

Technische Universität Darmstadt  
Fachbereich Informatik  
Lehrstuhl für Kryptografie und Computeralgebra



Diplomarbeit

# **T-Funktionsbasierte Blockchiffren**

Elmar Wolfgang Tischhauser

29. Juli 2007

Betreuer:

Prof. Dr. Johannes Buchmann,  
Dipl.-Inform. Kai Wirt



## **Danksagung**

Mein Dank gilt Prof. Dr. Johannes Buchmann, der diese Diplomarbeit ermöglicht hat, sowie meinem Betreuer Kai Wirt für seine Ideen und Anregungen, die stets sehr hilfreich gewesen sind.

Meine Eltern haben mich während meines Studiums in jeder nur erdenklichen Weise unterstützt. Dafür danke ich ihnen ganz herzlich. Ein besonderes Dankeschön geht auch an meine Schwester Gundula für ihr scharfes Auge beim Korrekturlesen. Dank schulde ich schließlich allen Freunden, welche mein Studium anregend begleitet und um viele interessante Diskussionen bereichert haben.

# Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>9</b>
1.1	Motivation . . . . .	9
1.2	Struktur der Arbeit . . . . .	9
<b>2</b>	<b>Blockchiffren</b>	<b>11</b>
2.1	Grundlegendes . . . . .	11
2.1.1	Verschlüsselung . . . . .	14
2.1.2	Betriebsmodi . . . . .	15
2.1.3	Iterierte Blockchiffren . . . . .	17
2.2	Konstruktion iterierter Blockchiffren . . . . .	18
2.2.1	Allgemeine Prinzipien . . . . .	18
2.2.2	Konfusion . . . . .	20
2.2.3	Diffusion . . . . .	20
2.2.4	Feistel-Netze . . . . .	22
2.2.5	Substitutions-Permutations-Netzwerke . . . . .	24
2.3	Kryptoanalyse . . . . .	25
2.3.1	Klassifikation von Angriffen auf Blockchiffren . . . . .	25
2.3.2	Differentielle Kryptoanalyse . . . . .	26
2.3.3	Lineare Kryptoanalyse . . . . .	32
2.4	Linearitätsmaße Boole'scher Abbildungen . . . . .	38
2.4.1	Algebraische Normalform und algebraischer Grad . . . . .	39
2.4.2	Die Walsh-Hadamard-Transformation . . . . .	41
2.4.2.1	Transformation und Faltung . . . . .	41
2.4.2.2	Schnelle Walsh-Transformation . . . . .	43
2.4.3	Walsh-Spektrum, lineare Approximationstabelle und lineares Potenzial . . . . .	47
2.4.4	Differenztabelle und differentielles Potenzial . . . . .	52
<b>3</b>	<b>T-Funktionen</b>	<b>55</b>
3.1	Motivation . . . . .	55
3.2	Definition und Notation . . . . .	57
3.2.1	Parameter und Bit-Slice-Analyse . . . . .	61
3.2.2	Breite T-Funktionen . . . . .	66
3.3	Invertierbarkeit . . . . .	69
3.3.1	Charakterisierung invertierbarer T-Funktionen . . . . .	69
3.3.2	Testen auf Invertierbarkeit . . . . .	70

3.3.3	Konstruktion invertierbarer T-Funktionen . . . . .	76
3.3.3.1	Ansatz . . . . .	77
3.3.3.2	Parameterkonstruktion . . . . .	77
3.3.3.3	Lineare Bit-Slices mit additiven Parametern . . . . .	82
3.3.3.4	Multiplikative Parameter . . . . .	83
3.3.3.5	Nichtlineare Bit-Slices . . . . .	84
3.3.3.6	Breite T-Funktionen und nichtprimitive Operationen . . . . .	86
<b>4</b>	<b>T-Funktionsbasierte Blockchiffren</b>	<b>89</b>
4.1	T-Funktionen als Konfusionselemente . . . . .	89
4.1.1	Untersuchungsansatz . . . . .	90
4.1.2	Linearitätsmaße . . . . .	91
4.1.3	Funktions- und Parameter-Enumeration . . . . .	93
4.1.4	Differentielle und lineare Eigenschaften . . . . .	97
4.1.4.1	Lineare Bit-Slices und breite T-Funktionen . . . . .	97
4.1.4.2	Nichtlineare Bit-Slices . . . . .	103
4.1.5	Andere Konstruktionsprinzipien . . . . .	114
4.2	T-Funktionen als Diffusionselemente . . . . .	115
4.2.1	MDS-Abbildungen mittels T-Funktionen . . . . .	115
4.2.2	MDS- $m$ -Abbildungen mittels T-Funktionen . . . . .	122
4.2.3	Andere Konstruktionsprinzipien . . . . .	128
4.3	Effizienz . . . . .	128
4.4	Sicherheit . . . . .	129
4.5	T-Funktionsbasierte Feistel-Netze . . . . .	130
4.5.1	Einsatzparameter . . . . .	131
4.5.2	Exemplarische Konstruktion . . . . .	132
4.6	T-Funktionsbasierte SP-Netzwerke . . . . .	136
4.6.1	Einsatzparameter . . . . .	136
4.6.2	Exemplarische Konstruktion . . . . .	136
<b>5</b>	<b>Zusammenfassung</b>	<b>143</b>
5.1	Ergebnis . . . . .	143
5.2	Ausblick . . . . .	145

# Abbildungsverzeichnis

2.1	Verschlüsselung in verschiedenen blockweise operierenden Betriebsmodi. . .	16
2.2	Operationsweise von Substitutions-Boxen. . . . .	20
2.3	Diffusion durch Transposition. . . . .	21
2.4	Rundenfunktion eines Feistel-Netzes. . . . .	23
2.5	Rundenfunktion eines SP-Netzwerks. . . . .	24
2.6	Differentielle 1-Runden-Charakteristiken eines 6-Bit-Feistel-Netzes. . . . .	30
2.7	2-Runden-Charakteristik eines 6-Bit-Feistel-Netzes. . . . .	30
2.8	Exemplarische Durchführung der schnellen Walsh-Transformation. . . . .	47
3.1	Notationsweise und Abhängigkeitsverhalten einer T-Funktion. . . . .	59
3.2	Parametrisierte Darstellung einer T-Funktion. . . . .	63
3.3	Aus unabhängigen Teilausdrücken konstruierte breite T-Funktion. . . . .	67
3.4	Aus abhängigen Teilausdrücken konstruierte breite T-Funktion. . . . .	68
3.5	Nichtprimitive Operationen in breiten T-Funktionen. . . . .	68
4.1	Repräsentation von T-Funktionen in mehreren Veränderlichen als vektorielle Abbildungen. . . . .	91
4.2	Zusammenfassung des Entwurfs- und Suchraums für invertierbare T-Funktionen. . . . .	92
4.3	Illustration der Aussage von Proposition 4.16. . . . .	117
4.4	Differentielle 2-Runden-Charakteristik des Feistel-Netzes $\mathcal{FN}$ . . . . .	134
4.5	Beste ermittelte differentielle 3-Runden-Charakteristik von $\mathcal{SPN}$ . . . . .	140
4.6	Beste ermittelte lineare Approximation der ersten 3 Runden von $\mathcal{SPN}$ . . .	142

# Tabellenverzeichnis

2.1	Beispielhafte Verteilung der Ausgabedifferenzen einer Funktion. . . . .	27
2.2	Vollständige Differenzentabelle einer Funktion. . . . .	28
2.3	Gültigkeit einer lineare Approximation einer nichtlinearen Funktion. . . . .	33
2.4	Lineare Approximationstabelle einer Funktion. . . . .	34
3.1	Konstruktionstechniken für Parameter. . . . .	78
3.2	Konstruktionsrezepte für Parameter mit Null in der LSB . . . . .	80
3.3	Korrekturtechniken für das niederwertigste Bit . . . . .	81
4.1	Untersuchte Kombinationen aus Variablenanzahl und Wortbreite. . . . .	94
4.2	T-Funktionen in 3 Variablen: Differentielle und lineare Eigenschaften. . .	103
4.3	Differentielles und lineares Potenzial von T-Funktionen mit nichtlinearen Bit-Slices in 3 Variablen. . . . .	106
4.4	Differenzentabelle einer Bit-Slice von $f_{12465}$ . . . . .	107
4.5	Untere Schranken für differentielles und lineares Potenzial einer T-Funktion mit nichtlinearen Bit-Slices in $k$ Variablen. . . . .	112
4.6	Beste experimentell ermittelte Permutationen für $k = 4, 8$ . . . . .	113
4.7	Differentielle Potenziale von $x \wedge (x - 1)^2$ für verschiedene $n$ . . . . .	114
4.8	Konstruktionsparameter für MDS-Diffusionsebenen in Feistel-Netzen. . .	122
4.9	Konstruktionsparameter für MDS-Diffusionsebenen in SP-Netzwerken. . .	123
4.10	Vollständige Übersicht der MDS- $m$ -Matrizen für $m = 2, 3, 4$ . . . . .	127
4.11	Parameterkombinationen und Rundenzahlen für T-Funktionsbasierte Feistel- Netze. . . . .	132
4.12	Parameterkombinationen und Rundenzahlen für T-Funktionsbasierte SP- Netzwerke. . . . .	138

# Algorithmenverzeichnis

2.1	Differentieller 1R-Angriff . . . . .	31
2.2	Auf einer $r - 1$ -Runden-Approximation basierender linearer Angriff . . .	38
2.3	In-place arbeitende schnelle Walsh-Transformation . . . . .	46
4.1	Berechnung der Differenzentabelle einer Boole'schen Abbildung. . . . .	92
4.2	Effiziente Berechnung der linearen Approximationstabelle. . . . .	93
4.3	Aufzählung parametrisierter $k \times k$ -Matrizen. . . . .	95
4.4	NUMBER-TO-PERMUTATION . . . . .	96
4.5	Aufzählung von Permutationen . . . . .	97
4.6	Test auf Parametereigenschaft der LSB. . . . .	98
4.7	T-Funktionsbasiertes 16-Bit-Feistelnetz $\mathcal{FN}$ . . . . .	133
4.8	T-Funktionsbasiertes 24-Bit-SP-Netzwerk $\mathcal{SPN}$ . . . . .	137

# Kapitel 1

## Einführung

### 1.1 Motivation

Die Klasse der Abbildungen von  $k$   $n$ -Bit-Worten auf  $l$   $n$ -Bit-Worte, bei denen das  $i$ -te Bit der Ausgabeworte nur von den ersten  $i$  Bits der Eingabeworte abhängig ist (so genannte *T-Funktionen*), wurde vor wenigen Jahren von Klimov und Shamir eingeführt [15] und auf verschiedene Eigenschaften hin untersucht, unter anderem Invertierbarkeit, Zyklenstruktur und *Maximum Distance Separability* (MDS) [16, 17, 18].

Die meisten gängigen Mikroprozessorinstruktionen und deren Komposition sind T-Funktionen. Wird die Länge  $n$  der einzelnen Variablen nach der natürlichen Wortbreite des Prozessors gewählt (etwa  $n = 8, n = 32$  oder  $n = 64$ ), so lassen sich viele T-Funktionen in wenigen Instruktionen und damit sehr schnell berechnen. Da mit Hilfe von T-Funktionen außerdem leicht Operationen aus verschiedenen algebraischen Strukturen kombiniert werden können, wurden sie als kryptografische Primitive zur Konstruktion von Pseudozufallszahlengeneratoren, Stromchiffren und MDS-Abbildungen vorgeschlagen.

Die Evaluation des Einsatzes von T-Funktionen in *Blockchiffren* wird in [14] zwar angeregt, aber nicht weiter verfolgt. Auch in aktuellen Entwürfen von Blockchiffren sind über das Zufällige hinausgehende Verwendungen von T-Funktionen selten. So wurde bereits 1998 in der Blockchiffre RC6, einem AES-Finalisten, die T-Funktion  $f(x) = 2x^2 + x$  als Bestandteil der Rundenfunktion verwendet [33], allerdings zusammen mit datenabhängigen zyklischen Rotationen, welche keine T-Funktionen sind.

Diese Diplomarbeit soll nun untersuchen, inwieweit sich T-Funktionen beim Entwurf der Konfusions- und Diffusionsebenen iterierter Blockchiffren verwenden lassen. Die Eignung von T-Funktionen für die (üblicherweise mittels Substitutions-Boxen realisierte) Konfusions-ebene wird dabei anhand ihrer Resistenz gegen differentielle und lineare Kryptoanalyse bewertet [1, 27], während die Diffusionseigenschaften von T-Funktionen anhand der MDS-Eigenschaft sowie der von Daemen [8] eingeführten *Branch Number* beurteilt werden.

### 1.2 Struktur der Arbeit

Die Arbeit ist folgendermaßen aufgebaut:

**Kapitel 2** stellt Grundlagen zu den verwendeten mathematischen Strukturen zusammen und gibt eine Einführung in Blockchiffren, ihre Konstruktion und ihre Kryptoanalyse. Dabei liegt ein besonderer Fokus auf den Linearitätsmaßen der differentiellen und linearen Kryptoanalyse sowie deren Behandlung mit Hilfe der Walsh-Hadamard-Transformation.

**Kapitel 3** behandelt T-Funktionen und deren grundlegende Formen und Eigenschaften, wobei insbesondere die von Klimov und Shamir entwickelte Bit-Slice-Analysetechnik im Mittelpunkt steht. Anschließend wird beschrieben, wie jene zur Untersuchung der Invertierbarkeit von T-Funktionen sowie zur systematischen Konstruktion invertierbarer T-Funktionen genutzt werden kann.

**In Kapitel 4** werden die verwendeten Ansätze zur Untersuchung bzw. Konstruktion von T-Funktionen zum Einsatz in Blockchiffren erläutert und die experimentellen Untersuchungsergebnisse vorgestellt. Viele davon lassen sich darüber hinaus analytisch bestätigen, wobei einige neue Resultate zu den Nichtlinearitätseigenschaften von T-Funktionen formuliert werden können.

**Kapitel 5** schließlich fasst die Ergebnisse dieser Arbeit zusammen und gibt einen Ausblick auf offene Probleme und weiterführende Fragestellungen.

# Kapitel 2

## Blockchiffren

### 2.1 Grundlegendes

Die meisten in dieser Arbeit behandelten kryptografischen Primitive und Algorithmen, insbesondere auch Blockchiffren, operieren auf Bitvektoren. Folglich sollen zunächst der Menge  $\{0, 1\}$  und ihrem  $n$ -fachen kartesischen Produkt  $\{0, 1\}^n$  algebraische Struktur verliehen und die auf ihnen üblichen Operationen erklärt werden.

**Definition 2.1** (Restklassenring modulo  $m$ ). Sei  $m$  eine natürliche Zahl. Die Äquivalenzklassen der Kongruenz modulo  $m$

$$a + m\mathbb{Z} := \{b \mid b \equiv a \pmod{m}\}$$

bilden den Ring  $(\mathbb{Z}/m\mathbb{Z}, +, \cdot, 0 + m\mathbb{Z}, 1 + m\mathbb{Z})$ . Dieser wird Restklassenring modulo  $m$  genannt.  $\square$

Für bestimmte Zahlen  $m$  ist  $\mathbb{Z}/m\mathbb{Z}$  sogar ein Körper. Diese lassen sich wie folgt charakterisieren:

**Satz 2.2.** *Der Restklassenring  $\mathbb{Z}/m\mathbb{Z}$  ist genau dann ein Körper, wenn  $m$  eine Primzahl ist.*

*Beweis.* Siehe [5], Theorem 2.6.4.  $\square$

Zu jedem Restklassenkörper gibt es einen dazu isomorphen Körper, dessen Elemente aus den kleinsten nichtnegativen Repräsentanten der Äquivalenzklassen  $a + m\mathbb{Z}$  bestehen:

**Definition 2.3** (Galoiskörper von Primzahlordnung, [25]). Für eine Primzahl  $p$  sei  $\mathbb{F}_p$  die Menge  $\{0, 1, \dots, p-1\}$  ganzer Zahlen sowie die Abbildung  $\Phi : \mathbb{Z}/p\mathbb{Z} \rightarrow \mathbb{F}_p$  definiert durch  $\Phi(a + p\mathbb{Z}) = a$  für  $a = 0, \dots, p-1$ . Dann ist  $\mathbb{F}_p$  mit der durch  $\Phi$  induzierten Körperstruktur ein endlicher Körper, der **Galoiskörper der Ordnung  $p$** .  $\square$

*Bemerkung 2.4.* Anstatt  $\mathbb{F}_p$  schreibt man auch  $GF(p)$  (vom englischen *Galois field*).

Die Abbildung  $\Phi$  ist dabei ein Isomorphismus, der die Struktur des Restklassenkörpers  $\mathbb{Z}/p\mathbb{Z}$  auf  $\mathbb{F}_p$  überträgt. Null- und Einselement von  $\mathbb{F}_p$  sind folglich  $\Phi(0 + p\mathbb{Z}) = 0$  beziehungsweise  $\Phi(1 + p\mathbb{Z}) = 1$ , und das Rechnen in  $\mathbb{F}_p$  bedeutet nichts anderes als die gewohnte Ganzzahlarithmetik mit Reduktion modulo  $p$ .

Von besonderem Interesse ist nun der Spezialfall  $p = 2$ :

**Beispiel 2.5** ( $\mathbb{F}_2, GF(2)$ ). Der endliche Körper  $\mathbb{F}_2$  besteht aus den zwei Elementen 0 und 1; die binären Operationen  $+$  und  $\cdot$  werden durch die folgenden Tabellen erklärt:

$$\begin{array}{c|cc} + & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 1 & 0 \end{array} \qquad \begin{array}{c|cc} \cdot & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 0 & 1 \end{array}$$

□

Als **Bit** bezeichnen wir nun ein Element von  $\mathbb{F}_2$ . Man sieht sofort, dass Addition und Multiplikation in  $\mathbb{F}_2$  exakt den aus der Logik bekannten Junktoren  $\oplus$  (exklusives Oder) beziehungsweise  $\wedge$  (logisches Und) entsprechen. Wenn später im Zusammenhang mit T-Funktionen bitorientierte Abbildungen dargestellt werden, verwenden wir zur besseren Unterscheidung zur arithmetischen Addition das  $\oplus$ -Symbol für die Additionsoperation in  $\mathbb{F}_2$ .

