42 | 68 | 41 | 99 | 2D | 0F | B0 | 54 | BB | 16 |

Abbildung 3: Die $S$-Box für AES. $S(z_1 z_0)$ für Hexziffern $z_1$, $z_0$ steht in Zeile $z_1$, Spalte $z_0$. Beispiel: $S(\texttt{A4}) = \texttt{49}$.

### Die *S*-Box von AES:

Die $S$-Box ist als 16×16-Tabelle gegeben, siehe Abb. 3. Zur kompakten Darstellung wird ein Element $(a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)$ von $\mathrm{GF}(2^8)$ in zwei Hexziffern zerlegt: $(a_7, a_6, a_5, a_4)$ ist der Zeilenindex, $(a_3, a_2, a_1, a_0)$ der Spaltenindex der Position von $S((a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0))$.

Interessanterweise steckt hinter dieser chaotisch aussehenden Tabelle eine einfache Formel, die eine „nichtlineare“ Komponente in AES einbringt. Zu $x \in \mathrm{GF}(2^8)$ betrachtet man $x^{-1}$, das multiplikative Inverse. Künstlich definiert man $\texttt{00}^{-1} := \texttt{00}$ (nur hier und nur für diesen Zweck). Weiter ist $h$ eine invertierbare lineare Funktion auf dem $\mathbb{Z}_2$-Vektorraum $\{0,1\}^8$. Dazu werden die Elemente von $\mathrm{GF}(2^8)$ als Spaltenvektoren aufgefasst, die 8 Bits enthalten.

$$h((a_7, \ldots, a_0))^{\mathsf{T}} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} a_7 \\ a_6 \\ a_5 \\ a_4 \\ a_3 \\ a_2 \\ a_1 \\ a_0 \end{pmatrix}$$

Dann ist die $S$-Box wie folgt definiert:

$$S(x) = h(x^{-1}) \oplus_8 \texttt{63}.$$

Auch hinter der Matrix $M$, die im Teil (c) „Lineare Spaltendurchmischung“ benutzt wird, steckt eine relativ einfache lineare Operation über einer passenden algebraischen Struktur.

Wir nehmen den endlichen Körper $\mathbb{F}_{2^8} = \mathrm{GF}(2^8)$ als Ausgangssituation und betrachten Polynome $b_0 + b_1X + b_2X^2 + b_3X^3$ vom Grad maximal 3 über diesem Körper. Diese bilden die Grundmenge des *Rings* $\mathbb{F}_{2^8}[X]/(X^4+1)$: Addition wie gewöhnlich, Multiplikation modulo $X^4+1$. Ein solches Polynom wird durch den Spaltenvektor $(b_0, b_1, b_2, b_3)^\mathsf{T} \in (\mathbb{F}_{2^8})^4$ beschrieben. Wir multiplizieren dieses Polynom über $\mathbb{F}_{2^8}[X]/(X^4+1)$ mit dem festen Polynom $c(X) = \texttt{02}+\texttt{01}\cdot X+\texttt{01}\cdot X^2+\texttt{03}\cdot X^3$. (Da die Koeffizienten Elemente von $\mathbb{F}_{2^8}$ sind, werden sie durch zwei Hexziffern dargestellt.) Der Koeffizientenvektor $(c_0, c_1, c_2, c_3)^\mathsf{T}$ des Produktpolynoms bildet die transformierte Spalte. Man kann nachrechnen, dass sie sich als $M \cdot (b_0, b_1, b_2, b_3)^\mathsf{T}$ ergibt.

Die Rundenschlüssel werden mit einem iterativen Verfahren berechnet. Dabei wird zunächst Spalte 3 intensiv manipuliert und auf Spalte 0 addiert.

**AES-Rundenschlüsselberechnung:**

$K_0 = \kappa(k, 0) \leftarrow k$ // 128-Bit-Schlüssel als $4 \times 4$-Matrix

Für $r = 1, \ldots, 10$ führe Runde $r$ aus.

Input: Rundenschlüssel $K_{r-1}$ als $4 \times 4$-Matrix
Output: Rundenschlüssel $K_r$ als $4 \times 4$-Matrix

Berechnung von Spalte 0:
(a) **Rotation:** // analog zur Zeilenrotation:
$U \leftarrow \mathrm{rotUp}_1(\mathrm{col}_3(K_{r-1}))$ // Spalte 3 zyklisch eine Position nach oben
(b) **Substitution:**
$V_i \leftarrow S(U_i)$, für $0 \le i \le 3$;
(c) **Konstantenaddition:**
$V_0 \leftarrow V_0 + \texttt{02}^{r-1}$; // Potenz in $\mathrm{GF}(2^8)$;
(d) **Addition auf Spalte 0:**
$\mathrm{col}_0(K_r) \leftarrow \mathrm{col}_0(K_{r-1}) + V$; // Vektoraddition;
Berechnung von Spalten 1 bis 3:
**Iterative Addition:**
für $i = 1, 2, 3$: $\mathrm{col}_i(K_r) \leftarrow \mathrm{col}_i(K_{r-1}) + \mathrm{col}_{i-1}(K_r)$; // Vektoraddition;

**Ergebnis:** $K_r = \kappa(k, r)$ für $r = 0, 1, \ldots 10$.

**Entschlüsselung:** Man geht nach dem schon diskutierten grundsätzlichen Ansatz bei

Substitutions-Permutations-Kryptosystemen vor. Erst werden *alle Rundenschlüssel* berechnet. Diese werden in umgekehrter Reihenfolge benutzt. Für die Substitution wird stets die inverse $S$-Box $S^{-1}$ benutzt. Rotationen werden in umgekehrter Richtung ausgeführt. Die „lineare Spaltendurchmischung“ wird mit der zu $M$ inversen Matrix $M^{-1}$ ausgeführt. Ansonsten ist der Ablauf völlig analog der Verschlüsselung.

Aktuelle Lage: Bisher gilt AES als praktisch sicher (insbesondere die 192-Bit- und die 256-Bit-Variante), wenn auch nicht in dem im Folgenden zu besprechenden technischen Sinn. Es wurden einige Angriffe vorgeschlagen, die zu schnellerem „Brechen“ führen als dem vollständigen Durchsuchen des Schlüsselraums (nur noch $2^{99}$ Schlüssel müssen getestet werden ...), aber dies führt nicht zu praktisch durchführbaren Verfahren.

## 2.3 Bemerkungen zu randomisierten Algorithmen

Wir benötigen ein Algorithmenmodell, um die Angreiferin Eva zu modellieren. Offensichtlich wird sie alle ihr zur Verfügung stehenden Mittel einsetzen, insbesondere auch Zufallsexperimente (wenn es ihr nützt). Also müssen unsere Algorithmen auch Zufallsexperimente ausführen können. Grundsätzlich werden wir klassische Turingmaschinen bzw. üblichen Pseudocode (für gewöhnliche Computer) verwenden und ihre Laufzeit messen, allerdings mit einigen Änderungen/Erweiterungen/Einschränkungen, wie im Weiteren erläutert wird.

### 2.3.1 Ressourcenverbrauch

Da wir nicht erwarten können, dass reale Kryptosysteme in einem idealen Sinn wie der informationstheoretischen Sicherheit sicher sind, wollen wir Sicherheit gegenüber einer Angreiferin Eva studieren, die nur beschränkte Ressourcen hat. Welche Ressourcen kommen in Frage? Sicherlich Zeitverbrauch bzw. Anzahl der Rechenoperationen, aber auch andere.

*Vorüberlegung*: Ein „zeiteffizienter“ Angriff (mit Klartextwahl, „chosen-plaintext attack“, CPA)

Sei $\mathcal{B} = (\{0,1\}^\ell, \{0,1\}^s, \{0,1\}^\ell, e, d)$ ein Block-Kryptosystem und seien $x_i, y_i \in \{0,1\}^\ell$ für $1 \leq i \leq t$ paarweise verschiedene Klar- bzw. Chiffretexte. Ein Schlüssel $k \in \{0,1\}^s$ ist *konsistent* mit den Paaren $(x_i, y_i)$, wenn $e(x_i, k) = y_i$ für alle $1 \leq i \leq t$ gilt. Für realistische (d. h. praktisch relevante) SPKS gibt es im Fall $t \approx 5$ höchstens einen konsistenten Schlüssel. (Man denke an AES. $t = 5$ bedeutet, dass $5 \cdot 128$ Bit Information über $k$ vorliegen, eventuell redundant, aber $k$ ist nur 128 Bit lang.)

Wir können also für Eva in Szenario 2 den folgenden Angriff nach dem CPA-Muster planen:

1. Lasse fünf Klartexte $x_1, \ldots, x_5$ zu $y_1, \ldots, y_5$ verschlüsseln.
2. „Schaue nach“, welcher Schlüssel $k$ mit den Paaren $(x_1, y_1), \ldots, (x_5, y_5)$ konsistent ist (wenigstens einer ist es und es gibt höchstens einen).
3. Höre Chiffretext $y$ ab und berechne $x = d(y, k)$ aus $y$.

Um in Schritt 2. „nachsehen“ zu können, benötigt Eva eine Tabelle mit allen möglichen Chiffretextkombinationen für die fünf Klartexte $x_1, \ldots, x_5$. Dafür gibt es $\leq |K| = 2^s$ Möglichkeiten, nämlich für jeden Schlüssel eine. Die Hashtabelle oder der Suchbaum (oder die entsprechende Struktur in einem Turingmaschinen-Programm) hat Größe $2^s$. Wenn der Programmtext oder die Turingmaschine binäre Suche (oder einen binären Suchbaum) benutzt, dann führt dieser „Angriff“ in Zeit $O(\log(|K|)) = O(\log(2^s)) = O(s)$ zum Klartext $x$.

Dieses Vorgehen ist natürlich unrealistisch, denn niemand kann ein so großes Programm schreiben oder speichern, wenn etwa (wie bei AES) die Zahl der Schlüssel $2^{128} > 10^{38}$ beträgt. Um diesen Trick auszuschließen, werden wir den *Ressourcenverbrauch* eines Algorithmus als *Summe* von Laufzeit und Länge des Programmcodes (= Größe der Transitionstabelle der Turingmaschine) messen. Die Programmgröße ist eine untere Schranke für die Zeit, die Eva zur *Erstellung* ihres Programms benötigt. Unser Ressourcenmaß erfasst also die Zeit für die Erstellung und die Zeit für die Ausführung des Algorithmus.

Unsere Algorithmen werden oft mit Kryptosystemen einer festen Blocklänge $\ell$ und einer fixen Anzahl von Klartext-/Chiffretext-Paaren arbeiten. Damit sind nur endlich viele Eingaben möglich, der Ressourcenverbrauch kann also nicht asymptotisch („bei großen Eingaben“) angegeben werden. Daher betrachten wir zunächst Algorithmen mit konstanter Laufzeit. Das führt dazu, dass alle eventuell vorhandenen Schleifen nur konstant oft durchlaufen werden. Damit können wir aber auf Schleifenanweisungen vollständig verzichten. Die resultierenden Programme heißen „Straight-Line-Programme“. Man kann sie sich in Pseudocode geschrieben vorstellen (ohne „while“ und „for“ und ohne Rückwärtssprünge), oder als Turingmaschinenprogramme mit der Eigenschaft, dass nie eine Programmzeile zweimal ausgeführt wird.