Analog verstehen wir unter einem  $n$ -elementigen **Bitvektor** oder einem  $n$ -bit-Wort  $(x_{n-1}, \dots, x_0)$  ein Element aus  $\mathbb{F}_2^n$ , dem  $n$ -dimensionalen Vektorraum über  $\mathbb{F}_2$ , in welchem Addition und skalare Multiplikation wie üblich komponentenweise erklärt sind. Auch hier wird aus Gründen der Klarheit für die Vektorraumaddition das  $\oplus$ -Symbol verwendet; für das Skalarprodukt zweier Vektoren des  $\mathbb{F}_2^n$  schreiben wir  $a \bullet b := \bigoplus_{i=0}^{n-1} a_i b_i$ .

Zusätzlich identifizieren wir mit dem Bitvektor  $(x_{n-1}, \dots, x_0)$  die natürliche Zahl  $x$  vermöge der Vorschrift  $x = \sum_{i=0}^{n-1} x_i \cdot 2^i$  und erklären dadurch die Anwendung arithmetischer Operationen modulo  $2^n$  auf Bitvektoren.

Für auf Bitvektoren operierende (möglicherweise vektorwertige) Funktionen vereinbaren wir

**Definition 2.6** (Boole'sche Funktion, Boole'sche Abbildung). Eine funktionale Abbildung

$$f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$$

heißt Boole'sche Funktion, ist sie vektorwertig:

$$f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q$$

nennen wir sie eine Boole'sche Abbildung. □

Den **Polynomring** über  $\mathbb{F}_p$ , also den Ring der Polynome in einer Unbestimmten  $X$  mit Koeffizienten aus  $\mathbb{F}_p$ , bezeichnen wir mit  $\mathbb{F}_p[X]$ . Der **Grad** eines Polynoms  $p$  wird mit  $\deg(p)$  bezeichnet. Ein Polynom  $f$  teilt  $g$ , wenn es ein  $q \in \mathbb{F}_p[X]$  gibt mit  $g = qf$ . Besitzt ein Polynom keine nichttriviale Zerlegung in Faktorpolynome, nennen wir es *irreduzibel*:

**Definition 2.7** (Irreduzibles Polynom). Ein Polynom  $f \in \mathbb{F}_p[X]$  heißt *irreduzibel* in  $\mathbb{F}_p[X]$ , wenn  $\deg(f) > 0$ , und aus  $f = gh$  mit  $g, h \in \mathbb{F}_p[X]$  folgt, dass  $\deg(g) = 0$  oder  $\deg(h) = 0$ . □

In  $\mathbb{F}_p[X]$  ist Division mit Rest möglich:

**Satz 2.8** (nach [25], Satz 1.52). Sei  $0 \neq f \in \mathbb{F}_p[X]$ . Dann gibt es für jedes  $g \in \mathbb{F}_p[X]$  eindeutig bestimmte Polynome  $q, r \in \mathbb{F}_p[X]$ , so dass

$$g = qf + r,$$

wobei  $\deg(r) < \deg(f)$ . □

Ist  $f \in \mathbb{F}_p[X]$  ungleich dem Nullpolynom, so entsteht bei Division eines beliebigen Polynoms  $g \in \mathbb{F}_p[X]$  durch  $f$  ein nach Satz 2.8 eindeutiges  $r \in \mathbb{F}_p[X]$  als Divisionsrest. Lassen zwei Polynome  $g, h$  bei Division durch  $f$  den gleichen Rest, nennen wir sie **kongruent modulo  $f$**  und schreiben  $g(X) \equiv h(X) \pmod{f(X)}$ . Analog zu Definition 2.1 ergibt sich

**Definition 2.9** (Restklassenring modulo  $f$ ). Sei  $0 \neq f \in \mathbb{F}_p[X]$ . Die Äquivalenzklassen der Kongruenz modulo  $f$

$$g + f\mathbb{F}_p[X] := \{h \in \mathbb{F}_p[X] \mid h \equiv g \pmod{f}\}$$

bilden den Ring  $\mathbb{F}_p[X]/(f)$ . □

Für bestimmte Reduktionspolynome  $f$  ist der Restklassenring sogar ein Körper:

**Satz 2.10.**  $\mathbb{F}_p[X]/(f)$  ist genau dann ein Körper, wenn  $f$  irreduzibel in  $\mathbb{F}_p[X]$  ist.

*Beweis.* Siehe [25], Theorem 1.61. □

Jede Äquivalenzklasse  $g + f\mathbb{F}_p[X]$  enthält dabei einen eindeutig bestimmten Repräsentanten  $r \in \mathbb{F}_p[X]$  mit  $\deg(r) < \deg(f)$ , nämlich gerade den Rest bei Division von  $g$  durch  $f$ . Die Elemente des Restklassenrings modulo  $f$  können also alle in der Form  $r + f\mathbb{F}_p[X]$  dargestellt werden, wobei  $r$  alle Polynome aus  $\mathbb{F}_p[X]$  vom Grad kleiner  $\deg(f)$  durchläuft. Folglich enthält  $\mathbb{F}_p[X]/(f)$  genau  $p^n$  Elemente, wobei  $n = \deg(f)$ .

**Definition 2.11** (Galoiskörper von Primzahlpotenzordnung). Für eine Primzahl  $p$  und ein irreduzibles Polynom  $f \in \mathbb{F}_p[X]$  mit  $\deg(f) = n$  sei  $\mathbb{F}_{p^n}$  definiert als die Menge der Polynome aus  $\mathbb{F}_p[X]$  vom Grad kleiner  $n$ , und die Abbildung  $\Psi : \mathbb{F}_p[X]/(f) \rightarrow \mathbb{F}_{p^n}$  vermöge  $\Psi(r + f\mathbb{F}_p[X]) = r$  für alle  $r \in \mathbb{F}_p[X]$  mit  $\deg(r) < n$ . Dann ist  $\mathbb{F}_{p^n}$  mit der durch  $\Psi$  induzierten Körperstruktur ein endlicher Körper, der **Galoiskörper der Ordnung  $p^n$** . □

*Bemerkung 2.12.* Wie beim Galoiskörper von Primzahlordnung schreibt man auch  $GF(p^n)$  für  $\mathbb{F}_{p^n}$ .

Es kann gezeigt werden, dass für jedes  $n \in \mathbb{N}$  mindestens ein irreduzibles Polynom vom Grad  $n$  existiert, es also für alle Primzahlen  $p$  und alle natürlichen Zahlen  $n$  auch einen Galoiskörper der Ordnung  $p^n$  gibt [25]. Darüber hinaus können endliche Körper nur für Primzahlpotenzen  $q = p^n$  existieren, und alle endlichen Körper der Ordnung  $q$  sind isomorph – damit kann bei der Konstruktion insbesondere die Wahl des in der Regel nicht eindeutigen Reduktionspolynoms beliebig erfolgen.

Im Falle von  $p = 2$  bezeichnen wir die Elemente von  $GF(2^n)$  kurz über die Hexadezimaldarstellung der Koeffizientenbits, in  $GF(2^4)$  ist etwa  $X^3 + X^2 + 1 = 1101_2 = \mathbf{d}_x$ .

### 2.1.1 Verschlüsselung

Für die Kryptografie als Lehre von der Verschlüsselung zentraler Bedeutung ist der Begriff des Kryptosystems:

**Definition 2.13** (Kryptosystem). Ein Kryptosystem ist ein Fünftupel  $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$  mit folgenden Eigenschaften:

- (a)  $\mathcal{P}$  ist die (endliche) Menge der möglichen Klartexte;
- (b)  $\mathcal{C}$  ist die (endliche) Menge der möglichen Chiffretexte;
- (c)  $\mathcal{K}$  ist die (endliche) Menge der möglichen Schlüssel;
- (d)  $\mathcal{E} = \{e_k : k \in \mathcal{K}\}$  ist eine Familie von Verschlüsselungsfunktionen  $e_k : \mathcal{P} \rightarrow \mathcal{C}$ ;
- (e)  $\mathcal{D} = \{d_k : k \in \mathcal{K}\}$  ist eine Familie von Entschlüsselungsfunktionen  $d_k : \mathcal{C} \rightarrow \mathcal{P}$ ;
- (f) Für alle Schlüssel  $k$  und alle Klartexte  $p$  gilt  $d_k(e_k(p)) = p$ . □

Als unmittelbare Konsequenz aus Eigenschaft (f) ergibt sich

**Korollar 2.14.** (a) Die Verschlüsselungsfunktionen eines Kryptosystems sind injektiv.

(b) Ist  $\mathcal{P} = \mathcal{C}$ , dann sind sie Permutationen. □

Blockchiffren sind spezielle Kryptosysteme:

**Definition 2.15** (Blockchiffre, [29]). Eine Blockchiffre ist ein Kryptosystem, in welchem  $\mathcal{P} = \mathcal{C} = \mathbb{F}_2^n$  für festes  $n \in \mathbb{N}$ . Wir nennen  $n$  die **Blocklänge** der Chiffre. Eine Blockchiffre der Blocklänge  $n$  wird als  $n$ -Bit-Blockchiffre bezeichnet. □

Nach vorigem Korollar sind die Verschlüsselungsfunktionen  $e_k$  einer Blockchiffre stets bijektiv. Die allgemeinste oder auch *ideale* Blockchiffre erhält man daher, indem man in  $\mathcal{E}$  abhängig von  $k$  jede Permutation von  $\mathbb{F}_2^n$  zulässt:

**Definition 2.16** (Ideale Blockchiffre). Die ideale Blockchiffre mit Blocklänge  $n$  ist definiert durch  $\mathcal{P} = \mathcal{C} = \mathbb{F}_2^n$ ,  $\mathcal{K} = S_{2^n} = \{\pi \mid \pi \text{ ist Permutation von } \mathbb{F}_2^n\}$  und

$$\begin{aligned} e_k(x) &:= \pi(x) \\ d_k(y) &:= \pi^{-1}(y) \end{aligned}$$

für  $k = \pi \in \mathcal{K}$  und  $x, y \in \mathbb{F}_2^n$ . □

(Um tatsächlich von der *idealen* Blockchiffre sprechen zu können, muss man wie üblich voraussetzen, dass die Schlüssel zufällig und gleichverteilt gewählt werden, d.h. das Verschlüsseln mit einem bestimmten Schlüssel der Anwendung einer zufällig aus  $S_{2^n}$

gewählten Permutation entspricht.) Die ideale Blockchiffre hat unter allen für die feste Blocklänge  $n$  denkbaren Blockchiffren den größten Schlüsselraum, nämlich

$$|\mathcal{K}| = (2^n)!,$$

was bereits für kleines  $n$  einen vollkommen impraktikablen Speicherplatzbedarf zur Ablage eines Schlüssels zur Folge hat: Ein Schlüssel  $k$  lässt sich in etwa  $\log_2(2^n!)$  Bits binär kodieren, was unter Verwendung der Stirling-Approximation<sup>1</sup> bereits für die Blocklänge  $n = 32$  etwa  $2^{37}$  Bits  $\approx 16$  GByte Speicher bedeutet; für die heute übliche Blocklänge  $n = 128$  ergeben sich gar ungefähr  $2^{135}$  Bits  $\approx 2^{102}$  GByte.

Damit ist klar, dass praktisch verwendbare Blockchiffren nur einen Bruchteil aller möglichen Permutationen von  $\mathbb{F}_2^n$  implementieren können. Eine gute Blockchiffre sollte jedoch nicht effizient von der idealen Blockchiffre *unterscheidbar* sein, d.h. die  $|\mathcal{K}|$  Verschlüsselungsfunktionen  $e_k$  sollten so erscheinen, als seien sie zufällig aus der Menge aller  $(2^n)!$  Permutationen von  $\mathbb{F}_2^n$  gewählt.

## 2.1.2 Betriebsmodi

Ihrem Namen gemäß verschlüsseln Blockchiffren Klartexte der festen Blocklänge von  $n$  Bits auf gleich lange Chiffretexte. Soll mehr als ein Klartextblock verarbeitet, etwa  $P_1, \dots, P_m$  zu  $C_1, \dots, C_m$  verschlüsselt werden, so muss man die Blockchiffre wiederholt anwenden. Die Art und Weise, in der dies geschieht, nennt man einen Betriebsmodus für Blockchiffren.

Grundsätzlich zu unterscheiden sind dabei Modi, welche ganze Blöcke, und solche, welche nur Teile von Blöcken auf einmal verarbeiten. Die hauptsächlichen Modi ersteren Typs sind

**Electronic Codebook (ECB)**, in welchem jeder Block unabhängig von allen anderen einzeln verarbeitet wird:

$$\begin{aligned} \text{Verschlüsselung:} \quad & C_i = e_k(P_i), \\ \text{Entschlüsselung:} \quad & P_i = d_k(C_i); \end{aligned}$$

**Cipher Block Chaining (CBC)**, wo der  $i$ -te Chiffretextblock vom  $i$ -ten Klartext- und dem  $i - 1$ -ten Chiffretextblock abhängt:

$$\begin{aligned} \text{Verschlüsselung:} \quad & C_i = e_k(P_i \oplus C_{i-1}), \\ \text{Entschlüsselung:} \quad & P_i = d_k(C_i) \oplus C_{i-1}, \end{aligned}$$

und für  $C_0 \in \mathbb{F}_2^n$  ein Startwert, der so genannte **Initialisierungsvektor (IV)**, gewählt wird; sowie

---

<sup>1</sup>Die Stirling-Formel zur Approximation der Fakultät ist

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n,$$

zur Approximationsgüte siehe [21], S. 115.

Counter (CTR), in dem der  $i$ -te Klartextblock mit der Verschlüsselung der Zahl  $c + i - 1$  verknüpft wird:

$$\begin{aligned} \text{Verschlüsselung:} \quad C_i &= e_k(c + i - 1 \bmod 2^n) \oplus P_i, \\ \text{Entschlüsselung:} \quad P_i &= e_k(c + i - 1 \bmod 2^n) \oplus C_i, \end{aligned}$$

wobei der Startwert  $c$  eine Zahl zwischen 0 und  $2^n - 1$  ist.

Eine Gegenüberstellung dieser Modi befindet sich in Abbildung 2.1.

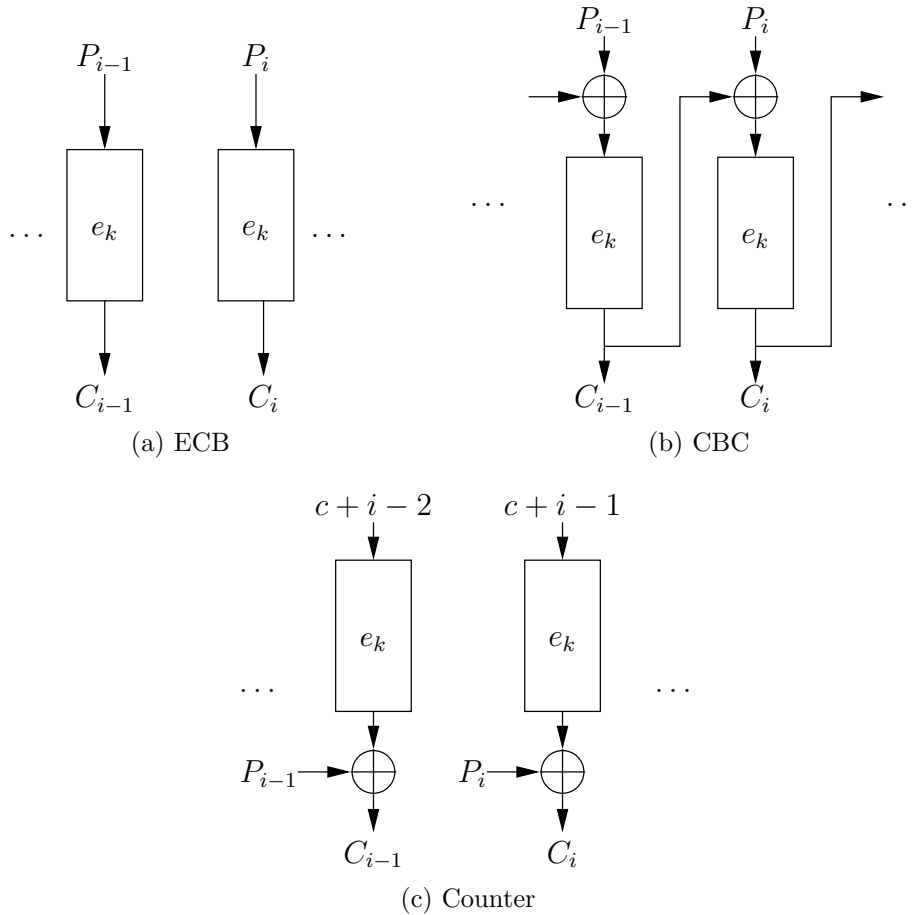


Abbildung 2.1: Verschlüsselung in verschiedenen blockweise operierenden Betriebsmodi.

Da im ECB-Modus identische Klartextblöcke zu identischen Chiffratextblöcken verschlüsselt werden, ist seine Verwendung nur für (nahezu) zufällige Klartexte empfehlenswert, wo eine Wiederholung nicht gefürchtet werden muss. CBC- und Counter-Modus haben diese Schwäche nicht und sind daher auch für redundante Klartexte verwendbar. Von ihnen ist CBC älter und damit besser untersucht, während CTR im Gegensatz zu CBC effizienten wahlfreien Zugriff auf einzelne verschlüsselte Klartextblöcke erlaubt. Außerdem muss im Counter-Modus lediglich die Verschlüsselungs-, nicht aber die Entschlüsselungsfunktion der Blockchiffre implementiert sein. Bei beiden ist essentiell, dass

für einen Schlüssel  $k$  der IV oder der Startwert  $c$  nicht wiederverwendet wird. Eine ausführliche Diskussion des CTR-Modus findet sich in [26].

Falls der Modus nur Teile eines Blocks auf einmal verarbeitet, emuliert die Blockchiffre quasi eine Stromchiffre. Die zwei bekanntesten Modi dieses Typs sind der **Output Feedback (OFB)**- sowie der **Cipher Feedback (CFB)**-Modus, siehe dazu u.a. [20].