### 2.3.2 Randomisierung bei Straight-Line-Programmen

In unseren Algorithmen wollen wir zusätzlich die Anweisung „$y \leftarrow \mathrm{flip}(M)$“ erlauben, wobei $M$ eine endliche Menge ist. Die Idee dabei ist, dass der Variablen $y$ ein zufälliges Element von $M$ (bzgl. der Gleichverteilung auf $M$) zugewiesen wird. Damit berechnen unsere Algorithmen keine Funktionen, sondern zufällige Werte, die durch eine Wahrscheinlichkeitsverteilung gesteuert werden.

Um den Wahrscheinlichkeitsraum definieren zu können, machen wir die folgenden Annahmen (die keine Einschränkung darstellen):

(1) Bei nacheinander ausgeführten Kommandos der Form $y \leftarrow \mathrm{flip}(M)$, mit identischem oder unterschiedlichem $M$, werden immer neue, unabhängige Zufallsexperimente ausgeführt.

(2) Wie eben diskutiert, enthalten unsere Algorithmen keine Schleifen. Wir können daher jedes Ergebnis eines Zufallsexperiments in einer eigenen Variable speichern. Zusätzlich können wir die flip-Kommandos aus dem gesamten Programm, auch den bedingten Anweisungen, herausziehen und sie alle (vorab, auf Vorrat) ausführen. Damit können wir annehmen: Wenn die Anweisung $y \leftarrow \mathrm{flip}(M)$ im Programmtext vorkommt, dann wird sie in jedem Durchlauf des Algorithmus (unabhängig von der Eingabe und den Ergebnissen der flip-Anweisungen) genau einmal ausgeführt.

Abkürzung: flip($\ell$) für flip($\{0,1\}^\ell$) ($\ell$-facher Münzwurf) und flip() für flip(1), also flip($\{0,1\}$) (einfacher Münzwurf).

Sei nun $A$ ein randomisierter Algorithmus (also ein Straight-Line-Programm), in dem die flip-Anweisungen $y_i \leftarrow \text{flip}(M_i)$ für $1 \leq i \leq r$ vorkommen. Dann verwenden wir als zugrundliegenden Wahrscheinlichkeitsraum die Menge

$$M = M_1 \times M_2 \times \cdots \times M_r$$

mit der Gleichverteilung. Dies modelliert die Idee, dass die $r$ Zufallsexperimente jeweils mit der uniformen Verteilung und insgesamt unabhängig voneinander ausgeführt werden.

$I$ sei die Menge der Eingaben, $Z$ sei die Menge der Ausgaben.

Ist $x \in I$ eine Eingabe, so erhalten wir für jedes $m = (m_1, \ldots, m_r) \in M$ genau eine Ausgabe $A^m(x) \in Z$, indem wir in $A$ die Anweisung $y_i \leftarrow \text{flip}(M_i)$ durch $y_i \leftarrow m_i$ ersetzen. Auf diese Weise erhalten wir

- Für jedes $m \in M$ eine Funktion $A^m \colon I \to Z$, $x \mapsto A^m(x)$, und
- für jedes $x \in I$ eine Zufallsgröße $A(x) \colon M \to Z$, $m \mapsto A^m(x)$.

**Beispiel 2.8** Betrachte den folgenden Algorithmus:

$A(x : \{0,1\}^4) : \{0,1\}$ // nach dem Doppelpunkt: Typ der Eingabe bzw. Ausgabe
  $b_0 \leftarrow$ `flip`()
  $b_1 \leftarrow$ `flip`()
  $b_2 \leftarrow$ `flip`()
  $b_3 \leftarrow$ `flip`()
  `if` $b_0 = 1$ `then return` $x(0)$
  `if` $b_1 = 1$ `then return` $x(1)$
  `if` $b_2 = 1$ `then return` $x(2)$
  `if` $b_3 = 1$ `then return` $x(3)$
  `return` 1

Dann ist $M = \{0,1\}^4$, und es gilt: $A^{0110}(1101) = 1$ und $A^{0010}(1101) = 0$. Kompakt: Wenn $b_0b_1b_2b_3$ mit $i$ Nullen und einer 1 (an Position $i$) beginnt, dann ist die Ausgabe $x(i)$, für $i = 0, 1, 2, 3$. Ist $b_0b_1b_2b_3 = 0000$, so ist die Ausgabe 1. Also gilt:

$$\Pr(A(1101) = 0) = \Pr(\{w \in \{0,1\}^4 \mid w_0 = w_1 = 0, w_2 = 1\}) = \left(\frac{1}{2}\right)^3 = \frac{1}{8},$$

und

$$\Pr(b_1 = 1) = \Pr(\{w \in \{0,1\}^4 \mid w(1) = 1\}) = \frac{1}{2}$$

Damit erhalten wir auch sinnvolle kombinierte und bedingte Wahrscheinlichkeiten:

$$\Pr(A(1101) = 0, b_0 = 0) = \Pr(\{w \in \{0,1\}^4 \mid w_0 = w_1 = 0, w_2 = 1\}) = \frac{1}{8}$$
$$\Pr(A(1101) = 0 \mid b_0 = 0) = \frac{\Pr(A(1101) = 0, b_0 = 0)}{\Pr(b_0 = 0)} = \frac{1}{4}$$

Wir können auch den Algorithmus $A$ so ändern, dass eine Anweisung $y_i \leftarrow \mathrm{flip}(M_i)$ durch $y_i \leftarrow m_i^{(0)}$ ersetzt wird. (Diesen Algorithmus bezeichnen wir dann mit $A\langle y_i = m_i^{(0)}\rangle$.) Es gilt dann für alle Eingaben $x$ und Ausgaben $z$:

$$\Pr(A(x) = z \mid y_i = m_i^{(0)}) = \Pr(A\langle y_i = m_i^{(0)}\rangle(x) = z)$$

Die verallgemeinerte Notation $A\langle y_1 = m_1^{(0)}, \ldots, y_s = m_s^{(0)}\rangle$ erklärt sich von selbst.

### 2.3.3 Prozeduren als Parameter

Wir werden Algorithmen $A$ betrachten, die als Eingabe eine Prozedur $B$ (z. B. die Verschlüsselungsfunktion einer Blockchiffre mit fest eingesetzemem Schlüssel) erhalten. Diese Prozedur darf nur aufgerufen werden, sie wird nicht als Text in den Rumpf von $A$ eingefügt. Sie könnte aber wiederum zufallsgesteuert sein. Um den Wahrscheinlichkeitsraum von $A$ mit einem solchen Funktionsparameter zu bestimmen, müssen folgende Informationen gegeben sein:

- $B$,
- die Anzahl der Aufrufe von $B$ in $A$,
- welche Aufrufe $y \leftarrow \texttt{flip}(M)$ in $B$ vorkommen.

Wir behandeln dann die Variablen $y$ in verschiedenen Aufrufen von $B$ genau wie die aus einem gewöhnlichen randomisierten Programm (Umbenennen der Variablen, Herausziehen der Zufallsexperimente an den Anfang). In dem resultierenden Wahrscheinlichkeitsraum sind dann die Zufallsexperimente in den verschiedenen Aufrufen von $B$ und die in Teilen von $A$ außerhalb von $B$ unabhängig.

## 2.4 Sicherheit von Block-Kryptosystemen

Wir betrachten hier *$\ell$-Block-Kryptosysteme*[26] $\mathcal{B} = (X, K, Y, e, d)$ mit $X = Y = \{0,1\}^\ell$ und $K \subseteq \{0,1\}^s$ für ein $s$. $e$ ist die Verschlüsselungsfunktion und $d$ die Entschlüsselungsfunktion. Wir erinnern uns:

1. Im Szenario 2 (chosen-plaintext attack, CPA) kann die Angreiferin Eva sich eine „geringe Zahl“ von Klartexten verschlüsseln lassen, also hat sie eine Liste von Klartext-Chiffretext-Paaren:
$$(x_1, y_1), \ldots, (x_r, y_r). \tag{2.4}$$
Nun kann jedenfalls nur ein „neuer“ Klartext $x$, also ein $x \in X \setminus \{x_1, \ldots, x_r\}$, geheim übertragen werden. Die possibilistische Sicherheit verlangt genau, dass dies möglich ist: Wenn $y \in Y \setminus \{y_1, \ldots, y_r\}$ ein neuer gegebener Chiffretext ist, dann gibt es für jeden Klartext $x \in X \setminus \{x_1, \ldots, x_r\}$ einen Schlüssel $k$, der $x_i$ zu $y_i$ chiffriert, $1 \leq i \leq r$, und $x$ zu $y$ chiffriert.

2. Wenn dabei $r$ beliebig groß sein darf, können nach Prop. 2.3 nur Substitutionskryptosysteme in diesem Sinne sicher sein. Da sie $|X|!$ Schlüssel haben müssen und daher immense Schlüssellänge (mindestens $|X| \log |X| - O(|X|)$ Bits) erfordern, kommen sie in der Praxis nicht in Frage.

Idee eines *neuen Sicherheitsbegriffs* (für Block-Kryptosysteme): Gegeben sei eine Angreiferin Eva mit beschränkten Berechnungsressourcen. Wir fragen, wie sehr aus Evas Sicht das $\ell$-Block-Kryptosystem $\mathcal{B} = (\{0,1\}^\ell, K, \{0,1\}^\ell, e, d)$ dem *Substitutionskryptosystem* $\mathcal{S}' = (\{0,1\}^\ell, \mathcal{P}_{\{0,1\}^\ell}, \{0,1\}^\ell, e', d')$ (siehe Def. 1.9) ähnelt. Ist es ihr mit „signifikanter Erfolgswahrscheinlichkeit“ möglich, aufgrund der ihr vorliegenden Information (2.4) und im Rahmen ihrer Ressourcen, zu unterscheiden, welches der beiden Systeme verwendet wird? Wenn dies nicht der Fall ist, kann das Kryptosystem $\mathcal{B}$ als sicher gelten, wie die folgende Überlegung zeigt.

- Wenn Eva aufgrund der ihr vorliegenden Information (2.4) nicht mit nennenswerter Erfolgswahrscheinlichkeit unterscheiden kann, ob sie es mit der Chiffre $e(\,.\,, k)$ (für ein zufälliges $k$ in $K$) oder mit $e'(\,.\,, \pi)$ (für ein zufälliges $\pi \in \mathcal{P}_{\{0,1\}^\ell}$) zu tun hat, dann hat sie aus der ihr vorliegenden Information keine nennenswerte Information über die konkrete Chiffre $e(\,.\,, k)$ gelernt. Insbesondere kann sich ihre Information über die Klartextverteilung nicht (wesentlich) ändern, wenn ihr ein neuer Chiffretext $y$ vorgelegt wird, da bei $\mathcal{S}' = (\{0,1\}^\ell, \mathcal{P}_{\{0,1\}^\ell}, \{0,1\}^\ell, e', d')$ keine solche Änderung eintritt.

[26] Klartexte und Chiffretexte sind Blöcke von Bits gleicher Länge $\ell$ und die Schlüsselmenge besteht aus $s$-Bit-Strings für ein $s$.

Wir modellieren Verfahren, die eine Chiffre benutzen dürfen und dann „raten“ sollen, ob es eine Chiffre zu einem BKS $\mathcal{B}$ oder eine Zufallspermutation ist, als randomisierte Algorithmen.