### 2.1.3 Iterierte Blockchiffren

Praktisch alle modernen Blockchiffren sind als so genannte iterierte Chiffren realisiert:

**Definition 2.17** (Iterierte Blockchiffre). Eine **iterierte Blockchiffre** ist eine Blockchiffre, in welcher die Verschlüsselungsfunktion aus der  $r$ -fachen Iteration einer **Rundenfunktion**  $g : \mathcal{P} \times \mathcal{K} \rightarrow \mathcal{P}$  besteht. Dabei ist  $r \geq 1$  die **Rundenzahl** und  $(K^1, \dots, K^r)$  die Liste der vom Schlüssel  $k \in \mathcal{K}$  abgeleiteten **Rundenschlüssel**. Für  $p, c \in \mathbb{F}_2^n$  ist

$$\begin{aligned} e_k(p) = c \quad \text{mit} \quad & s^0 := p \\ & s^1 := g(s^0, K^1) \\ & \vdots \\ & s^r := g(s^{r-1}, K^r) \\ & c := s^r. \end{aligned}$$

Wir nennen  $s^i$  für  $0 \leq i \leq r$  den (**internen**) **Zustand** der Chiffre und die Liste der Rundenschlüssel den **Key Schedule** von  $k$ . Die Rundenfunktion  $g(s, k)$  muss für festes  $k$  eine Umkehrfunktion besitzen. Dann ist

$$\begin{aligned} d_k(c) = p \quad \text{mit} \quad & s^r := c \\ & s^{r-1} := g^{-1}(s^r, K^r) \\ & \vdots \\ & s^0 := g^{-1}(s^1, K^1) \\ & p := s^0. \end{aligned}$$

□

Bei einer iterierten Blockchiffre wird ein Klartext also verschlüsselt, indem eine schlüsselabhängige Rundenfunktion iteriert auf ihn angewandt wird, wobei das Ergebnis der letzten die Eingabe der nächsten Runde ist. Nach  $r$  Iterationen erhält man dann den Chiffretext. Zur Entschlüsselung wird der ganze Prozess umgekehrt. Damit nicht in jeder Runde die gleichen Schlüsseldaten verwendet werden, wird der Schlüssel  $k$  durch das Key-Scheduling in eine Liste von  $r$  Rundenschlüsseln expandiert. Dabei ist zu beachten, dass dieser deterministische Vorgang die effektive Schlüssellänge der Chiffre nicht vergrößert, sondern lediglich eine gute „Durchmischung“ der Abhängigkeit von  $c$  von den einzelnen Bits des Schlüssels  $k$  bewirken soll (vgl. dazu auch die Key-Schedule-basierten Angriffe in [3] und [13]).

## 2.2 Konstruktion iterierter Blockchiffren

Wann ist eine Blockchiffre „sicher“? Das hängt in erster Linie davon ab, welche Anforderungen an dieses Adjektiv gestellt werden. Nehmen wir die informationstheoretische Sicherheit ([5], Kapitel 4.4) als Maßstab, so erfordert dies zwingend  $|\mathcal{K}| \geq |\mathcal{C}| = 2^n$ , also eine Schlüssellänge größer oder gleich der Blocklänge, was bei vielen modernen Blockchiffren erfüllt ist. Jedoch muss für perfekte Sicherheit bei der Verschlüsselung jedes Blocks zufällig und gleichverteilt ein neuer Schlüssel gewählt werden. Unter dieser meist impraktikablen Rahmenbedingung bietet allerdings bereits das **One-Time-Pad** (vgl. [5], Kapitel 4.5) perfekte Sicherheit, so dass dafür nicht erst aufwändigere iterierte Blockchiffren verwendet werden müssen.

Für den Entwurf von Blockchiffren wird daher zumeist eine Art „praktische Sicherheit“ zu Grunde gelegt, welche erreicht ist, wenn jeder bekannte Angriff auf die Blockchiffre eine zu hohe (exponentielle) Laufzeitkomplexität aufweist. Dieser Sicherheitsbegriff ist folglich zwangsläufig recht diffus, da sowohl vom aktuellen Stand der Untersuchung der Blockchiffre als auch von den aktuellen berechnungstechnischen Möglichkeiten abhängig.

### 2.2.1 Allgemeine Prinzipien

Beim Entwurf einer iterierten Blockchiffre sind viele Parameter geeignet zu wählen: Die Blocklänge  $n$ , die Größe des Schlüsselraums  $\mathcal{K}$ , die Rundenzahl  $r$  sowie vor allem die Rundenfunktion  $g$ . Es ist (bislang) nicht möglich, genaue Kriterien anzugeben, wie diese Parameter zu wählen sind, um eine im praktischen Sinne sichere Konstruktion zu erhalten, wie etwa Knudsen in [20], S. 38 anmerkt:

„There have been many suggestions in the past of how to obtain good properties [...] for a block cipher, but so far there is no known exact recipe of how to construct a secure block cipher.“

Als notwendige Anforderungen bzw. empfehlenswerte Ansätze haben sich jedoch einige Aspekte etabliert.

**Nicht-Unterscheidbarkeit zur idealen Blockchiffre.** Die Blockchiffre sollte durch einen ressourcenbeschränkten Angreifer nicht mit Wahrscheinlichkeit signifikant größer als  $1/2$  von der idealen Blockchiffre unterscheidbar sein (vgl. Definition 2.16). Das ist eine sehr umfassende Forderung, da jeder Angriff als konkrete Ausprägung eines solchen „Unterscheidungsalgorithmus“ (englisch *Distinguisher*) interpretiert werden kann [11]. Für konkrete Chiffren ist ein solcher Nachweis schwer bis unmöglich und rührt an fundamentale Fragen der Komplexitätstheorie [4]. Unter der Annahme, dass eine nicht von einer Zufallspermutation unterscheidbare Blockchiffre eingesetzt wird, sind in diesem Modell jedoch diverse Betriebsmodi als sicher nachgewiesen worden.

**Ausreichende Schlüssellänge.** Jedes Kryptosystem kann durch probeweise Entschlüsselung mit allen möglichen Schlüsseln gebrochen werden („Brute Force“). Im Mittel

sind dabei  $|\mathcal{K}|/2$  Versuche notwendig, bis der verwendete Schlüssel identifiziert werden kann. Beträgt die Länge eines Schlüssels  $k$  Bits, so muss  $2^{k-1}$  groß genug sein, so dass kein Angreifer derart viele Probeentschlüsselungen durchführen kann. Knudsen [20] nennt  $k \geq 90$  als untere empfehlenswerte Schranke, bei den meisten modernen Blockchiffren ist  $k \geq 128$ .

**Ausreichende Blocklänge.** Nicht nur die Schlüssellänge, auch die Blocklänge sollte nicht zu klein gewählt sein. Wie in [19], Theoreme 4.4.1–4.4.3 gezeigt, kann ein Angreifer nach der Beobachtung und Speicherung von  $\mathcal{O}(2^{n/2})$  mit einem festen Schlüssel erzeugten Chiffretextblöcken mit hoher Wahrscheinlichkeit Informationen über die Klartexte gewinnen, etwa das XOR von zwei Klartextblöcken. (Das folgt aus dem Geburtstagsparadoxon, siehe [5], Kapitel 4.3.) Um solche Angriffe impraktikabel zu machen, sollte auch die Blocklänge ausreichend groß gewählt werden. Da sie für schnelle Software-Implementierungen gerne als Vielfaches der Wortgröße (32 oder 64 Bit) gewählt wird, sollte sie auf jeden Fall  $n \geq 96$  oder  $n \geq 128$  betragen.

**Resistenz gegen bekannte Angriffe.** Offensichtlich sollte eine Blockchiffre so entworfen sein, dass sie von keinem bekannten Angriff schneller als mittels exhaustiver Suche zu brechen ist. Konkret bedeutet dies vor allem, dass sie immun gegen die so genannte differentielle und lineare Kryptoanalyse (Abschnitte 2.3.2 und 2.3.3), gegen Angriffe algebraischer Natur (Interpolation, Gleichungssysteme, ...) sowie gegen Key-Schedule-Angriffe sein sollte.

**Resistenz gegen unbekannte Angriffe.** Obwohl zweifelsohne wünschenswert, ist die Realisierbarkeit dieses Kriteriums sehr umstritten. Während Fokussierung auf die Abwehr bekannter Angriffe mitunter neue Angriffsformen geradezu herausfordert (Knudsen etwa nennt in [20] einige Beispiele), garantiert auch das nicht selten als Mittel gegen unbekannte Angriffe gewählte Prinzip, möglichst „zufällige“ und „wilde“ Konstruktionen in die Rundenfunktion einzubauen, nicht zwangsläufig Immunität.<sup>2</sup> (Siehe jedoch das Konzept der Nicht-Unterscheidbarkeit von einer Zufallspermutation.)

**Produktchiffren.** Ein weithin anerkanntes Entwurfsprinzip ist die Hintereinanderausführung mehrerer, oft unterschiedlichen Zwecken dienender Transformationen. Geschieht dies in einer Weise, die das Resultat (hoffentlich) sicherer als die Einzeltransformationen werden lässt, spricht man von einer **Produktchiffre**. Iterierte Blockchiffren sind auf Grund der  $r$ -fachen Hintereinanderausführung der Rundenfunktion definitionsgemäß Produktchiffren; dieses Prinzip wird jedoch auch gerne für die Rundenfunktion selbst angewandt, die dann aus zwei oder mehreren verschiedenen Transformationen besteht. Zwei allgemeine angrifferschwerende Eigenschaften, die dabei angestrebt werden, sind *Konfusion* und *Diffusion*. Sie wurden bereits 1949 von Claude Shannon vorgeschlagen und werden in den nächsten beiden Abschnitten besprochen.

---

<sup>2</sup>Ein instruktives nichtkryptografisches Beispiel hierfür ist „Algorithm K“, ein besonders wild entworfener Zufallszahlengenerator ([22], Kapitel 3.1), der trotz seiner Komplexität nur sehr kurze Zyklen erzeugt oder sogar gegen einen festen Wert konvergiert.

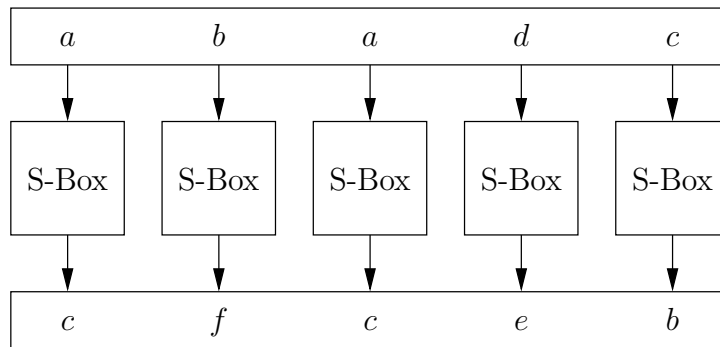


Abbildung 2.2: Operationsweise von Substitutions-Boxen.

### 2.2.2 Konfusion

Unter Konfusion versteht man ein möglichst komplexes Ein-/Ausgabeverhältnis einer Transformation. James Massey (zitiert nach [20], S. 38) erklärt Konfusion folgendermaßen:

„The ciphertext statistics should depend on the plaintext statistics in a manner too complicated to be exploited by the cryptanalyst.“

Diese sehr informelle Definition wird zumeist so interpretiert, dass eine Funktion um so höhere Konfusion bewirkt, desto nichtlinearer sie ist bzw. desto mehr sich ihre Eigenschaften von denen linearer und affiner Funktionen unterscheiden. Verschiedene Ansätze zur Präzisierung werden in Abschnitt 2.4 (Linearitätsmaße Boole'scher Abbildungen) behandelt.

Üblicherweise wird Konfusion in der Rundenfunktion durch Substitution realisiert, d.h. Blöcke der Länge  $l$  (wobei  $l \leq n$ ) werden unabhängig voneinander in Blöcke der Länge  $l'$  ersetzt, siehe Abbildung 2.2. Eine solche Transformation  $s : \mathbb{F}_2^l \rightarrow \mathbb{F}_2^{l'}$  wird auch als **Substitutions-Box** (oder kurz **S-Box**) bezeichnet. Häufig ist  $l = l'$  und die Substitutionsvorschrift bijektiv.

### 2.2.3 Diffusion

Erneut nach Massey bedeutet Diffusion:

„Each digit of the plaintext and each digit of the secret key should influence many digits of the ciphertext.“

Im Gegensatz zur Konfusion steht bei der Diffusion also das Abhängigkeitsverhältnis jedes einzelnen Ausgabebits von den einzelnen Eingabebits im Mittelpunkt. Damit soll sichergestellt werden, dass bereits kleine Änderungen in der Eingabe zu umfangreichen Änderungen in der Ausgabe führen können. In der Regel werden mindestens zwei Runden benötigt, bis nennenswerte Diffusionseffekte auftreten.

In der Rundenfunktion wird Diffusion häufig durch Transposition realisiert, d.h. die vorhandenen Zeichen werden permutiert, es wird jedoch im Gegensatz zur Konfusion

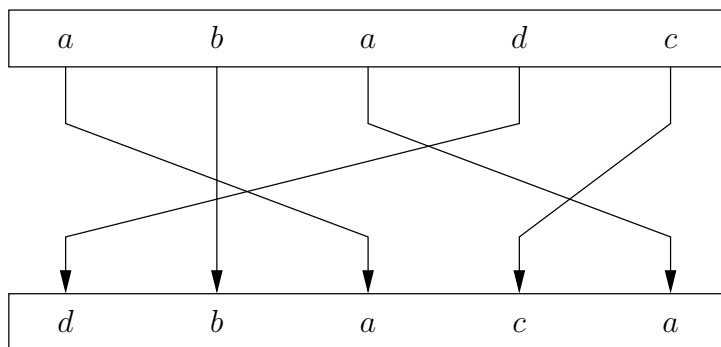


Abbildung 2.3: Diffusion durch Transposition.

keine *Ersetzung* vorgenommen. Im Falle von Blockchiffren bedeutet dies eine Bitpermutation. Um stärkere Diffusionseffekte zu erzielen, werden in modernen Blockchiffren zunehmend allgemeinere (in der Regel lineare) Transformationen eingesetzt, die im klassischen Sinne keine Transpositionen mehr sind. Das Diffusionsprinzip ist in Abbildung 2.3 anhand einer Transposition illustriert.

Es gibt verschiedene Ansätze zur Messung von Diffusion, unter anderem das Konzept von *Maximum Distance Separability* oder die so genannte *Branch Number*.

**Definition 2.18** (Hamming-Gewicht, Hamming-Distanz). Für einen Bitvektor  $a \in \mathbb{F}_2^n$  ist das **Hamming-Gewicht**  $w_h(a)$  definiert als

$$w_h(a) := \#\{i \mid a_i \neq 0\},$$

d.h. genau  $w_h(a)$  viele Komponenten von  $a$  sind ungleich Null. Als **Hamming-Distanz**  $d_h(a, b)$  zweier Vektoren  $a, b \in \mathbb{F}_2^n$  bezeichnen wir das Hamming-Gewicht von  $a \oplus b$ :

$$d_h(a, b) := w_h(a \oplus b).$$

□

Aus der Definition ist unmittelbar klar, dass die Hamming-Distanz von  $a$  und  $b$  gerade die Anzahl von Bitpositionen ist, an denen sich  $a$  und  $b$  unterscheiden:

$$d_h(a, b) = w_h(a \oplus b) = \#\{i \mid a_i \oplus b_i \neq 0\} = \#\{i \mid a_i \neq b_i\}.$$

**Definition 2.19** (Maximum Distance Separability, [18]). Für eine beliebige Abbildung  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$  vereinbaren wir

$$\text{Dist}_f := \min_{\substack{x, x' \in \mathbb{F}_2^n \\ x \neq x'}} d_h(x, x') + d_h(f(x), f(x')).$$

$f$  heißt **Maximum Distance Separable (MDS)**, wenn

$$\text{Dist}_f = n + 1. \tag{2.1}$$

Allgemein nennen wir  $f$  **MDS- $m$** , wenn  $\text{Dist}_f = m$ .

□

Die Bezeichnung „maximum“ rührt daher, dass MDS-Abbildungen hier eine obere Schranke erreichen: Für beliebiges  $f$  gilt  $d_h(f(x), f(x')) \leq n$ . Da aus  $d_h(x, x') = 0$  sofort  $x = x'$  folgt, kann  $d_h(x, x')$  minimal 1 sein, wenn  $x \neq x'$ . Folglich ist allgemein  $\text{Dist}_f \leq n + 1$ .

Für lineare Abbildungen lässt sich Diffusion mittels der von Joan Daemen eingeführten Verzweigungszahlen messen:

**Definition 2.20** (Branch Number, [8]). Die Branch Number (Verzweigungszahl) einer linearen Abbildung  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$  ist gegeben durch

$$\mathcal{B}(f) := \min_{0 \neq a \in \mathbb{F}_2^n} w_h(a) + w_h(f(a)). \quad (2.2)$$

□

Aus ähnlichen Überlegungen wie oben folgt allgemein  $\mathcal{B}(f) \leq n + 1$ . Darüber hinaus stellt sich heraus, dass der MDS-Distanzbegriff für lineare Abbildungen mit der Branch Number übereinstimmt:

**Proposition 2.21.** Für eine lineare Abbildung  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$  gilt

$$\text{Dist}_f = \mathcal{B}(f).$$

Insbesondere ist  $f$  genau dann MDS, wenn  $\mathcal{B}(f) = n + 1$ .

*Beweis.* Die Behauptung folgt sofort aus der Definition von  $\text{Dist}_f$  resp.  $\mathcal{B}(f)$ :

$$\begin{aligned} \text{Dist}_f &= \min_{x \neq x' \in \mathbb{F}_2^n} d_h(x, x') + d_h(f(x), f(x')) \\ &= \min_{x \neq x' \in \mathbb{F}_2^n} w_h(x \oplus x') + w_h(f(x) \oplus f(x')) \\ &= \min_{x \neq x' \in \mathbb{F}_2^n} w_h(x \oplus x') + w_h(f(x \oplus x')) \\ &= \min_{0 \neq a \in \mathbb{F}_2^n} w_h(a) + w_h(f(a)) \\ &= \mathcal{B}(f), \end{aligned}$$

was zu zeigen war. □

Da für die Diffusionsebene geeignete T-Funktionen in der Regel nichtlinear sind, werden wir zur Messung der Diffusion von gesamten Funktionen die MDS- bzw. MDS- $m$ -Eigenschaft heranziehen, während sich etwa für lineare Bit-Slice-Formen (vgl. Kapitel 3) die effizienter berechenbare Branch Number anbietet.

## 2.2.4 Feistel-Netze

Zwei hauptsächliche Konstruktionsformen für iterierte Blockchiffren haben sich in den letzten Jahrzehnten etabliert: Die von Horst Feistel 1973 vorgeschlagenen Feistel-Netze, sowie die allgemeinen Substitutions-Permutations-Netzwerke.

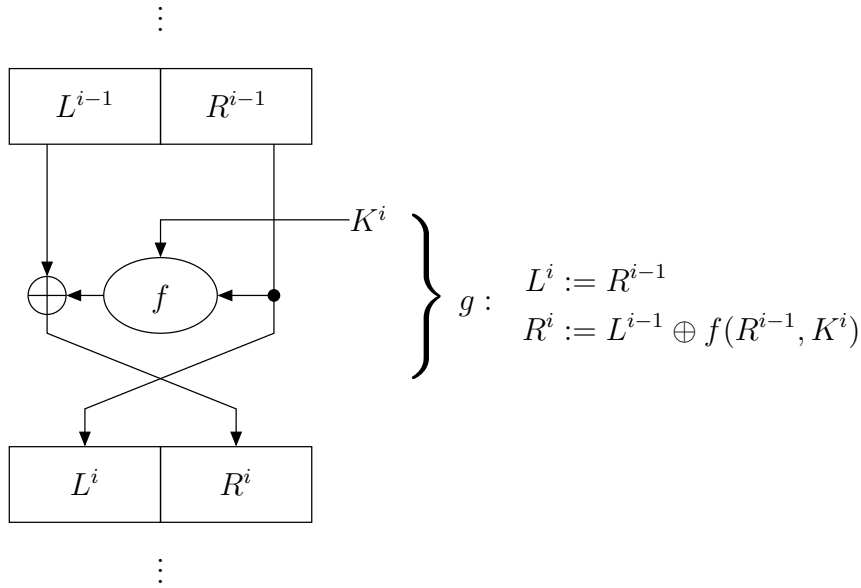


Abbildung 2.4: Rundenfunktion eines Feistel-Netzes.

In einem Feistel-Netz wird der Zustand der Chiffre in zwei Hälften aufgeteilt, eine linke und eine rechte:

$$s^i = (L^i, R^i),$$

wobei  $n = 2b$ ,  $s^i \in \mathbb{F}_2^n$ ,  $L^i, R^i \in \mathbb{F}_2^b$ ; und die Rundenfunktion  $g : \mathbb{F}_2^b \times \mathbb{F}_2^b \times \mathcal{K} \rightarrow \mathbb{F}_2^n$  ist definiert als

$$(L^i, R^i) = g(L^{i-1}, R^{i-1}, K^i) := (R^{i-1}, L^{i-1} \oplus f(R^{i-1}, K^i)) \quad (2.3)$$

mit einer Funktion  $f : \mathbb{F}_2^b \times \mathcal{K} \rightarrow \mathbb{F}_2^b$ , welche sozusagen die „eigentliche“ Verschlüsselungsfunktion konstituiert und stets auf der rechten Hälfte und dem  $i$ -ten Rundenschlüssel operiert. Dabei ist  $f$  sowohl für Konfusion als auch für Diffusion zuständig und daher meist selbst eine Produktchiffre. Von einer Runde zur nächsten werden dabei die Rollen der linken und rechten Hälfte vertauscht, die linke wird zusätzlich mit dem schlüsselabhängigen Funktionsergebnis der rechten Hälfte verknüpft. Eine schematische Darstellung findet sich in Abbildung 2.4. In der letzten Runde wird die Vertauschung von  $L$  und  $R$  unterlassen, da auf diese Weise die Entschlüsselung exakt der Verschlüsselung mit umgekehrtem Key-Schedule entspricht.

Diese Konstruktion hat die Besonderheit, dass  $f$  nicht injektiv (und damit auch nicht invertierbar) sein muss, damit die Rundenfunktion  $g$  und damit das Feistel-Netz invertierbar ist, denn es gilt

$$\begin{aligned} R^{i-1} &= L^i \\ L^{i-1} &= R^i \oplus f(L^i, K^i) \end{aligned}$$

nach Definition von  $g$ .

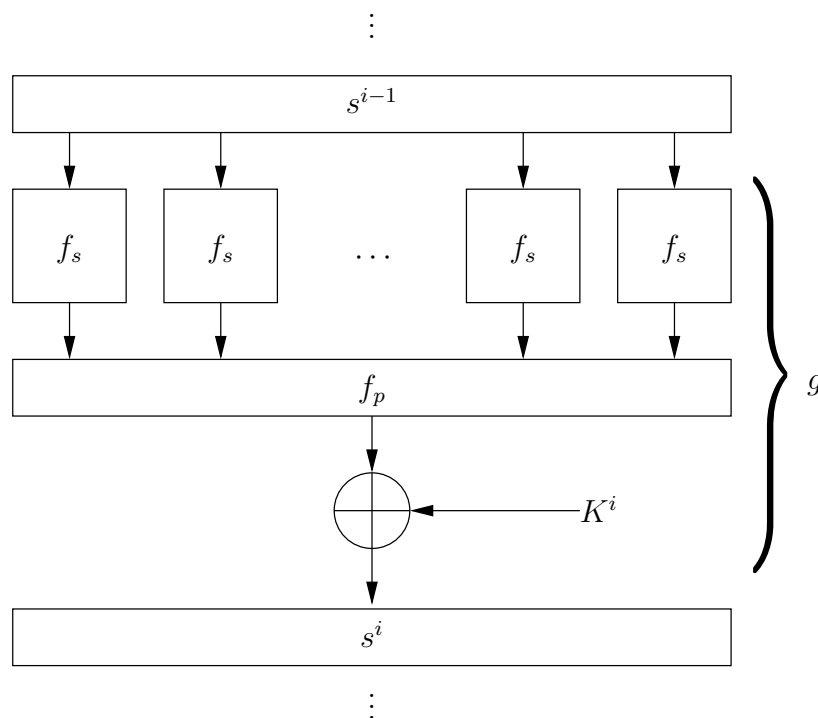


Abbildung 2.5: Rundenfunktion eines SP-Netzwerks.

### 2.2.5 Substitutions-Permutations-Netzwerke

Substitutions-Permutations-Netzwerke (SP-Netzwerke) sind eine sehr allgemeine Form iterierter Blockchiffren: Die einzelnen Runden bestehen hier aus der alternierenden Hintereinanderausführung von Substitution, Permutation und Schlüsseladdition. (Der Name „Permutation“ für die Diffusionsebene liegt darin begründet, dass sie klassischerweise mittels Bitpermutationen implementiert wurde.) Damit werden die Konfusions- und Diffusionsaufgaben zwei Funktionen  $f_s$  und  $f_p$  zugeordnet, wobei  $f_s$  eine S-Box und  $f_p$  eine Funktion mit guten Diffusionseigenschaften ist. Jede der dadurch beschriebenen „Schichten“ operiert auf dem gesamten Zustand, wobei die Substitution üblicherweise in unabhängigen kleineren Blöcken erfolgt und die Größe der S-Boxen demnach ein Teiler der Blocklänge sein muss, während  $f_p$  auf dem gesamten  $n$ -Bit-Zustand operiert. In jeder Runde wird die Schlüsseladdition getrennt von der Anwendung von  $f_s$  und  $f_p$  auf  $s^i$  durchgeführt; sie ist meist eine XOR-Verknüpfung von  $s^i$  und dem Rundenschlüssel  $K^i$ , der demnach ebenfalls  $n$  Bit lang sein muss (siehe Abbildung 2.5).

Diese Konstruktion erfordert, dass alle Ebenen und damit auch  $f_s$  und  $f_p$  invertierbar sein müssen, damit die Rundenfunktion umkehrbar ist. Die Entschlüsselung arbeitet dann das SP-Netzwerk „rückwärts“ ab, wobei an Stelle von  $f_s$  und  $f_p$  deren Umkehrfunktionen zum Einsatz kommen.

SP-Netzwerke sind insofern allgemeiner als Feistel-Netze, als dass sich jedes Feistel-Netz als SP-Netzwerk darstellen lässt ([31], S. 10).

## 2.3 Kryptoanalyse

Dual zur Kryptografie als Lehre von der Verschlüsselung bezeichnet Kryptoanalyse die Lehre vom Brechen von Kryptosystemen, oder – etwas weniger martialisch formuliert – die Lehre von der Sicherheitsanalyse von Kryptosystemen. Ziel der Kryptoanalyse ist es, entweder Chiffretexte ohne Kenntnis des Schlüssels zu entschlüsseln, oder aber den a priori unbekanntem Schlüssel zu bestimmen. Ein solches Vorgehen wird als **Angriff** bezeichnet. Da ein Brute-Force-Angriff zur Schlüsselsuche unabhängig von der konkreten Struktur der Chiffre stets<sup>3</sup> möglich ist, sind nur solche Angriffe von Interesse, welche den Schlüssel oder signifikante Teile des Schlüssels in weniger als  $|\mathcal{K}|/2$  Schritten liefern.

### 2.3.1 Klassifikation von Angriffen auf Blockchiffren

Je nachdem, welche Daten dem Angreifer für die Durchführung eines bestimmten Angriffs vorliegen müssen, kann man Angriffe auf Blockchiffren wie folgt klassifizieren. In allen Fällen wird vorausgesetzt, dass sämtliche Chiffretexte mit genau einem, dem gesuchten, Schlüssel verschlüsselt worden sind, also  $C_i = e_k(P_i)$  für fixes  $k$  gilt:

**Ciphertext Only.** Dem Angreifer liegt eine gewisse Menge an Chiffretexten vor.

**Known Plaintext.** Der Angreifer besitzt Klartexte  $P_1, \dots, P_s$  und dazugehörige Chiffretexte  $C_1, \dots, C_s$ .

**Chosen Plaintext.** Der Angreifer *wählt* vor dem Angriff eine Reihe  $P_1, \dots, P_s$  von Klartexten und *erhält* die Verschlüsselungen  $C_1, \dots, C_s$ .

**Adaptively Chosen Plaintext.** Der Angreifer wählt während des Angriffs *interaktiv* Klartexte  $P_1, \dots, P_s$  und erhält korrespondierende Chiffretexte  $C_1, \dots, C_s$ . Mit anderen Worten: Er wählt  $P_1$ , erhält  $C_1$ , wählt  $P_2$  usw.

Die Angriffsarten sind hier in absteigender Mächtigkeit geordnet aufgeführt. Offensichtlich ist ein Angriff, zu dessen erfolgreicher Durchführung nur einige Chiffretexte vorliegen müssen, stets auch dann anwendbar, wenn dazugehörige Klartexte bekannt sind usw.

Zusätzlich zur Kategorie wird die Wirksamkeit eines Angriffs durch seine Komplexität beschrieben, sowohl an Laufzeit (zur Berechnung von Probeentschlüsselungen, Lösung von Gleichungssystemen, ...) als auch an Speicherplatz (zur Aufnahme der vorliegenden Klar- und Chiffretexte). Häufig ist die Laufzeitkomplexität bis auf einen kleineren Faktor an die Speicherplatzkomplexität gekoppelt, da für viele Angriffe die Anzahl der benötigten Klar- oder Chiffretexte so hoch ist, dass der Aufwand zur ihrer Verarbeitung den zusätzlichen Berechnungsaufwand dominiert.

---

<sup>3</sup>Präziser: Solange eine die Unizitätsdistanz übersteigende Menge Chiffretext vorhanden ist, siehe etwa [19], Abschnitt 4.2. Für praktische Zwecke kann davon ausgegangen werden, dass diese Voraussetzung stets erfüllt ist. Darüber hinaus wäre eine Blockchiffre, bei der man nach wenigen Blöcken den Schlüssel wechseln müsste, indiskutabel.

### 2.3.2 Differentielle Kryptoanalyse

Eine mächtige allgemeine Angriffstechnik ist die **differentielle Kryptoanalyse**, die 1990 von Biham und Shamir [1] vorgestellt wurde und neben Blockchiffren auf weitere kryptografische Primitive, insbesondere Hashfunktionen, anwendbar ist. Sie basiert auf der Beobachtung, dass für zwei Klartexte  $x, x^*$  mit einer bestimmten *Differenz*  $x'$  die möglichen Differenzen  $y'$  der Chiffretexte  $y = e_k(x), y^* = e_k(x^*)$  in der Regel nicht gleichverteilt sind. Die Idee ist nun, eine ausreichende Anzahl von Klartexten einer festen Differenz zu wählen, für welche eine bestimmte Chiffretextdifferenz mit besonders hoher Wahrscheinlichkeit auftritt, und diese Ungleichverteilung zur Bestimmung des wahrscheinlichsten Kandidaten für einen Teil des Schlüssels zu benutzen. Die differentielle Kryptoanalyse ist folglich ein Chosen-Plaintext-Angriff, bei dem eine genügend große Menge an Klartext-Chiffretext-Paaren mit bestimmten Klartextdifferenzen gewählt wird; die eigentlichen Werte der Klar- bzw. Chiffretexte spielen dabei keine Rolle.

Der Begriff der Differenz ist dabei folgendermaßen erklärt:

**Definition 2.22** (Differenz). Sei  $(G, \odot)$  eine Gruppe mit Neutralelement  $e$ . Für zwei Elemente  $x, x^* \in G$  nennen wir

$$\Delta(x, x^*) := x \odot (x^*)^{-1} \quad (2.4)$$

die *Differenz* von  $x$  und  $x^*$  bezüglich  $\odot$ . Anstatt  $\Delta(x, x^*)$  schreiben wir abkürzend auch  $x'$ .  $\square$

In aller Regel wird bei einem differentiellen Angriff für die Differenzenbildung diejenige Operation zu Grunde gelegt, die für die Schlüsseladdition verwendet wird; zumeist ist dies die Addition in  $\mathbb{F}_2^n$ , also das XOR von  $n$ -Bit-Vektoren. In diesem Fall gilt  $x' = x \oplus x^*$  für die Differenz von  $x$  und  $x^*$ .

Um die Differenzenpropagierung einer gesamten iterierten Blockchiffre zu untersuchen, ist es nützlich, sich zunächst auf eine Runde, genauer: die Bestandteile einer Rundenfunktion zu konzentrieren. Es stellt sich heraus, dass die Differenz invariant unter der Schlüsseladdition ist:

$$\begin{aligned} \Delta(s \odot k, s^* \odot k) &= (s \odot k) \odot (s^* \odot k)^{-1} \\ &= (s \odot k) \odot (k^{-1} \odot (s^*)^{-1}) \\ &= s \odot (k \odot k^{-1}) \odot (s^*)^{-1} \\ &= s \odot (s^*)^{-1} \\ &= \Delta(s, s^*). \end{aligned} \quad (2.5)$$

Außerdem ist die Differenz mit Gruppenendomorphismen  $\phi : G \rightarrow G$  verträglich, denn für beliebiges  $a \in G$  gilt  $e = \phi(e) = \phi(a \odot a^{-1}) = \phi(a) \odot \phi(a^{-1})$ , also  $\phi(a)^{-1} = \phi(a^{-1})$  und damit

$$\begin{aligned} \Delta(\phi(x), \phi(x^*)) &= \phi(x) \odot \phi(x^*)^{-1} \\ &= \phi(x) \odot \phi((x^*)^{-1}) \\ &= \phi(x \odot (x^*)^{-1}) \\ &= \phi(\Delta(x, x^*)). \end{aligned} \quad (2.6)$$

$(x, x^*)$	$y' = \Delta(h(x), h(x^*))$
$(000_2, 001_2)$	$101_2$
$(001_2, 000_2)$	$101_2$
$(010_2, 011_2)$	$001_2$
$(011_2, 010_2)$	$001_2$
$(100_2, 101_2)$	$101_2$
$(101_2, 100_2)$	$101_2$
$(110_2, 111_2)$	$001_2$
$(111_2, 110_2)$	$001_2$

Tabelle 2.1: Verteilung der Ausgabedifferenzen von  $h$  bei  $x' = 001_2$ .

Im Falle von  $G = (\mathbb{F}_2^n, \oplus)$  bedeutet dies Linearität in einer linearen Abbildung  $f$ :

$$\Delta(f(x), f(x^*)) = f(x) \oplus f(x^*) = f(x \oplus x^*) = f(\Delta(x, x^*)). \quad (2.7)$$

Bestünde eine Rundenfunktion lediglich aus linearen Komponenten, so könnte man also bei Kenntnis der Eingabedifferenz  $x'$  unabhängig vom Rundenschlüssel die Ausgabedifferenz mit Wahrscheinlichkeit 1 als  $g(x')$  angeben, und durch Iteration der Rundenfunktion die Chiffretextdifferenz in Abhängigkeit von der Klartextdifferenz. Dies wäre eine sehr deutliche Ungleichverteilung auf den Chiffretextdifferenzen.

Nichtlineare Komponenten wie die S-Boxen erlauben jedoch im Allgemeinen keinen sicheren Schluss von der Eingabe- auf die Ausgabedifferenz. Mit  $\oplus$  als Differenz sind bei der nichtlinearen Funktion

$$h : \mathbb{F}_2^3 \rightarrow \mathbb{F}_2^3 \\ x \mapsto x \oplus ((x + 5)^2 \vee 1)$$

für eine Eingabedifferenz von  $x' = 001_2$  lediglich die Ausgabedifferenzen  $y' = 001_2$  und  $y' = 101_2$  möglich. Tabelle 2.1 führt alle acht Eingabepaare  $(x, x^*)$  mit  $x' = 001_2$  auf. Diese Verteilung der Ausgabedifferenzen bedeutet, dass aus  $x' = 001_2$  jeweils mit Wahrscheinlichkeit  $\frac{4}{8} = \frac{1}{2}$  auf  $y' = 001_2$  bzw.  $y' = 101_2$  geschlossen werden kann. Alle sechs anderen Ausgabedifferenzen können nicht auftreten und haben somit Wahrscheinlichkeit 0. Dies ist eine starke Ungleichverteilung. Mit XOR als Gruppenoperation ist  $\Delta(x, x^*) = \Delta(x^*, x)$ , so dass zwar allgemein keine Gleichverteilung von  $y'$  möglich ist, dennoch könnten zumindest vier verschiedene Werte jeweils zwei Mal auftreten.

Fährt man auf diese Weise für alle Eingabedifferenzen fort, so erhält man die *Differenzverteilungstabelle*, oder kurz: *Differenztabelle* von  $h$ , in der für jede Eingabedifferenz  $x'$  die Verteilung der daraus resultierenden Ausgabedifferenzen  $y'$  tabelliert ist (siehe Tabelle 2.2 auf der nächsten Seite). Die Zeile  $x' = 001_2$  bestätigt dabei das Ergebnis von Tabelle 2.1; auffällig ist der Maximaleintrag 8 bei  $x' = y' = 100_2$ , der besagt, dass diese Eingabedifferenz in jedem Fall eine identische Ausgabedifferenz zur Folge hat. Wir beobachten weiterhin, dass eine Eingabedifferenz von 0 stets zu einer Ausgabedifferenz von 0 führt, was für jede deterministische Funktion gelten muss.