**Definition 2.9** *Ein* $\ell$-Unterscheider *ist ein randomisierter Algorithmus* $U(F\colon \{0,1\}^\ell \to \{0,1\}^\ell)\colon \{0,1\}$*, dessen Laufzeit bzw. Ressourcenaufwand durch eine Konstante beschränkt ist.*

Das Argument des $\ell$-Unterscheiders ist also eine Chiffre $F$. Diese ist als „Orakel“ gegeben, das heißt als Prozedur, die nur ausgewertet werden kann, deren Programmtext $U$ aber nicht kennt. Das Programm $U$ kann $F$ endlich oft aufrufen, um sich Paare wie in (2.4) zu besorgen. Danach kann $U$ noch weiter rechnen, um zu einer Entscheidung zu kommen. Das von $U$ gelieferte Ergebnis ist ein Bit.

Am liebsten hätte man folgendes Verhalten, für ein gegebenes Block-Kryptosystem $\mathcal{B}$: Programm $U$ sollte 1 liefern, wenn $F$ eine Chiffre $e(\,.\,,k)$ zu $\mathcal{B}$ ist, und 0, wenn $F = \pi$ für eine Permutation $\pi \in \mathcal{P}_{\{0,1\}^\ell}$ ist, die keine $\mathcal{B}$-Chiffre ist. Das gewünschte Verhalten wird sich bei $U$ natürlich niemals mit absoluter Sicherheit, sondern nur mit mehr oder weniger großer Wahrscheinlichkeit einstellen, je nach Situation.

**Beispiel 2.10** Als Beispiel betrachten wir das Vernam-Kryptosystem $\mathcal{B} = \mathcal{B}_{\text{Vernam}}$, siehe Beispiel 1.6. Wir definieren einen $\ell$-Unterscheider $U = U_{\text{Vernam}}$, der als Parameter eine Funktion $F\colon \{0,1\}^\ell \to \{0,1\}^\ell$ erhält und Folgendes tut:

1. $k \leftarrow F(0^\ell)$
2. $y \leftarrow F(1^\ell)$
3. falls $1^\ell \oplus_\ell k = y$, dann gib 1 aus, sonst 0.

Dieser Unterscheider benutzt keine Zufallsexperimente, obwohl es ihm erlaubt wäre. Man sieht leicht Folgendes: Wenn $F(\,.\,) = e(\,.\,,k)$ für die Vernam-Chiffre mit beliebigem Schlüssel $k$, liefert $U_{\text{Vernam}}$ immer das Ergebnis 1. Wenn hingegen $F$ eine zufällige Permutation $\pi$ von $\{0,1\}^\ell$ ist, dann ist die Wahrscheinlichkeit, dass $F(1^\ell) = 1^\ell \oplus_\ell F(0^\ell)$ gilt, genau $\frac{1}{2^\ell - 1}$, also wird die Ausgabe 1 nur mit sehr kleiner Wahrscheinlichkeit ausgegeben (wenn $\ell$ nicht ganz klein ist).

Wir definieren nun ein *Spiel* („*game*“), mit dem ein beliebiges Block-Kryptosystem $\mathcal{B}$ und ein beliebiger Unterscheider $U$ darauf getestet werden, ob $\mathcal{B}$ gegenüber $U$ „anfällig“ ist oder nicht. (Das Spiel ist ein Gedankenexperiment, es ist nicht algorithmisch.) Die Idee ist folgende. Man entscheidet mit einem Münzwurf (Zufallsbit $b$), ob $U$ für seine Untersuchungen als $F(\,.\,)$ eine zufällige Chiffre $e(\,.\,,k)$ von $\mathcal{B}$ („Realwelt“) oder eine zufällige Permutation $\pi$ von $\{0,1\}^\ell$ („Idealwelt“) erhalten soll. Dann rechnet $U$ mit $F$ als Orakel und gibt dann

seine Meinung ab, ob er sich in der Realwelt oder in der Idealwelt befindet. $U$ „gewinnt“, wenn diese Meinung zutrifft.

**Definition 2.11** *Sei $\mathcal{B} = (\{0,1\}^\ell, K, \{0,1\}^\ell, e, d)$ ein $\ell$-Block-Kryptosystem, und sei $U$ ein Unterscheider. Das zugehörige* Experiment *(oder* Spiel*) ist der folgende Algorithmus:*

$G_U^{\mathcal{B}}()\colon \{0,1\}$ // kein Argument, Ausgabe ist ein Bit

1. $b \leftarrow \mathrm{flip}(\{0,1\})$
   **if** $b = 1$ („Realwelt“) **then** $k \leftarrow \mathrm{flip}(K)$; $F \leftarrow e(\,.\,,k)$ // Zufallschiffre von $\mathcal{B}$
   **if** $b = 0$ („Idealwelt“) **then** $F \leftarrow \mathrm{flip}(\mathcal{P}_{\{0,1\}^\ell})$ // Zufallspermutation
2. $b' \leftarrow U(F)$

   // Der $\ell$-Unterscheider versucht herauszubekommen, ob $b = 0$ oder $b = 1$ gilt.
3. **if** $b = b'$ **then return** 1 **else return** 0.

   // 1 heißt, dass $U$ das richtige Ergebnis hat.

*Das* verkürzte Experiment/Spiel $S_U^{\mathcal{B}}$ *(„*short*“) gibt im 3. Schritt einfach $b'$ aus.*

Der Wahrscheinlichkeitsraum für die Analyse des Spiels wird über die Zufallsexperimente zur Wahl von $b$, von $k$, der Zufallspermutation $\pi$ aus $\mathcal{P}_{\{0,1\}^\ell}$ und die Zufallsexperimente in $U$ gebildet. Die Wahrscheinlichkeit dafür, dass der Unterscheider richtig liegt, ist $\Pr(G_U^{\mathcal{B}} = 1)$, gemessen in diesem etwas verschachtelten Wahrscheinlichkeitsraum.

Wir erläutern intuitiv[27], wieso diese Definition eine sehr allgemeine Situation erfasst. Angenommen, Angreiferin Eva kann auf irgendeine algorithmische Weise mit einer gewissen positiven Wahrscheinlichkeit eine „nichttriviale Information“ über die Klartexte gewinnen, wenn diese mittels einer zufälligen Chiffre des $\ell$-Block-Kryptosystems $\mathcal{B}$ verschlüsselt worden sind. Damit kann sie einen Unterscheider mit nichttrivialer Erfolgswahrscheinlichkeit bauen! Sie ruft die Verschlüsselungsfunktion $F$ auf, um einige selbstgewählte Klartexte $x_1, \ldots, x_q$ zu $F(x_1) = y_1, \ldots, F(x_q) = y_q$ zu verschlüsseln. Dann berechnet sie aus $y_1, \ldots, y_q$ ihre „nichttriviale Information“ $I_1, \ldots, I_q$ zu den entsprechenden Klartexten und prüft, ob $I_r$ auf $x_r$ zutrifft, für $r = 1, \ldots, q$. Gibt es eine gute Übereinstimmung, so geht sie davon aus, dass $F$ aus der „Realwelt“ stammt, also eine Chiffre von $\mathcal{B}$ ist. Ansonsten vermutet sie, dass $F$ aus der „Idealwelt“ stammt, also eine zufällige Permutation ist.

**Beispiel 2.12** Für einen $\ell$-Bit-String $z$ sei $p(z) = \bigoplus_{1 \le i \le \ell} z_i$ seine Parität. Nehmen wir an, das Chiffrierverfahren von $\mathcal{B}$ ist unvorsichtig geplant und zwar so, dass $p(e(x,k)) = p(x)$ ist für beliebige $x \in X$ und $k \in K$. Bei der Chiffrierung ändert sich also die Parität nicht. (Die Parität ist sicher „nichttriviale Information“, auch wenn sie vielleicht nicht unmittelbar nützlich ist.)

[27] Im Buch von Küsters und Wilke wird diese Idee in Beispiel 4.7.2 genauer ausgearbeitet. Interessierte mögen dort nachlesen!

$U_{\text{Parität}}(F)$:

1. Wähle Klartexte $x_1, \ldots, x_q$ mit $p(x_1) = \cdots = p(x_q) = 0$.
2. $y_r \leftarrow F(x_r)$, für $r = 1, \ldots, q$.
3. falls $p(y_1) = \cdots = p(y_q) = 0$, dann gib 1 aus, sonst 0.

Wenn wir in der „Realwelt“ sind ($b = 1$), also $F$ eine Chiffre $e(\,.\,, k)$ ist, dann ist die Antwort von $U$ immer „1“, also korrekt. Wenn wir in der „Idealwelt“ sind ($b = 0$), also $F$ eine zufällige Permutation ist, dann ist die Antwort nur mit Wahrscheinlichkeit

$$\frac{2^{\ell-1}(2^{\ell-1}-1)(2^{\ell-1}-2)\ldots(2^{\ell-1}-q+1)}{2^{\ell}(2^{\ell}-1)(2^{\ell}-2)\ldots(2^{\ell}-q+1)} \leq \frac{1}{2^q}$$

falsch. (Alle Werte $F(x_1), \ldots, F(x_q)$ müssen zufällig zu Chiffretexten geführt haben, die Parität 0 haben, von denen es $2^{\ell-1}$ viele gibt.) Es ergibt sich $\Pr(G_U^{\mathcal{B}} = 1) \geq \frac{1}{2}(1+1-2^{-q}) = 1 - 2^{-(q+1)}$. Mit der effizienten Berechenbarkeit der „nichttrivialen Information“ hat man also einen Unterscheider gefunden, der $\Pr(G_U^{\mathcal{B}} = 1)$ „groß“ macht.

**Beispiel:** Der folgende triviale $\ell$-Unterscheider $U_{\text{trivial}}$ erreicht $\Pr(G_{U_{\text{trivial}}}^{\mathcal{B}} = 1) = \frac{1}{2}$:

$$b \leftarrow \text{flip}(\{0,1\}); \text{return } b.$$

Daher interessieren wir uns nicht für die Wahrscheinlichkeit $\Pr(G_U^{\mathcal{B}} = 1)$ an sich, sondern für den Abstand von $\Pr(G_U^{\mathcal{B}} = 1)$ zu $\Pr(G_{U_{\text{trivial}}}^{\mathcal{B}} = 1) = \frac{1}{2}$:

**Definition 2.13** *Sei $U$ ein $\ell$-Unterscheider und $\mathcal{B}$ ein $\ell$-Block-KS. Der* Vorteil *von $U$ bzgl. $\mathcal{B}$ ist*[28]

$$\text{adv}(U, \mathcal{B}) := 2\left(\Pr(G_U^{\mathcal{B}} = 1) - \frac{1}{2}\right).$$

Klar ist:

- Für jeden $\ell$-Unterscheider $U$ und jedes $\ell$-Block-KS $\mathcal{B}$ gilt $-1 \leq \text{adv}(U, \mathcal{B}) \leq 1$.
- Werte $\text{adv}(U, \mathcal{B}) < 0$ sind uninteressant. (Wenn man einen Unterscheider $U$ mit $\text{adv}(U, \mathcal{B}) < 0$ hat, sollte man in $U$ die Ausgaben 0 und 1 vertauschen und erhält einen Unterscheider mit positivem Vorteil.)
- Für den trivialen $\ell$-Unterscheider $U_{\text{trivial}}$ gilt $\text{adv}(U, \mathcal{B}) = 0$.

[28] „adv“ für „advantage“.