		$y'$							
		000 <sub>2</sub>	001 <sub>2</sub>	010 <sub>2</sub>	011 <sub>2</sub>	100 <sub>2</sub>	101 <sub>2</sub>	110 <sub>2</sub>	111 <sub>2</sub>
$x'$	000 <sub>2</sub>	8	0	0	0	0	0	0	0
	001 <sub>2</sub>	0	4	0	0	0	4	0	0
	010 <sub>2</sub>	0	0	4	0	0	0	4	0
	011 <sub>2</sub>	0	0	0	4	0	0	0	4
	100 <sub>2</sub>	0	0	0	0	8	0	0	0
	101 <sub>2</sub>	0	4	0	0	0	4	0	0
	110 <sub>2</sub>	0	0	4	0	0	0	4	0
	111 <sub>2</sub>	0	0	0	4	0	0	0	4

Tabelle 2.2: Vollständige Differenzentabelle von  $h$ .

Dieses Vorgehen motiviert die folgende Definition:

**Definition 2.23** (Differenzentabelle). Die Differenzentabelle einer Funktion  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q$  ist definiert durch

$$\#D_f(u, v) := \#\{x \in \mathbb{F}_2^n \mid \Delta(f(x \odot u), f(x)) = v\}.$$

□

Die Menge  $D_f(u, v)$  enthält also alle diejenigen Eingaben  $x$ , für die eine Eingabedifferenz von  $u$  eine Ausgabedifferenz von  $v$  zur Folge hat. Die Wahrscheinlichkeit, dass die Eingabedifferenz  $u$  die Ausgabedifferenz  $v$  zur Folge hat, ist demnach  $\#D_f(u, v)/2^n$ .

Basierend auf den Differenzentabellen der nichtlinearen Bestandteile der Rundenfunktion  $g$  kann nun die Propagierung einer bestimmten Klartextdifferenz  $P'$  durch die einzelnen Runden einer iterierten Blockchiffre verfolgt werden. Für die linearen Komponenten kann die Ausgabedifferenz als Funktionswert der Eingabedifferenz berechnet werden, in den nichtlinearen Komponenten muss man sich dabei mit einer gewissen Erfolgswahrscheinlichkeit für eine bestimmte mögliche Ausgabedifferenz „entscheiden“. Eine solche Folge von Differenzen nennt man eine differentielle Charakteristik ([1], [20]):

**Definition 2.24** (Differentielle Charakteristik). Eine differentielle Charakteristik über  $\rho$  Runden (kurz:  $\rho$ -Runden-Charakteristik) ist ein  $\rho + 1$ -Tupel

$$\Xi = (x'_0, x'_1, \dots, x'_\rho)$$

von Differenzen, wobei  $x'_i$  für  $1 \leq i \leq \rho$  der erwartete Wert der Chiffretextdifferenz  $C'_i$  nach  $i$  Runden und  $P' = x'_0$  die Klartextdifferenz ist.

Als **Wahrscheinlichkeit** einer Charakteristik bezeichnen wir die Wahrscheinlichkeit, dass ein zufälliges Paar  $(x_0, x_0^*)$  mit Differenz  $x'_0$  bei iterierter Anwendung der Rundenfunktion die Differenzen  $x'_1, \dots, x'_\rho$  der Charakteristik hervorruft, wenn die Rundenschlüssel zufällig und gleichverteilt gewählt werden:

$$\Pr(\Xi) = \Pr((C'_\rho = x'_\rho, \dots, C'_1 = x'_1) \mid P' = x'_0). \quad (2.8)$$

□

Die Wahrscheinlichkeit einer Charakteristik nach Gleichung (2.8) zu bestimmen, ist sehr häufig zu aufwändig. Daher wird sie zumeist als das Produkt der Wahrscheinlichkeiten von Ein-Runden-Charakteristiken berechnet:

$$\begin{aligned} \Pr(\Xi) &= \Pr((C'_\rho = x'_\rho, \dots, C'_1 = x'_1) \mid P' = x'_0) \\ &= \prod_{i=1}^{\rho} \Pr(C_1 = x'_i \mid P' = x'_{i-1}). \end{aligned} \quad (2.9)$$

Hierbei wird angenommen, dass alle Runden unabhängig voneinander operieren. Für gewisse Klassen von Chiffren lässt sich diese Vereinfachung sogar theoretisch fundieren (siehe etwa [19], Kapitel 5.2).

Eine zweite Vereinfachung besteht darin, dass in der Praxis die gemäß Key-Scheduling bestimmten Rundenschlüssel in aller Regel nicht unabhängig sind; diese Annahme hat sich aber für viele Chiffren als geeignet herausgestellt, d.h. die Qualität der Schätzung für die Wahrscheinlichkeit einer Charakteristik nicht signifikant beeinträchtigt. Rijmen gibt in [31] jedoch zahlreiche Beispiele, in denen Angriffe durch schlüsselabhängige Erwägungen verbessert werden konnten.

**Beispiel 2.25.** Zur Illustration differentieller Charakteristiken betrachten wir das durch  $f(x, k) = h(x) \oplus k$  definierte einfache 6-Bit-Feistel-Netz. Die Rundenfunktion  $g$  ist dann durch (2.3) gegeben. Nach der Struktur von Feistel-Netzen können wir eine „sichere“ 1-Runden-Charakteristik mit Wahrscheinlichkeit 1 erhalten, indem wir die Klartextdifferenz der rechten Hälfte als 0 wählen, siehe Abbildung 2.6a auf der nächsten Seite. (Für Feistel-Netze ist es zweckmäßig, die einzelnen Differenzen als Tupel  $(L', R')$  anzugeben.) Ist die Differenz der rechten Hälfte hingegen  $001_2$ , so können wir nach der Differenzentabelle von  $h$ , die in diesem Fall mit derjenigen von  $f$  übereinstimmt, in 4 von 8 Fällen die Chiffretextdifferenz  $C'_1 = (L' \oplus 101_2, 101_2)$  erwarten. Das ergibt die Charakteristik in Abbildung 2.6b auf der nächsten Seite.

Längere Charakteristiken können durch **Konkatenierung** von kürzeren Charakteristiken konstruiert werden: Entspricht die Chiffretextdifferenz  $x'_\rho$  der  $\rho$ -Runden-Charakteristik  $\Xi_1$  (bei Feistel-Netzen: bis auf  $(L, R)$ -Vertauschung) der Klartextdifferenz  $x'_0$  der  $\sigma$ -Runden-Charakteristik  $\Xi_2$ , so ergibt die Konkatenierung von  $\Xi_1$  und  $\Xi_2$  eine  $\rho + \sigma$ -Runden-Charakteristik  $\Xi_3$ . Zum Beispiel können die 1-Runden-Charakteristiken aus 2.6b und 2.6a konkateniert werden, falls  $L' = 101$ . Die dabei entstehende Zwei-Runden-Charakteristik ist in Abbildung 2.7 auf der nächsten Seite dargestellt und hat dann gemäß (2.9) die Wahrscheinlichkeit  $\frac{1}{2} \cdot 1$ , also das Produkt der Wahrscheinlichkeiten der Einzelcharakteristiken.  $\square$

Für eine Charakteristik  $\Xi$  heißen Klartext-Chiffretext-Paare mit passender Klartextdifferenz  $x'_0$  **gute Paare**, wenn sie der Charakteristik folgen, d.h. die vorhergesagten Differenzen erzeugen; ansonsten nennen wir sie **falsche Paare**. Ist die Wahrscheinlichkeit von  $\Xi$  kleiner 1, so werden sich unter den dem Angreifer vorliegenden Paaren mit der gewünschten Klartextdifferenz gute und falsche befinden.

Ein differentieller Angriff kann nun folgendermaßen durchgeführt werden: Angenommen, wir haben eine  $r - 1$ -Runden-Charakteristik mit möglichst hoher Wahrscheinlichkeit

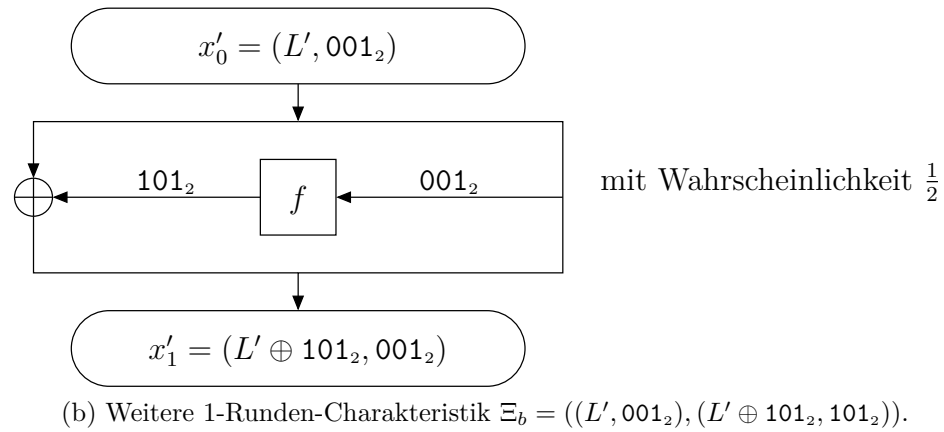
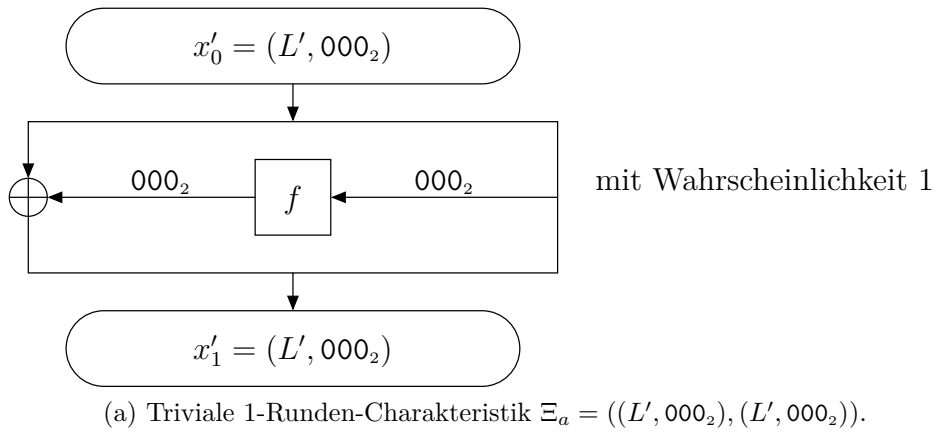


Abbildung 2.6: Differentielle 1-Runden-Charakteristiken eines 6-Bit-Feistel-Netzes.

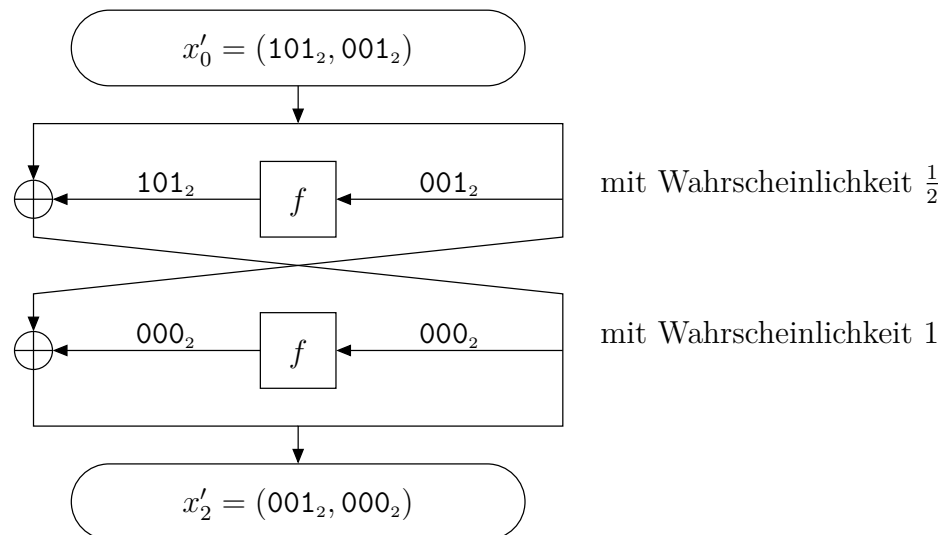


Abbildung 2.7: Durch Konkatenierung von (b) und (a) entstehende 2-Runden-Charakteristik  $\Xi_c = ((101_2, 001_2), (000_2, 001_2), (001_2, 000_2))$ .

$p$  für eine iterierte Blockchiffre von  $r$  Runden konstruiert, d.h. wir kennen mit Wahrscheinlichkeit  $p$  die Eingabedifferenz der letzten, vom Schlüssel  $K^r$  abhängigen Runde. Wir gehen der Reihe nach alle vorhandenen Klartext-Chiffretext-Paare durch. Dabei wird jeweils für alle möglichen Werte von  $K^r$  oder eines Teils davon die letzte Runde (partiell) rückgängig gemacht, d.h. die Chiffretexte (partiell) um eine Runde entschlüsselt. Nun kann die dabei auftretende Differenz mit dem von der Charakteristik vorhergesagten  $x'_{r-1}$  verglichen werden. Stimmt sie überein, so wird ein Zähler für diesen Schlüssel erhöht; wir sagen: der Schlüssel wird von diesem Paar **vorgeschlagen**.

Für gute Paare wird der korrekte Schlüssel vorgeschlagen werden; sowohl gute als auch falsche Paare werden falsche Schlüsselwerte vorschlagen. Zentrale Annahme dabei ist, dass ein falscher Schlüssel zu mehr oder weniger zufälligen Differenzen  $C'_{r-1}$  führen wird, während mit dem richtigen Schlüssel ungefähr mit Wahrscheinlichkeit der Charakteristik  $C'_{r-1} = x'_{r-1}$  gelten und somit dieser Schlüssel gezählt werden wird. Werden also genügend Paare durchprobiert, sollte der korrekte Schlüssel signifikant öfter vorgeschlagen werden als alle anderen (falschen) Schlüsselwerte. Ist der Rundenschlüssel  $K^r$ , und damit ein Teil des Hauptschlüssels  $k$ , einmal bekannt, können die restlichen Schlüsselbits entweder über exhaustive Suche oder über wiederholte Anwendung differentieller Kryptoanalyse ermittelt werden. In letzterem Fall käme dann zunächst eine  $r - 2$ -Runden-Charakteristik zum Einsatz, mit deren Hilfe  $K^{r-1}$  bestimmt wird, und so fort.

Die Anzahl der tatsächlich (partiell) zu entschlüsselnden Chiffretexte kann oft durch **Filterung** reduziert werden, indem offensichtlich falsche Paare erkannt und von der weiteren Betrachtung ausgeschlossen werden, deren Chiffretextdifferenz mit der Differenz  $x'_{r-1}$  inkompatibel ist. Das Verfahren ist in Algorithmus 2.1 schematisch dargestellt. Beispiele für differentielle Angriffe finden sich in Kapitel 4.5 und 4.6.

---

### Algorithmus 2.1 Differentieller 1R-Angriff

---

**Input:**  $r - 1$ -Runden-Charakteristik  $\Xi = (x'_0, \dots, x'_{r-1})$ .

**Input:** Klartext-Chiffretext-Paare  $(x, y, x^*, y^*)$  mit Klartextdifferenz  $x'_0$ .

**Output:** Rundenschlüssel  $K^r$  oder ein Teil davon.

```

1: for all vorliegende Paare  $(x, y, x^*, y^*)$  do
2:   if  $y'$  kompatibel zu  $x'_{r-1}$  then  $\triangleright$  Filterung
3:     for all Schlüsselkandidaten  $k$  do
4:       Berechne (ggf. partiell)  $C_{r-1} = g^{-1}(y, k)$  und  $C_{r-1}^* = g^{-1}(y^*, k)$ .
5:       if  $C_{r-1} = x'_{r-1}$  then
6:         Inkrementiere Zähler von  $k$ .
7:       end if
8:     end for
9:   end if
10: end for
11: return  $k$  mit maximalem Zählwert.

```

---

Ein wie zuvor beschrieben auf einer  $r - 1$ -Runden-Charakteristik basierender differentieller Angriff wird **1R-Angriff** genannt und ist die häufigste Spielart. Je nach Struktur der

Blockchiffre können auch *mR-Angriffe* möglich sein; dabei werden dann Zwischenwerte der letzten  $m$  Runden einbezogen.

Eine wichtige offene Frage ist noch, wieviele Paare benötigt werden, damit ein differentieller Angriff erfolgreich ist, also der korrekte Rundenschlüssel häufiger gezählt wird als alle anderen. Ist die Wahrscheinlichkeit der verwendeten Charakteristik  $p$ , so kann ein gutes Paar bei zufälliger Wahl etwa alle  $1/p$  Paare erwartet werden. Gewissermaßen als Faustregel für die Anzahl der benötigten Paare kann daher

$$N_{\text{pairs}} \approx c \cdot p^{-1} \quad (2.10)$$

verwendet werden, wobei  $c$  ein vom konkreten Angriff abhängiger kleiner konstanter Faktor ist (siehe [1]).

### 2.3.3 Lineare Kryptoanalyse

Neben der differentiellen Kryptoanalyse hat sich die 1993 von Matsui [27] entwickelte lineare Kryptoanalyse zu einer der bedeutendsten Angriffstechniken auf Blockchiffren entwickelt. Sie basiert auf der Idee, die typischerweise nichtlineare Chiffre durch *lineare Approximationen* der Form

$$(\alpha \bullet P) \oplus (\beta \bullet C) \oplus (\kappa \bullet K) = 0 \quad (2.11)$$

anzunähern, wobei  $P, C, K$  Klar- und Chiffretext sowie den Schlüssel bezeichnen, und  $\alpha, \beta, \kappa$  Auswahlvektoren für die an der linearen Approximation teilhabenden Bits von  $P, C$  und  $K$  sind. Wären  $P, C$  und  $K$  unabhängig voneinander und zufällig gewählte Vektoren, würde (2.11) mit Wahrscheinlichkeit  $1/2$  gelten, mit ebensolcher Wahrscheinlichkeit wäre der Ausdruck gleich Eins. Können für eine Blockchiffre die Auswahlvektoren  $\alpha, \beta, \kappa$  so gewählt werden, dass die Wahrscheinlichkeit  $p$ , mit der (2.11) gilt, signifikant von  $1/2$  verschieden ist, so ist die dabei entstehende lineare Approximation der Chiffre effektiver als reines Raten und kann mit Hilfe einer genügenden Anzahl von Klartext-Chiffretext-Paaren zur Bestimmung des wahrscheinlichsten Kandidaten für einen Teil des Schlüssels genutzt werden. Im Gegensatz zur differentiellen ist die lineare Kryptoanalyse also ein Known-Plaintext-Angriff.