Um den Vorteil eines Unterscheiders auszurechnen, sind seine „Erfolgswahrscheinlichkeit“ und seine „Misserfolgswahrscheinlichkeit“ hilfreiche Werte: Der *Erfolg* von $U$ bzgl. $\mathcal{B}$ ist (für das verkürzte Spiel $S = S_U^{\mathcal{B}}$)

$$\mathrm{suc}(U, \mathcal{B}) := \Pr(S\langle b = 1\rangle = 1)\,,$$

d. h. die Wahrscheinlichkeit, dass $U$ die Verwendung des $\ell$-Block-KS $\mathcal{B}$ richtig erkennt. Der *Misserfolg* ist

$$\mathrm{fail}(U, \mathcal{B}) := \mathrm{fail}(U) := \Pr(S\langle b = 0\rangle = 1)\,,$$

d. h. die Wahrscheinlichkeit, dass $U$ die Verwendung des idealen Substitutionskryptosystems *nicht* erkennt. Man kann in der Notation für „fail“ das „$\mathcal{B}$“ auch weglassen, da im Fall $b = 0$ das Kryptosystem $\mathcal{B}$ überhaupt keine Rolle spielt.

**Lemma 2.14** $\mathrm{adv}(U, \mathcal{B}) = \mathrm{suc}(U, \mathcal{B}) - \mathrm{fail}(U)$.

*Beweis*:

$$\begin{aligned}
\Pr(G_U^{\mathcal{B}} = 1) &= \Pr(S_U^{\mathcal{B}} = b) \\
&= \Pr(S_U^{\mathcal{B}} = 1, b = 1) + \Pr(S_U^{\mathcal{B}} = 0, b = 0) \\
&= \Pr(S_U^{\mathcal{B}} = 1 \mid b = 1) \cdot \Pr(b = 1) \\
&\qquad + \Pr(S_U^{\mathcal{B}} = 0 \mid b = 0) \cdot \Pr(b = 0) \\
&= \frac{1}{2}\left(\Pr(S_U^{\mathcal{B}}\langle b = 1\rangle = 1) + \Pr(S_U^{\mathcal{B}}\langle b = 0\rangle = 0)\right) \\
&= \frac{1}{2}\left(\Pr(S_U^{\mathcal{B}}\langle b = 1\rangle = 1) + (1 - \Pr(S_U^{\mathcal{B}}\langle b = 0\rangle = 1))\right) \\
&= \frac{1}{2}\left(\mathrm{suc}(U, \mathcal{B}) + (1 - \mathrm{fail}(U))\right) \\
&= \frac{1}{2}\left(\mathrm{suc}(U, \mathcal{B}) - \mathrm{fail}(U)\right) + \frac{1}{2}.
\end{aligned}$$

Durch Umstellen ergibt sich die Behauptung. □

Unterscheider mit Werten $\mathrm{adv}(U, \mathcal{B})$ substanziell über 0 können als interessant (oder möglicherweise gefährlich) für $\mathcal{B}$ gelten. – Wir müssen noch die beschränkten Ressourcen des Unterscheiders ins Spiel bringen.

**Definition 2.15** *Sei $\ell \in \mathbb{N}$, $U$ ein $\ell$-Unterscheider, $\mathcal{B}$ ein $\ell$-Block-KS. Für $q, t \in \mathbb{N}$ heißt $U$* $(q, t)$-beschränkt*, wenn die Laufzeit des Aufrufs von $U$ innerhalb von $G_U^{\mathcal{B}}$ durch $t$ beschränkt ist und dieser Aufruf die Funktion $F$ mit höchstens $q$ verschiedenen Argumenten ausführt.*

**Beispiel 2.10 (Fortsetzung)** Der $\ell$-Unterscheider $U_{\mathrm{Vernam}}$ wertet die Funktion $F$ an zwei Stellen $0^\ell$ und $1^\ell$ aus. aus. Die Laufzeit des Aufrufs von $U$ ist beschränkt durch $c \cdot \ell$ für eine Konstante $c \geq 1$. Also ist $U_{\mathrm{Vernam}}$ $(2, c \cdot \ell)$-beschränkt.

**Definition 2.16** *Sei $\varepsilon > 0$. Dann heißt $\mathcal{B}$ $(q,t,\varepsilon)$-*sicher*, wenn für jeden $(q,t)$-beschränkten $\ell$-Unterscheider $U$ gilt*

$$\mathrm{adv}(U,\mathcal{B}) < \varepsilon\,.$$

Man kann diese Bedingung auch umgedreht lesen, nämlich so: „Um mit Wahrscheinlichkeit größer als $\frac{1}{2}(1+\varepsilon)$ im Experiment $G_U^{\mathcal{B}}$ die richtige Antwort 1 zu erzeugen, braucht ein $\ell$-Unterscheider mehr als $q$ Auswertungen oder mehr Laufzeit/Platz als $t$.“ Mit beliebig hohen Rechenzeiten lassen sich praktisch alle Block-Kryptosysteme brechen, selbst mit sehr kleinem $q$. (Siehe Beispiel mit $q = 5$ in Abschnitt 2.3.1.)

**Beispiel 2.10 (Fortsetzung)** Sei $\mathcal{B}$ das Vernam-System der Länge $\ell$. Um den Vorteil von $U_{\text{Vernam}}$ bzgl. $\mathcal{B}$ zu bestimmen, berechnen wir Erfolg und Misserfolg von $U$:

Es gilt offensichtlich $1 = \Pr(S_U^{\mathcal{B}}\langle b=1\rangle = 1) = \mathrm{suc}(U,\mathcal{B})$.