Ist  $p > \frac{1}{2}$ , so ist (2.11) eine wahrscheinliche *lineare* Approximation, während  $p < \frac{1}{2}$  bedeutet, dass die *affine* Approximation  $(\alpha \bullet P) \oplus (\beta \bullet C) \oplus (\kappa \bullet K) = 1$  mit Wahrscheinlichkeit  $1 - p > \frac{1}{2}$  gilt. Im Kontext der linearen Kryptoanalyse sind sowohl lineare als auch affine Approximationen nutzbar und werden unter dem Begriff „linear“ zusammengefasst. Die Abweichung von  $1/2$  wird dabei als *Bias* bezeichnet:

**Definition 2.26** (Bias). Sei  $X$  eine binäre Zufallsvariable. Der Bias  $\epsilon$  von  $X$  ist gegeben durch

$$\epsilon := \Pr(X = 0) - \frac{1}{2}.$$

□

$x$	$h(x)$	$L_h$
$X_2X_1X_0$	$Y_2Y_1Y_0$	$X_2 \oplus Y_2 \oplus Y_1$
0 0 0	0 0 1	0
0 0 1	1 0 0	1
0 1 0	0 1 1	1
0 1 1	0 1 0	1
1 0 0	1 0 1	0
1 0 1	0 0 0	1
1 1 0	1 1 1	1
1 1 1	1 1 0	1

Tabelle 2.3: Gültigkeit der durch  $\alpha = 100_2$  und  $\beta = 110_2$  beschriebenen linearen Approximation  $L_h$  von  $h$ .

Da in einem linearen Angriff der gesuchte Schlüssel  $K$  fix ist, beeinflusst  $\kappa \bullet K$  die Effektivität einer linearen Approximation nicht, denn

$$(\alpha \bullet P) \oplus (\beta \bullet C) = 0 \quad (2.12)$$

wird je nachdem, ob  $\kappa \bullet K = 0$  oder  $\kappa \bullet K = 1$ , ebenfalls mit Wahrscheinlichkeit  $p$  bzw.  $1 - p$  gelten. Die Absolutbeträge des Bias von (2.12) und desjenigen von (2.11) sind daher gleich.

Entstände  $C$  als lineare Funktion von  $P$ , so könnten wir sofort lineare Approximationen der Form (2.12) mit maximalem Bias von  $1/2$  angeben: Ist  $f(x) = Ax$ , so wählen wir für beliebiges  $\beta$  den Vektor  $\alpha := A^T \beta$  und erhalten

$$\begin{aligned} (\alpha \bullet x) \oplus (\beta \bullet f(x)) &= (A^T \beta)^T x \oplus \beta^T (Ax) \\ &= (\beta^T A)x \oplus (\beta^T A)x \\ &= 0 \end{aligned}$$

für alle  $x$ .

Für nichtlineare Komponenten wie die S-Boxen können effektive Approximationen in der Regel jedoch nicht so direkt angegeben werden. Als Beispiel betrachten wir erneut die Funktion  $h : \mathbb{F}_2^3 \rightarrow \mathbb{F}_2^3$  von Seite 27. Ihr Ein-/Ausgabe-Verhalten soll durch lineare Approximationen der Form  $(\alpha \bullet x) \oplus (\beta \bullet h(x)) = 0$  angenähert werden. Die einzelnen Bits werden dabei als Zufallsvariablen  $X_i$  bzw.  $Y_i$  notiert:  $x = X_2X_1X_0$  sowie  $h(x) = Y_2Y_1Y_0$ . Werden nun beispielsweise die Auswahlvektoren  $\alpha = 100_2$  für die Eingabe sowie  $\beta = 110_2$  für die Ausgabe von  $h$  gewählt, so entsteht die lineare Relation

$$L_h : X_2 \oplus Y_2 \oplus Y_1 = 0. \quad (2.13)$$

Ihre Wahrscheinlichkeit kann bestimmt werden, indem sie für alle 8 möglichen Eingabe-Ausgabe-Paare  $(x, h(x))$  ausgewertet wird. Aus Tabelle 2.3 ist ersichtlich, dass (2.13) in

		$\beta$							
		$000_2$	$001_2$	$010_2$	$011_2$	$100_2$	$101_2$	$110_2$	$111_2$
$\alpha$	$000_2$	4	0	0	0	0	0	0	0
	$001_2$	0	-4	0	0	0	0	0	0
	$010_2$	0	0	4	0	0	0	0	0
	$011_2$	0	0	0	-4	0	0	0	0
	$100_2$	0	0	0	0	2	-2	-2	-2
	$101_2$	0	0	0	0	2	-2	2	2
	$110_2$	0	0	0	0	-2	-2	2	-2
	$111_2$	0	0	0	0	2	2	2	-2

Tabelle 2.4: Lineare Approximationstabelle von  $h$ .

2 von 8 Fällen, also mit Wahrscheinlichkeit  $1/4$  gilt. Der Bias der durch  $L_h$  beschriebenen Zufallsvariablen  $Z = X_2 \oplus Y_2 \oplus Y_1$  ist folglich  $\epsilon_Z = \frac{1}{4} - \frac{1}{2} = -\frac{1}{4}$ .

Fährt man auf diese Weise für alle möglichen Auswahlvektoren fort, entsteht die *lineare Approximationstabelle* von  $h$ , welche für alle Werte von  $\alpha$  und  $\beta$  die halbe Differenz zwischen der Anzahl an Fällen, in denen die durch  $\alpha, \beta$  beschriebene lineare Approximation gilt bzw. nicht gilt, auflistet. Für  $\alpha = 100_2, \beta = 110_2$  aus obigem Beispiel wäre der Eintrag demnach  $\frac{(2-6)}{2} = -2$ , siehe Tabelle 2.4. Ist ein Eintrag  $(\alpha, \beta) = 0$ , so ist die durch  $\alpha, \beta$  beschriebene lineare Approximation nicht besser als reines Raten. Negative Einträge korrespondieren zu affinen, positive zu linearen Approximationen, die in mehr als der Hälfte der Fälle zutreffen. Division eines Eintrages durch 8 ergibt den Bias der entsprechenden Approximation von  $h$ . Für eine beliebige Funktion stets zutreffend ist natürlich die triviale Relation  $0^T x \oplus 0^T f(x) = 0$ , so dass der Eintrag  $(0, 0)$  immer den maximalen Wert hat.

Trifft eine Approximation in  $a$  von  $n$  Fällen zu, so beträgt der korrespondierende Eintrag in der Tabelle  $\frac{a-(n-a)}{2} = a - \frac{n}{2}$ . Allgemein haben wir also

**Definition 2.27** (Lineare Approximationstabelle). Für eine Funktion  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q$  sowie  $\alpha \in \mathbb{F}_2^n, \beta \in \mathbb{F}_2^q$  bezeichnet

$$L_f(\alpha, \beta) := \{x \in \mathbb{F}_2^n \mid \alpha \bullet x = \beta \bullet f(x)\}$$

die Menge derjenigen  $x \in \mathbb{F}_2^n$ , für welche die durch  $\alpha, \beta$  beschriebene lineare Approximation von  $f$  zutrifft.

Die lineare Approximationstabelle (LAT) von  $f$  ist dann definiert als

$$\text{LAT}_f(\alpha, \beta) := \#L_f(\alpha, \beta) - 2^{n-1}.$$

□

Mit Hilfe linearer Approximationstabellen können nun mehrere Runden umspannende lineare Approximationen konstruiert werden, indem Ein-Runden-Approximationen so aneinandergehängt (präziser: summiert) werden, dass sich die Zwischenergebnisse

herauskürzen. Dieses stark an die Konkatenierung von Charakteristiken bei der differentiellen Kryptoanalyse erinnernde Vorgehen sieht schematisch so aus:<sup>4</sup>

$$[(\alpha \bullet P) \oplus (\beta \bullet C_\rho) \oplus (\kappa \bullet K) = 0] \quad (2.14)$$

$$\oplus [(\beta \bullet C_\rho) \oplus (\gamma \bullet C_{\rho+1}) \oplus (\delta \bullet K) = 0] \quad (2.15)$$

$$\rightsquigarrow [(\alpha \bullet P) \oplus (\gamma \bullet C_{\rho+1}) \oplus ((\kappa \oplus \delta) \bullet K) = 0]. \quad (2.16)$$

Hier wurden eine  $\rho$ -Runden-Approximation (2.14) und eine Ein-Runden-Approximation (2.15) zu einer  $\rho + 1$ -Runden-Approximation (2.16) kombiniert. Wie zuvor erläutert, haben bei der Summation auftretende Schlüsselbits betragsmäßig keinen Einfluss auf den Bias des Gesamtausdrucks. Jedoch benötigen wir nun ein Mittel zur Berechnung der Wahrscheinlichkeit von Ausdrücken des Typs (2.16) – und damit mittelbar auch deren Bias – aus den Wahrscheinlichkeiten der Einzelapproximationen. Ein solches Werkzeug liefert Matsui so genanntes *Piling-Up-Lemma*:

**Lemma 2.28** (Piling-Up-Lemma, [27] Lemma 3). *Sind  $X_1, X_2, \dots, X_n$  unabhängige binäre Zufallsvariablen mit  $\Pr(X_i = 0) = p_i$  für  $1 \leq i \leq n$ , so ist*

$$\Pr(X_1 \oplus X_2 \oplus \dots \oplus X_n = 0) = \frac{1}{2} + 2^{n-1} \prod_{i=1}^n \left( p_i - \frac{1}{2} \right). \quad (2.17)$$

*Beweis.* Induktion nach  $n$ : Die Behauptung gilt für  $n = 1$ , denn

$$\Pr(X_1 = 0) = \frac{1}{2} + 2^0 \prod_{i=1}^1 \left( p_i - \frac{1}{2} \right) = p_1$$

nach Voraussetzung. Sei nun die Behauptung wahr für  $n \geq 1$ . Da alle  $X_i$  unabhängig sind, folgt

$$\begin{aligned} \Pr(X_1 \oplus \dots \oplus X_n \oplus X_{n+1} = 0) &= \Pr(X_1 \oplus \dots \oplus X_n = 0, X_{n+1} = 0) \\ &\quad + \Pr(X_1 \oplus \dots \oplus X_n = 1, X_{n+1} = 1) \\ &= \Pr(X_1 \oplus \dots \oplus X_n = 0) \cdot \Pr(X_{n+1} = 0) \\ &\quad + \Pr(X_1 \oplus \dots \oplus X_n = 1) \cdot \Pr(X_{n+1} = 1) \end{aligned}$$

---

<sup>4</sup>Biham erläutert in [2], wie der Formalismus der differentiellen auf die lineare Kryptoanalyse übertragen werden kann, indem Approximationen als lineare Charakteristiken formuliert werden.

und mit der Induktionsannahme

$$\begin{aligned}
&= \left( \frac{1}{2} + 2^{n-1} \prod_{i=1}^n \left( p_i - \frac{1}{2} \right) \right) \cdot p_{n+1} \\
&\quad + \left( 1 - \frac{1}{2} - 2^{n-1} \prod_{i=1}^n \left( p_i - \frac{1}{2} \right) \right) \cdot (1 - p_{n+1}) \\
&= \frac{1}{2} + 2 \cdot 2^{n-1} \prod_{i=1}^n \left( p_i - \frac{1}{2} \right) \cdot p_{n+1} \\
&\quad - 2^{n-1} \prod_{i=1}^n \left( p_i - \frac{1}{2} \right) \\
&= \frac{1}{2} + 2^n \prod_{i=1}^n \left( p_i - \frac{1}{2} \right) \cdot \left( p_{n+1} - \frac{1}{2} \right) \\
&= \frac{1}{2} + 2^n \prod_{i=1}^{n+1} \left( p_i - \frac{1}{2} \right),
\end{aligned}$$

und somit gilt die Behauptung für alle  $n$ .  $\square$

Da  $p_i - \frac{1}{2}$  gerade dem Bias von  $X_i$  entspricht, kann das Produkt in (2.17) alternativ auch über die Biase der  $X_i$  gebildet werden. Für den Bias der Summe ergibt sich damit unmittelbar

**Korollar 2.29.** *Seien wie in vorigem Lemma  $X_1, X_2, \dots, X_n$  unabhängige binäre Zufallsvariablen. Bezeichnet  $\epsilon_i$  den Bias von  $X_i$ , gilt also  $\Pr(X_i = 0) = \frac{1}{2} + \epsilon_i$  für  $i = 1, \dots, n$  und  $-\frac{1}{2} \leq \epsilon_i \leq \frac{1}{2}$ , so ist der Bias der Zufallsvariablen  $Z = X_1 \oplus X_2 \oplus \dots \oplus X_n$*

$$\epsilon_Z = 2^{n-1} \prod_{i=1}^n \epsilon_i. \quad (2.18)$$

$\square$

Unter Vernachlässigung der Schlüsselbits suchen wir für die Bestimmung der Wahrscheinlichkeit der Konkatenierung der Approximationen (2.14) und (2.15) von Seite 35 also  $\Pr(Z_1 \oplus Z_2 = 0)$ , wobei  $Z_1 = (\alpha \bullet P) \oplus (\beta \bullet C_\rho)$  und  $Z_2 = (\beta \bullet C_\rho) \oplus (\gamma \bullet C_{\rho+1})$ . Diese beiden Zufallsvariablen sind offensichtlich nicht unabhängig. Dennoch wird in der linearen Kryptoanalyse das Piling-Up-Lemma zur Bestimmung von  $\Pr(Z_1 \oplus Z_2 = 0)$  verwendet, da die Unabhängigkeitsannahme in der Praxis zumeist zu guten Resultaten führt. Sind nun  $p_1, p_2$  die Wahrscheinlichkeiten, dass  $Z_1 = 0$  resp.  $Z_2 = 0$  gilt, und  $\epsilon_1, \epsilon_2$  die Biase von  $Z_1, Z_2$ , so berechnet sich nach dem Piling-Up-Lemma die Wahrscheinlichkeit  $p_3$  der linearen Approximation (2.16) zu

$$p_3 = \Pr(Z_1 \oplus Z_2 = 0) = \frac{1}{2} + 2 \left( p_1 - \frac{1}{2} \right) \left( p_2 - \frac{1}{2} \right),$$

und deren Bias als

$$\epsilon_3 = 2\epsilon_1\epsilon_2.$$

Auf diese Weise können also lineare Approximationen für beliebig viele Runden konstruiert und deren Wahrscheinlichkeiten bzw. Biase aus denjenigen der Komponenten bestimmt werden.

Im Folgenden soll nun ein auf einer  $r - 1$ -Runden-Approximation basierender linearer Angriff beschrieben werden. Es gibt auch andere Spielarten, bei denen die Approximation weniger als  $r - 1$  Runden umspannt oder erst in der  $m$ -ten Runde beginnt; ein  $r - 1$ -Runden-Angriff ist jedoch am gängigsten.

Angenommen, es liegt also eine lineare Approximation für die ersten  $r - 1$  Runden vor, wobei Schlüsselbits vernachlässigt werden:

$$L_{r-1} : (\alpha \bullet P) \oplus (\beta \bullet C_{r-1}) = 0, \quad (2.19)$$

und die Wahrscheinlichkeit von  $L_{r-1}$  sei  $\frac{1}{2} + \epsilon$ . Des Weiteren befindet sich der Angreifer im Besitz einer genügenden Anzahl von Klartext-Chiffretext-Paaren  $(P, C)$ . Für jedes Paar wird der Chiffretext für alle möglichen Werte des Rundenschlüssels  $K^r$  oder eines Teils davon eine Runde entschlüsselt. Gegebenenfalls wird diese Operation nur partiell durchgeführt, d.h. nur auf denjenigen Bits, die durch  $\beta$  ausgewählt sind. Für jeden angenommenen Wert  $k$  von  $K^r$  entsteht also die Approximation

$$L'_{r-1} : (\alpha \bullet P) \oplus (\beta \bullet g^{-1}(C_r, k)) = 0, \quad (2.20)$$

und wenn diese gilt, so wird ein Zähler für den Schlüssel  $k$  erhöht. Nachdem dieses Verfahren für alle vorliegenden Paare durchgeführt wurde, wird derjenige Schlüssel, dessen Zählwert am meisten von der Hälfte der Paaranzahl abweicht, als durch den Angriff bestimmter Wert (eines Teils) des Rundenschlüssels  $K^r$  gewählt.

Dieses Auswahlkriterium ist motiviert durch die Annahme, dass die Entschlüsselungsoperation mit falschen Schlüsseln in etwa gleichverteilte Werte für  $C_{r-1}$  ergeben wird. Die Approximation  $L'_{r-1}$  gilt dann mit Wahrscheinlichkeit  $\approx \frac{1}{2}$  (also mit Bias 0), während sie für den korrekten Schlüssel mit Wahrscheinlichkeit  $\frac{1}{2} + \epsilon$  gilt. Die Zähler der meisten Schlüsselkandidaten sollten bei  $m$  Klartext-Chiffretext-Paaren also ungefähr den Wert  $m/2$  annehmen, während für den korrekten Schlüssel etwa der Wert  $m/2 \pm \epsilon \cdot m$  erwartet wird. Der Zählwert mit dem *betragsmäßig* größten Abstand zu  $m/2$  muss gewählt werden, da die implizit an der Approximation teilhabenden Schlüsselbits aus einer linearen eine affine Approximation machen können und umgekehrt. Eine algorithmische Formulierung des Vorgehens findet sich in Algorithmus 2.2 auf der nächsten Seite. In Matsuis Terminologie entspricht dies seinem „Algorithmus 2“ [27].

Es verbleibt noch zu klären, welche Anzahl von bekannten Klartexten erforderlich ist, damit ein linearer Angriff erfolgreich durchgeführt werden kann. Nach Matsui ([27] Abschnitt 6) ist die benötigte Anzahl proportional zu  $|\epsilon|^{-2}$ , wobei  $\epsilon$  den Bias der verwendeten linearen Approximation bezeichnet. Es kann also davon ausgegangen werden, dass der Angriff bei

$$N \approx c \cdot |\epsilon|^{-2} \quad (2.21)$$

---

**Algorithmus 2.2** Auf einer  $r - 1$ -Runden-Approximation basierender linearer Angriff

---

**Input:**  $r - 1$ -Runden-Approximation  $L : (\alpha \bullet P) \oplus (\beta \bullet C_{r-1}) = 0$  mit Bias  $\epsilon$ .

**Input:**  $m$  Klartext-Chiffretext-Paare  $(P, C)$ .

**Output:** Rundenschlüssel  $K^r$  oder ein Teil davon.

▷ Schlüsselzählphase.

```
1: for all vorliegende Paare  $(P, C)$  do
2:   for all Schlüsselkandidaten  $k$  do
3:     Berechne (ggf. partiell)  $C_{r-1} = g^{-1}(C, k)$ .
4:     if  $(\alpha \bullet P) \oplus (\beta \bullet C_{r-1}) = 0$  then
5:       Inkrementiere Zähler  $count[k]$ .
6:     end if
7:   end for
8: end for
▷ Bestimme  $k$  mit  $\max_{k'} |count[k'] - m/2|$ .
9:  $k \leftarrow 0$ 
10: for all Schlüsselkandidaten  $k'$  do
11:   if  $|count[k'] - m/2| > |count[k] - m/2|$  then
12:      $k \leftarrow k'$ 
13:   end if
14: end for
15: return  $k$ 
```

---

vorliegenden bekannten Klartexten mit hoher Wahrscheinlichkeit erfolgreich sein wird ( $c$  bezeichnet eine experimentell zu ermittelnde kleine positive Konstante). Dabei ist zu beachten, dass im Gegensatz zur differentiellen Kryptoanalyse lediglich bekannte, keine nach bestimmten Eigenschaften *gewählten* Klartexte benötigt werden. Beispiele für lineare Angriffe sind in den Abschnitten 4.5 und 4.6 zu finden.

## 2.4 Linearitätsmaße Boole'scher Abbildungen

Sowohl die differentielle als auch die lineare Kryptoanalyse basieren darauf, Ungleichverteilungen im Eingabe-/Ausgabeverhalten der schlüsselabhängigen Rundenfunktion einer Blockchiffre auszunutzen. Diese Ungleichverteilung, und damit die Erfolgsaussichten beider Angriffe, hängen entscheidend davon ab, wie nahe die Rundenfunktion der Linearität kommt. Lineare oder affine Boole'sche Abbildungen haben besonders ungleich verteilte Differenzen- bzw. lineare Approximationstabellen und sind folglich durch beide Verfahren trivial angreifbar. Beim Entwurf einer Rundenfunktion ist es also von eminenter Bedeutung, diese möglichst nichtlinear zu gestalten – sowohl auf Grund der differentiellen und linearen Kryptoanalyse, als auch, um keine Darstellung der Blockchiffre als großes lineares oder fast lineares Gleichungssystem zu ermöglichen, aus dem dann mit Hilfe von bekannten Klartext-Chiffretextpaaren der Schlüssel bestimmt werden kann.

Unter einem **Linearitätsmaß** verstehen wir dabei eine Vorschrift, welcher einer Boole'schen Abbildung eine Größe zuordnet (in aller Regel eine natürliche oder reelle Zahl), anhand derer sich die Linearitätsnähe der Abbildung beurteilen lässt. Vergleichsmaßstab ist folglich stets das Maß gleichdimensionaler linearer oder affiner Abbildungen.

In diesem Abschnitt sollen neben dem so genannten „algebraischen Grad“ die der differentiellen und linearen Kryptoanalyse zu Grunde liegenden Linearitätsmaße eingeführt werden. Wie in der Literatur üblich, und im Gegensatz zur im Zusammenhang mit T-Funktionen üblichen Notation, verwenden wir hier die algebraische Schreibweise: Die Addition in  $\mathbb{F}_2$  sowie im Vektorraum  $\mathbb{F}_2^n$  wird mit  $+$  bezeichnet, was insbesondere eine übersichtlichere Darstellung (mehrfacher) Summationen erlaubt. Da 0 und 1 in  $\mathbb{F}_2$  ihre eigenen additiven Inversen sind, gilt  $a + b = a - b$  für alle  $a, b \in \mathbb{F}_2$  und damit auch für die Vektoraddition und -subtraktion in  $\mathbb{F}_2^n$ . Das kanonische Skalarprodukt in  $\mathbb{F}_2^n$  notieren wir als

$$a \bullet b := \sum_{i=0}^{n-1} a_i b_i.$$

### 2.4.1 Algebraische Normalform und algebraischer Grad

Bei Polynomen lassen sich lineare und affine Abbildungen besonders leicht von nichtlinearen abgrenzen: Erstere haben maximal den Grad 1, alle höheren Grade kennzeichnen nichtlineare Abbildungen. Bevor wir nun erwägen können, den polynomiellen Grad einer Abbildung als Linearitätsmaß zu nutzen, muss die Frage beantwortet werden, welche Boole'schen Funktionen und Abbildungen sich als Polynom ausdrücken lassen. Diese Fragestellung führt direkt auf die so genannte algebraische Normalform:

**Satz 2.30** (Algebraische Normalform, [30] Satz 1.2, [14] S. 42). *Jede Boole'sche Funktion*

$$f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$$

*lässt sich eindeutig als Polynom in  $n$  Unbestimmten schreiben, dessen sämtliche partiellen Grade<sup>5</sup> kleiner oder gleich Eins sind:*

$$f(x_0, \dots, x_{n-1}) = \sum_{(i_0, \dots, i_{n-1})=(0, \dots, 0)}^{(1, \dots, 1)} a_{(i_0, \dots, i_{n-1})} x_0^{i_0} \cdots x_{n-1}^{i_{n-1}}, \quad (2.22)$$

wobei  $i_0, \dots, i_{n-1} \in \mathbb{F}_2$ ,  $a_* \in \mathbb{F}_2$  und  $x^0 = 1$  sowie  $x^1 = x$ . Der Ausdruck (2.22) heißt die algebraische Normalform (ANF) von  $f$ . □

Dies lässt sich sofort auf (vektorwertige) Boole'sche Abbildungen übertragen:

**Korollar 2.31.** *Jede Boole'sche Abbildung*

$$f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q$$

*wird durch ein Tupel  $(p_0, \dots, p_{q-1})$  von Polynomen in  $n$  Unbestimmten mit Koeffizienten aus  $\mathbb{F}_2$  beschrieben, deren sämtliche partiellen Grade kleiner oder gleich Eins sind.* □

---

<sup>5</sup>Grad in einer einzelnen Unbestimmten.

Damit ist klar, dass sich jede Boole'sche Abbildung durch Polynome beschreiben lässt, indem man sie in algebraischer Normalform ausdrückt. Als ersten Ansatz für ein Linearitätsmaß können wir folglich den polynomiellen Grad der ANF einer Funktion betrachten:

**Definition 2.32** (Algebraischer Grad). Der Grad einer Boole'schen Funktion  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$  als Polynom:

$$\deg f = \max_{a_{(i_0, \dots, i_{n-1})} \neq 0} \#\{j \mid i_j = 1\},$$

wird als **algebraischer Grad** von  $f$  bezeichnet. Der algebraische Grad einer Boole'schen Abbildung  $g : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q$  ist das Maximum der Grade der Komponentenpolynome  $p_0, \dots, p_{q-1}$ .  $\square$

Wir illustrieren diese Begriffe durch ein Beispiel:

**Beispiel 2.33.** Die Boole'sche Funktion

$$\begin{aligned} f : \mathbb{F}_2^2 &\rightarrow \mathbb{F}_2 \\ (x_0, x_1) &\mapsto (\neg x_0) \wedge x_1 \end{aligned}$$

besitzt die Wahrheitstabelle

$x_0$	$x_1$	$f$
0	0	0
0	1	1
1	0	0
1	1	0

aus der sich durch Interpolation die folgenden ANF-Koeffizienten ableiten lassen:

Monom	$\hat{=}$ Koeffizient	Wert
1	$a_{00}$	0
$x_1$	$a_{01}$	1
$x_0$	$a_{10}$	0
$x_0x_1$	$a_{11}$	1

was der algebraischen Normalform  $f(x_0, x_1) = x_1 + x_0x_1$  entspricht. Es lässt sich leicht verifizieren, dass dieses Polynom tatsächlich die gleiche Funktion beschreibt. Der algebraische Grad von  $f$  ist demnach 2, da die ANF von  $f$  das Monom  $x_0x_1$  enthält.

Analog ergibt sich die algebraische Normalform der Boole'schen Abbildung

$$\begin{aligned} g : \mathbb{F}_2^2 &\rightarrow \mathbb{F}_2^2 \\ \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} &\mapsto \begin{pmatrix} \neg x_0 \\ (\neg x_0) \wedge x_1 \end{pmatrix} \end{aligned}$$

als ein Zweitupel von Polynomen, nämlich  $(p_0, p_1) = (1 + x_0, x_1 + x_0x_1)$ . Der algebraische Grad von  $g$  ist definitionsgemäß das Maximum der Grade von  $p_0$  und  $p_1$ , also ebenfalls 2.  $\square$

Wie in [30] angemerkt, ist der algebraische Grad als Linearitätsmaß insofern tauglich, als dass sich die Nullstellen von Funktionen mit hohem algebraischem Grad im Allgemeinen schwer bestimmen lassen, was damit auch das Lösen von Gleichungssystemen, in denen diese Funktionen vorkommen, erschwert. Jedoch lässt sich aus einem hohen algebraischen Grad nicht zwangsläufig auf hohe Komplexität der Nullstellenbestimmung schließen, wie man am Beispiel der Funktion

$$f(x_0, \dots, x_{n-1}) = x_0 \cdots x_{n-1}$$

sehen kann, welche zwar maximalen algebraischen Grad  $n$  besitzt, wo aber sofort alle Vektoren des  $\mathbb{F}_2^n$  bis auf  $(1, \dots, 1)$  als Nullstellen von  $f$  identifiziert werden können.

## 2.4.2 Die Walsh-Hadamard-Transformation

Zur Betrachtung der Linearitätsmaße der differentiellen und linearen Kryptoanalyse sehr nützlich erweist sich die *Walsh-Hadamard-Transformation*; sowohl aus theoretischer Sicht als auch zur praktischen Berechnung dieser Linearitätsmaße.

### 2.4.2.1 Transformation und Faltung

Die Walsh-Hadamard-Transformation ist eine spezielle Variante der diskreten Fourier-Transformation (vgl. [30] sowie [7], Kapitel 30). Sie kann allgemein für reellwertige Funktionen mit Definitionsbereich  $\mathbb{F}_2^n$  erklärt werden. Die Menge solcher Funktionen  $\phi : \mathbb{F}_2^n \rightarrow \mathbb{R}$  bezeichnen wir mit  $\mathbb{R}^{\mathbb{F}_2^n}$ .

**Definition 2.34** (Walsh-Transformation). Die Walsh-Hadamard-Transformation (oder kurz Walsh-Transformation)

$$\begin{aligned} \mathcal{W} : \mathbb{R}^{\mathbb{F}_2^n} &\rightarrow \mathbb{R}^{\mathbb{F}_2^n} \\ f &\mapsto \widehat{f} \end{aligned}$$

ist definiert durch

$$\mathcal{W}(f)(u) = \widehat{f}(u) := \sum_{x \in \mathbb{F}_2^n} f(x) \cdot (-1)^{u \bullet x}. \quad (2.23)$$

□

Die Transformation Boole'scher Funktionen wird dabei über die kanonische Einbettung des  $\mathbb{F}_2$  in  $\mathbb{R}$  erfasst. Wegen

$$\begin{aligned} \widehat{\alpha f + \beta g}(u) &= \sum_{x \in \mathbb{F}_2^n} (\alpha f + \beta g)(x) \cdot (-1)^{u \bullet x} \\ &= \sum_{x \in \mathbb{F}_2^n} (\alpha f(x) + \beta g(x)) \cdot (-1)^{u \bullet x} \\ &= \alpha \sum_{x \in \mathbb{F}_2^n} f(x) \cdot (-1)^{u \bullet x} + \beta \sum_{x \in \mathbb{F}_2^n} g(x) \cdot (-1)^{u \bullet x} \\ &= \alpha \widehat{f}(u) + \beta \widehat{g}(u) \end{aligned}$$

für zwei Funktionen  $f, g : \mathbb{F}_2^n \rightarrow \mathbb{R}$  und  $\alpha, \beta \in \mathbb{R}$  ist klar:

**Satz 2.35.**  $\mathcal{W}$  ist eine  $\mathbb{R}$ -lineare Abbildung. □

Es stellt sich heraus, dass die Walsh-Transformation eine Bijektion ist, also eine Rücktransformation  $\widehat{f} \mapsto f$  existiert:

**Satz 2.36** (Rücktransformation). *Die Umkehrung der Walsh-Transformation ist gegeben durch*

$$\mathcal{W}^{-1} = \frac{1}{2^n} \mathcal{W}.$$

*Beweis.* Siehe [30], Satz 2.1. □

Aus der Umkehrformel lässt sich die folgende nützliche Identität ableiten:

**Korollar 2.37.** *Für eine Funktion  $f : \mathbb{F}_2^n \rightarrow \mathbb{R}$  gilt*

$$f(0) = \frac{1}{2^n} \sum_{u \in \mathbb{F}_2^n} \widehat{f}(u). \quad (2.24)$$

*Beweis.* Nach der Umkehrformel gilt

$$\begin{aligned} f(0) &= \frac{1}{2^n} \widehat{f}(0) \\ &= \frac{1}{2^n} \sum_{u \in \mathbb{F}_2^n} \widehat{f}(u) \cdot \underbrace{(-1)^{u \bullet 0}}_{=1 \text{ für alle } u} \\ &= \frac{1}{2^n} \sum_{u \in \mathbb{F}_2^n} \widehat{f}(u), \end{aligned}$$

wie behauptet. □

Ähnlich wie bei der Fourier-Transformation ist die Transformierte der so genannten *Faltung* von  $f$  und  $g$  das Produkt der Transformierten  $\widehat{f}, \widehat{g}$ :

**Definition 2.38** (Faltung). Für zwei Funktionen  $f, g : \mathbb{F}_2^n \rightarrow \mathbb{R}$  heißt  $f * g : \mathbb{F}_2^n \rightarrow \mathbb{R}$  mit

$$f * g(w) := \sum_{x \in \mathbb{F}_2^n} f(x)g(w - x) \quad (2.25)$$

Faltung von  $f$  und  $g$ . □

**Satz 2.39** (Faltungssatz). *Für  $f, g : \mathbb{F}_2^n \rightarrow \mathbb{R}$  gilt*

$$\widehat{f * g} = \widehat{f} \cdot \widehat{g}.$$

*Beweis.* Siehe [30], Satz 2.3. □

Die Faltung wird bei der effizienten praktischen Berechnung von Linearitätsmaßen noch eine sehr nützliche Rolle spielen.

### 2.4.2.2 Schnelle Walsh-Transformation

Gegenstand dieses Abschnitts ist die Entwicklung einer effizienten Berechnungsmethode für die Walsh-Transformierte einer Funktion  $f : \mathbb{F}_2^n \rightarrow \mathbb{R}$ . Wir gehen dabei davon aus, dass die vollständige Walsh-Transformierte errechnet werden soll, also  $\widehat{f}(u)$  für alle Argumente  $u$ . Dabei ist  $f$  durch seine  $2^n$  Elemente umfassende Wertetabelle gegeben; gesucht ist demnach die gleich große Wertetabelle von  $\widehat{f}$ . (Soll die Walsh-Transformierte von  $f$  nur an einigen Punkten ausgewertet werden, können Spezialverfahren zum Einsatz kommen, siehe etwa [12].)

Die naive Berechnung der Walsh-Transformation direkt nach der Definition erfordert  $2^n$  Auswertungen der Formel (2.23), hat also eine Laufzeit von  $\Theta(2^{2n})$ , was quadratisch in der Eingabegröße ist.<sup>6</sup> Analog zur schnellen Fourier-Transformation ([7], Kap. 30) kann man die Laufzeit mit einem *Divide-and-Conquer*-Ansatz auf  $\Theta(n \cdot 2^n)$  reduzieren, ein solcher Algorithmus wird *Schnelle Walsh-Transformation* (englisch *Fast Walsh Transform* (FWT)) genannt. Die von Pommerening in [30] vorgestellte Variante hat die erwähnte Laufzeitkomplexität, erfordert jedoch zusätzlich zur Wahrheitstabelle von  $f$  weitere  $2^n$  Speicherplätze. Für unsere Zwecke können die zu verarbeitenden Tabellen gelegentlich bis zu  $2^{28}$  Bytes groß werden, so dass eine Implementierung der schnellen Walsh-Transformation erforderlich ist, welche nur  $\mathcal{O}(1)$  zusätzlichen Speicher benötigt und zudem „in-place“ arbeitet, d.h. die Wahrheitstabelle von  $f$  direkt mit derjenigen von  $\widehat{f}$  überschreibt.