Es gilt $\mathrm{fail}(U) = \Pr(S_U^{\mathcal{B}}\langle b=0\rangle = 1) = \Pr(S_U^{\mathcal{B}} = 1 \mid b = 0)$. Das verkürzte Experiment $S_U^{\mathcal{B}}$ gibt 1 aus, wenn der Unterscheider $U$ dies tut. Dieser tut es aber genau dann, wenn $1^\ell \oplus_\ell F(0^\ell) = F(1^\ell)$ gilt, d. h.

$$\mathrm{fail}(U) = \Pr(S_U^{\mathcal{B}}\langle b=0\rangle = 1) = \Pr(1^\ell \oplus_\ell F(0^\ell) = F(1^\ell)) = \frac{1}{2^\ell - 1}.$$

Damit haben wir aber

$$\mathrm{adv}(U,\mathcal{B}) = \mathrm{suc}(U,\mathcal{B}) - \mathrm{fail}(U) = 1 - \frac{1}{2^\ell-1} = \frac{2^\ell-2}{2^\ell-1} \approx 1\,.$$

Das Vernam-System der Länge $\ell$ ist also nicht $(2, c\cdot\ell, \frac{2^\ell-2}{2^\ell-1})$-sicher. Da für realistische Werte von $\ell$ (z. B. 128) der Wert $\frac{2^\ell-2}{2^\ell-1}$ praktisch 1 ist, muss das Vernam-System im Szenario 2 als unsicher angesehen werden.

Natürlich sieht man auch mit bloßem Auge, dass die Vernam-Chiffre bei mehrfacher Verwendung nicht „sicher“ ist, da sie leicht entschlüsselt werden kann. Hier geht es aber um die Formulierung eines Sicherheitskonzepts, das allgemein anwendbar ist. Es ist beruhigend, dass im Fall der Vernam-Chiffre herauskommt, dass sie auch im Sinn dieser Definition unsicher ist.

Am obigen Beispiel eines Block-Kryptosystems, das die Parität von $x$ auf den Chiffretext $e(x,k)$ überträgt, sieht man aber auch, dass das Unterscheiderkonzept sehr empfindlich ist: Obwohl die Information über die Parity klein und harmlos scheint, erklärt es das System für nicht $(q, O(q\ell), 1-2^{-q})$-sicher, weil die Auswertung der Parität nur Zeit $O(\ell)$ kostet.

**Beispiel 2.17** Es sei $\mathcal{B}$ ein $\ell$-Block-Kryptosystem. (Man denke an AES.) Wir nehmen (realistisch) an, dass $t$ (etwa $t = 6$) Klartext-Chiffretext-Paare $(x_1, y_1), \ldots, (x_t, y_t)$ in $\mathcal{B}$ den Schlüssel $k$ der Länge $s = \ell$ eindeutig bestimmen, für mindestens die Hälfte der Schlüssel $(*)$.

Der $\ell$-Unterscheider $U_{\text{brute-force}}$ wertet die Funktion $F$ an $t+1$ Stellen $x_1, \ldots, x_t, x$ aus, mit Ergebnissen $y_1, \ldots, y_t, y$. Er probiert alle $2^\ell$ Schlüssel $k$, um einen zu finden, der $e(x_1, k) = y_1, \ldots, e(x_t, k) = y_t$ erfüllt. Wenn kein solches $k$ gefunden wird, gibt $U$ den Wert 0 aus. Sonst wird getestet, ob $e(x, k) = y$ ist. Falls ja, wird 1 ausgegeben, sonst 0.

Das Verhalten lässt sich so beschreiben: In der „Realwelt“, wenn $F = e(\,.\,, k)$ ist für einen zufälligen Schlüssel $k$, dann findet $U$ dieses $k$ mit Wahrscheinlichkeit größer als $\frac{1}{2}$ (wegen $(*)$). Dann ist $e(x, k) = F(x)$ nach Wahl von $F$ und $y = F(x)$, weil $y$ so bestimmt wurde. Also ist $e(x, k) = y$, also gibt $U$ den Wert 1 aus. – In der „Idealwelt“ wird entweder kein passendes $k$ gefunden, oder wenn doch, dann ist die Wahrscheinlichkeit, dass $e(x, k) = y$ ist, wo $y$ ein zufälliger Wert in $\{0,1\}^\ell - \{y_1, \ldots, y_t\}$ ist, durch $1/(2^\ell - t)$ beschränkt. Damit ist

$$\text{adv}(U, \mathcal{B}) = \text{suc}(U, \mathcal{B}) - \text{fail}(U) \geq \frac{1}{2} - \frac{1}{2^\ell - t} \approx \frac{1}{2}.$$

Die Laufzeit des Aufrufs von $U$ ist beschränkt durch $O(|K|) = O(2^\ell)$.

Daher ist $\mathcal{B}$ nicht $(7, O(2^\ell), \frac{1}{2})$-sicher. Das ist natürlich nicht sehr schlimm, weil die Rechenzeit von $U$ unrealistisch hoch ist.[29]

**Schlussbemerkung:** Gesucht wäre nun natürlich ein $\ell$-Block-Kryptosystem, das $(q, t, \varepsilon)$-sicher ist, wobei $q$ und $t$ einigermaßen groß sind und $\varepsilon$ möglichst klein ist. Es gibt unter bestimmten Annahmen („Existenz von Einwegfunktionen“) theoretisch konstruierbare Systeme, die für große $\ell$ einigermaßen effiziente („polynomiell in $\ell$“) Verschlüsselung und Entschlüsselung erlauben und im definierten Sinn für Angreifer mit (im Bezug zu $\ell$) nicht zu großen Ressourcen („polynomiell in $\ell$“) $\varepsilon$-sicher sind, für recht kleine $\varepsilon$ (der Wert $1/\varepsilon$ darf polynomiell groß sein). Diese Systeme sind aber praktisch nicht verwendbar. Von praktisch relevanten Systemen wie AES weiß man nicht, für welche Werte sie sicher sind.

[29] Wenn man in der Literatur (oder der Presse) von erfolgreichen Angriffen auf Block-Kryptosysteme liest, ist meistens ein Angriff im Sinn eines Unterscheiders gemeint, wobei es meist schon als Erfolg gilt, die Anzahl der zu untersuchenden Schlüssel deutlich unter $|K|$ zu drücken.