Eine solche lässt sich auf dem folgenden Weg erhalten: Wir definieren für  $u \in \mathbb{F}_2^i, 0 \leq i \leq n$  die Funktion

$$\begin{aligned} f^{(u)} : \mathbb{F}_2^{n-i} &\rightarrow \mathbb{R} \\ x &\mapsto f(u, x), \end{aligned}$$

welche gerade  $f$  entspricht, wobei die ersten (höheren)  $i$  Bits des Arguments konstant auf  $u$  gesetzt sind. Für  $i = 0$  entspricht  $f^{(u)}$  gerade  $f$ , für  $i = n$  ergibt sich die Konstante  $f^{(u)}() = f(u)$ . Da für einen Vektor  $v \in \mathbb{F}_2^{n-i}$  entweder  $v = 0w$  oder  $v = 1w$  mit  $w \in \mathbb{F}_2^{n-i-1}$  gelten muss, lässt sich die Walsh-Transformierte  $\widehat{f^{(u)}}$  im Punkt  $v$  berechnen als

$$\begin{aligned} \widehat{f^{(u)}}(v) &= \widehat{f^{(u)}}(0w) = \sum_{\zeta \in \mathbb{F}_2} \sum_{z \in \mathbb{F}_2^{n-i-1}} (-1)^{0 \cdot \zeta + w \bullet z} f^{(u)}(\zeta, z) \\ &= \sum_{z \in \mathbb{F}_2^{n-i-1}} (-1)^{w \bullet z} f^{(u)}(0, z) + \sum_{z \in \mathbb{F}_2^{n-i-1}} (-1)^{w \bullet z} f^{(u)}(1, z) \\ &= \widehat{f^{(u0)}}(w) + \widehat{f^{(u1)}}(w), \end{aligned} \tag{2.26}$$

<sup>6</sup> $\Theta(g(n))$  bezeichnet die Menge der Funktionen  $f$ , welche asymptotisch bis auf einen konstanten Faktor genauso schnell wachsen wie  $g$ :

$$\Theta(g(n)) = \{f(n) \mid \exists c_1, c_2, N_0 > 0. \forall n \geq N_0. 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}.$$

falls  $v = 0w$ , beziehungsweise

$$\begin{aligned}
\widehat{f^{(u)}}(v) &= \widehat{f^{(u)}}(1w) = \sum_{\zeta \in \mathbb{F}_2} \sum_{z \in \mathbb{F}_2^{n-i-1}} (-1)^{1 \cdot \zeta + w \bullet z} f^{(u)}(\zeta, z) \\
&= \sum_{z \in \mathbb{F}_2^{n-i-1}} (-1)^{w \bullet z} f^{(u)}(0, z) - \sum_{z \in \mathbb{F}_2^{n-i-1}} (-1)^{w \bullet z} f^{(u)}(1, z) \\
&= \widehat{f^{(u0)}}(w) - \widehat{f^{(u1)}}(w), \tag{2.27}
\end{aligned}$$

falls  $v = 1w$ . Die Berechnung von  $\widehat{f}(v) = \widehat{f^{(0)}}(v)$  für den  $n$ -Bit-Vektor  $v = \zeta w$  kann also auf zwei Teilprobleme der Größe  $n - 1$  zurückgeführt werden, welche selbst wieder die gleiche Struktur wie das Ausgangsproblem besitzen, denn die Transformaten  $\widehat{f^{(0)}}(w)$  und  $\widehat{f^{(1)}}(w)$  können je nach Wert des vordersten Bits von  $w$  wiederum nach Formel (2.26) oder (2.27) bestimmt werden usw. Dabei bricht die Rekursion ab, sobald  $v$  nur noch aus einem Bit besteht, also  $u \in \mathbb{F}_2^{n-1}$  ist:

$$\begin{aligned}
\widehat{f^{(u)}}(0) &= \sum_{\zeta \in \mathbb{F}_2} (-1)^{0 \cdot \zeta} f^{(u)}(\zeta) \\
&= f^{(u)}(0) + f^{(u)}(1) \\
&= f^{(u0)}() + f^{(u1)}(),
\end{aligned}$$

beziehungsweise

$$\begin{aligned}
\widehat{f^{(u)}}(1) &= \sum_{\zeta \in \mathbb{F}_2} (-1)^{1 \cdot \zeta} f^{(u)}(\zeta) \\
&= f^{(u)}(0) - f^{(u)}(1) \\
&= f^{(u0)}() - f^{(u1)}().
\end{aligned}$$

Zum Zweck der Einheitlichkeit setzen wir  $\widehat{f^{(u)}}() := f^{(u)}$ .

Es ist zu beachten, dass eine direkte Implementierung dieser Rekursionsformeln noch keine Effizienzsteigerung ergibt: Für ein festes  $v \in \mathbb{F}_2^n$  ist die Laufzeit  $T(n) = 2T(n - 1)$ , also  $T(n) = \Theta(2^n)$ , wie mit Induktion sofort einzusehen ist. Jedoch fällt auf, dass die Berechnung von  $\widehat{f^{(u)}}(0w)$  sowie  $\widehat{f^{(u)}}(1w)$  auf die gleichen Teilprobleme zurückzuführen ist, die nur mit unterschiedlichem Vorzeichen kombiniert werden. In den Teilproblemen selbst gilt diese Eigenschaft erneut, usw. Zum Beispiel würde für  $n = 3$  und  $v = 010$  der folgende Rekursionsbaum entstehen:

$$\begin{array}{c}
\widehat{f^{(010)}} \\
\hline
\widehat{f^{(0)}}(10) \quad + \quad \widehat{f^{(1)}}(10) \\
\hline
\begin{array}{cc}
\widehat{f^{(00)}}(0) & - & \widehat{f^{(01)}}(0) \\
\hline
\widehat{f^{(000)}}() + \widehat{f^{(001)}}() & & \widehat{f^{(010)}}() + \widehat{f^{(011)}}()
\end{array}
\quad + \quad
\begin{array}{cc}
\widehat{f^{(10)}}(0) & - & \widehat{f^{(11)}}(0) \\
\hline
\widehat{f^{(100)}}() + \widehat{f^{(101)}}() & & \widehat{f^{(110)}}() + \widehat{f^{(111)}}()
\end{array}
\end{array}$$

während er für  $v' = 110$  so aussieht:

$$\frac{\frac{\widehat{f^{(00)}}(0)}{\widehat{f^{(000)}}() + \widehat{f^{(001)}}()} - \frac{\widehat{f^{(01)}}(0)}{\widehat{f^{(010)}}() + \widehat{f^{(011)}}()}}{\widehat{f^{(0)}}(10)} - \frac{\frac{\widehat{f^{(10)}}(0)}{\widehat{f^{(100)}}() + \widehat{f^{(101)}}()} - \frac{\widehat{f^{(11)}}(0)}{\widehat{f^{(110)}}() + \widehat{f^{(111)}}()}}{\widehat{f^{(1)}}(10)}}{\widehat{f^{(110)}}$$

Die kompletten Teilbäume unterhalb der Wurzel sind identisch, lediglich wird in der ersten Ebene entsprechend (2.27) subtrahiert statt addiert. Da diese Beobachtung wegen der Uniformität der Teilprobleme auf allen Ebenen gültig ist, würden bei einer direkten rekursiven Implementierung der Formeln viele Teilprobleme mehrfach berechnet. Um das zu vermeiden, können wir die Ebenen des Rekursionsbaums iterativ einzeln von unten nach oben berechnen, bis wir in der Wurzel (also bei  $\widehat{f}(v)$ ) angekommen sind. Da auf jeder Ebene lediglich die Teilergebnisse der darunterliegenden genutzt werden (siehe dazu auch die Rekursionsformeln (2.26) bzw. (2.27)), müssen beim Schritt von Ebene  $k$  nach Ebene  $k - 1$  keine Zwischenergebnisse aus Ebene  $k - 2$  mehr vorhanden sein. Ist die Wertetabelle von  $f$ , und damit diejenige von  $f^{(u)}()$ , so in einem Array  $A[0 \dots 2^{n-1}]$  gegeben, dass  $A[v] = f(v)$  gilt (also die Indizes den Argumenten entsprechen), dann können wir die Wertetabelle von  $\widehat{f}$  auf folgende Weise erhalten:

$$\begin{array}{c} A_3 = \begin{array}{|c|c|c|c|c|} \hline \widehat{f^{(0)}}(000) & \widehat{f^{(0)}}(001) & \dots & \widehat{f^{(0)}}(110) & \widehat{f^{(0)}}(111) \\ \hline 0 & 1 & & 6 & 7 \\ \hline \end{array} \\ \uparrow \\ A_2 = \begin{array}{|c|c|c|c|c|} \hline \widehat{f^{(0)}}(00) & \widehat{f^{(0)}}(01) & \dots & \widehat{f^{(1)}}(10) & \widehat{f^{(1)}}(11) \\ \hline 0 & 1 & & 6 & 7 \\ \hline \end{array} \\ \uparrow \\ A_1 = \begin{array}{|c|c|c|c|c|} \hline \widehat{f^{(00)}}(0) & \widehat{f^{(00)}}(1) & \dots & \widehat{f^{(11)}}(0) & \widehat{f^{(11)}}(1) \\ \hline 0 & 1 & & 6 & 7 \\ \hline \end{array} \\ \uparrow \\ A_0 = \begin{array}{|c|c|c|c|c|} \hline \widehat{f^{(000)}}() & \widehat{f^{(001)}}() & \dots & \widehat{f^{(110)}}() & \widehat{f^{(111)}}() \\ \hline 0 & 1 & & 6 & 7 \\ \hline \end{array} \end{array}$$

(hier im Beispiel für  $n = 3$ ). Nach der Definition von  $f^{(u)}$  beginnen wir also unten mit der Wahrheitstabelle von  $f$  und erhalten nach  $n$  Schritten die Wahrheitstabelle von  $\widehat{f}$ . Man kann dieses Schema auch so interpretieren, dass es alle zu den Einträgen  $\widehat{f}(v)$  gehörende Rekursionsbäume enthält, jedoch in derart komprimierter Form, dass identische Teilprobleme nur einmal berechnet und dann abgespeichert werden.

Der Schritt von einer Ebene zur vorherigen kann dabei anhand der Rekursionsformeln geschehen: Als ganze Zahlen interpretiert entspricht  $u1w = u0w + 2^k$ , wenn  $w \in \mathbb{F}_2^k$ . Die Elemente  $\widehat{f^{(u)}}(0)$  in  $A_1$ , also diejenigen an geraden Indizes, ergeben sich nach (2.26) als

$$A_1[u0] = f(u0) + f(u1) = A_0[u0] + A_0[u0 + 1];$$

analog gilt für die ungeraden Indizes nach (2.27)

$$A_1[u1] = f(u0) - f(u1) = A_0[u0] - A_0[u0 + 1].$$

In  $A_2$  und  $A_3$  setzt sich dieser Prozess fort, nur beträgt hier der Abstand der zu addierenden bzw. subtrahierenden Elemente  $2^1$  bzw.  $2^2$ .

Im Schritt von  $A_k$  zu  $A_{k+1}$  werden folglich jeweils  $2^{n-1}$  verschiedene Paare à 2 Elementen im Abstand von  $2^k$  addiert bzw. subtrahiert, insgesamt gibt es  $n$  Schritte. Für den Gesamtaufwand des Verfahrens ergibt sich demnach die Laufzeit  $\Theta(n \cdot 2^n)$  und, da  $A_{k+1}$  jeweils  $A_k$  ersetzen darf, ein Speicherbedarf von  $\mathcal{O}(1)$  zusätzlich zur Wahrheitstabelle. Das sich nach diesen Überlegungen ergebende Verfahren zur schnellen Walsh-Transformation ist in Algorithmus 2.3 dargestellt.

---

### Algorithmus 2.3 In-place arbeitende schnelle Walsh-Transformation

---

**Input:** Wertetabelle  $A$  der Funktion  $f : \mathbb{F}_2^n \rightarrow \mathbb{R}$ .

**Output:** Wertetabelle  $A$  der Walsh-Transformierten  $\hat{f}$  von  $f$ .

```

1: blocklength ← 2
2: pairdiff ← 1
3: while pairdiff <  $2^n$  do
4:    $i_a \leftarrow 0$ 
5:   while  $i_a < 2^n$  do
6:      $i_b \leftarrow i_a + \textit{pairdiff}$ 
7:      $A[i_a] \leftarrow A[i_a] + A[i_b]$ 
8:      $A[i_b] \leftarrow A[i_a] - A[i_b]$ 
9:     if  $i_b \equiv -1 \pmod{\textit{blocklength}}$  then    ▷  $i_b$  am Blockende
10:       $i_a \leftarrow i_b + 1$ 
11:     else    ▷  $i_b$  noch innerhalb Block
12:        $i_a \leftarrow i_a + 1$ 
13:     end if
14:   end while
15:   blocklength ← blocklength · 2
16:   pairdiff ← pairdiff · 2
17: end while

```

---

Der im Algorithmus verwendete Begriff des „Blocks“ lässt sich folgendermaßen erklären: In beispielsweise der zweiten Iteration der äußeren Schleife ist  $\textit{pairdiff} = 2^1 = 2$ , d.h.  $(A[0], A[2])$  sowie  $(A[1], A[3])$  bilden Additions-Subtraktionspaare. Bei  $i_a = 1, i_b = 3$  angekommen ist das nächste zu verarbeitende Paar nicht  $(A[2], A[4])$ , sondern  $(A[4], A[6])$ , da  $A[2]$  und  $A[3]$  bereits mit  $A[0]$  bzw.  $A[1]$  verarbeitet wurden. Bei  $\textit{pairdiff} = 2^k$  sind die  $2^n$  Elemente also in aufeinanderfolgende Blöcke à  $2^{k+1}$  Elementen eingeteilt, die jeweils untereinander gepaart werden. Ist nun der Index  $i_a$  am Ende eines solchen Blocks angekommen, so muss er hinter diesen Block weiterbewegt werden, sonst um eins innerhalb des Blocks. Dies erklärt die Zeilen 9–13 in Algorithmus 2.3. Die relativ aufwändige Modulo-Operation in Zeile 9 kann übrigens auf einen Gleichheitstest reduziert werden,

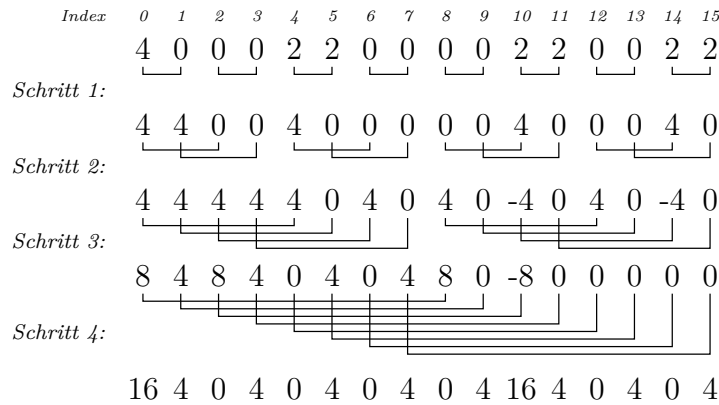


Abbildung 2.8: Exemplarische Durchführung der schnellen Walsh-Transformation mit  $n = 4$ .

wenn an Stelle der Blocklänge  $2^{k+1}$  die „Blockmaske“  $2^{k+1} - 1$  mitgeführt wird, denn dann gilt

$$i_b \equiv -1 \pmod{\text{blocklength}} \Leftrightarrow i_b \mathbf{and} \text{blockmask} = \text{blockmask}$$

(**and** bezeichnet dabei das bitweise logische Und).

**Beispiel 2.40.** Eine beispielhafte Durchführung von Algorithmus 2.3 an einer Funktion  $f : \mathbb{F}_2^4 \rightarrow \mathbb{R}$  enthält Abbildung 2.8. Im Gegensatz zur bisherigen Rekursionsbaumdarstellung sind die Ebenen von oben nach unten in der Iterationsreihenfolge aufgeführt; die erste Zeile enthält also die Wertetabelle von  $f$ , die letzte diejenige von  $\hat{f}$ . Die einzelnen Additions-Subtraktionspaare sind durch Klammerung verdeutlicht.  $\square$

### 2.4.3 Walsh-Spektrum, lineare Approximationstabelle und lineares Potenzial

Die hauptsächlichen (und häufig einzigen) nichtlinearen Komponenten einer Rundenfunktion sind die S-Boxen, welche allgemein Boole'sche Abbildungen  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q$  sind. Um die Nichtlinearitätseigenschaften von S-Boxen bzw. deren  $q$  Komponentenfunktionen  $f_i : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$  zu untersuchen, wurde von Chabaud und Vaudenay in [6] die charakteristische Funktion von  $f$  eingeführt:

**Definition 2.41** (Charakteristische Funktion). Für die Boole'sche Abbildung  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q$  heißt

$$\vartheta_f : \mathbb{F}_2^n \times \mathbb{F}_2^q \rightarrow \mathbb{R}$$

$$\vartheta_f(x, y) := \begin{cases} 1 & \text{wenn } y = f(x), \\ 0 & \text{sonst} \end{cases}$$

charakteristische Funktion von  $f$ .  $\square$

Interessant ist nun, dass die Walsh-Transformierte  $\widehat{\vartheta}_f$  der charakteristischen Funktion von  $f$  in engem Zusammenhang zur im Kontext der linearen Kryptoanalyse (vgl. Abschnitt 2.3.3) eingeführten Menge  $L_f(u, v)$  steht, welche die Güte der durch die Vektoren  $u$  und  $v$  gegebenen linearen Approximation der Funktion  $f$  misst. Es ist

$$L_f(u, v) = \{x \in \mathbb{F}_2^n \mid u \bullet x = v \bullet f(x)\},$$

und der Zusammenhang zu  $\vartheta_f$  ergibt sich als

**Satz 2.42** ([6] Lemma 2, [30] Satz 3.1). *Sei  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q$  eine Boole'sche Abbildung. Dann gilt für die Walsh-Transformierte der charakteristischen Funktion von  $f$ :*

$$\widehat{\vartheta}_f(u, v) = 2 \cdot \#L_f(u, v) - 2^n. \quad (2.28)$$

*Beweis.* Wir haben

$$\begin{aligned} \widehat{\vartheta}_f(u, v) &= \sum_{x \in \mathbb{F}_2^n} \sum_{y \in \mathbb{F}_2^q} \vartheta_f(x, y) \cdot (-1)^{u \bullet x + v \bullet y} \\ &= \sum_{x \in \mathbb{F}_2^n} \sum_{\substack{y \in \mathbb{F}_2^q \\ y = f(x)}} (-1)^{u \bullet x + v \bullet y} \\ &= \sum_{x \in \mathbb{F}_2^n} (-1)^{u \bullet x + v \bullet f(x)}, \end{aligned}$$

und der Term  $(-1)^{u \bullet x + v \bullet f(x)}$  wird genau dann zu  $+1$ , wenn  $u \bullet x = v \bullet f(x)$ , sonst zu  $-1$ . Folglich gilt

$$\begin{aligned} \sum_{x \in \mathbb{F}_2^n} (-1)^{u \bullet x + v \bullet f(x)} &= \#L_f(u, v) - (2^n - \#L_f(u, v)) \\ &= 2 \cdot \#L_f(u, v) - 2^n, \end{aligned}$$

was zu zeigen war. □

Die Walsh-Transformierte der charakteristischen Funktion verdient wegen ihrer im Folgenden deutlich werdenden Bedeutung einen eigenen Namen:

**Definition 2.43** (Walsh-Spektrum). *Sei  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q$  eine Boole'sche Abbildung. Die Walsh-Transformierte ihrer charakteristischen Funktion*

$$\widehat{\vartheta}_f : \mathbb{F}_2^n \times \mathbb{F}_2^q \rightarrow \mathbb{R}$$

heißt **Walsh-Spektrum** (oder kurz **Spektrum**) von  $f$ . □

Das Walsh-Spektrum von  $f$  kann vollständig durch eine  $2^n \times 2^q$ -Matrix  $\mathcal{S}$  dargestellt werden. Anhand des in Satz 2.42 hergestellten Zusammenhangs ist klar, dass das Spektrum wegen  $\text{LAT}_f(u, v) = \#L_f(u, v) - 2^{n-1}$  (vgl. Definition 2.27) bis auf den Faktor 2 der linearen Approximationstabelle entspricht:

$$\widehat{\vartheta}_f = 2 \cdot \text{LAT}_f. \quad (2.29)$$

Der Absolutbetrag eines Eintrags  $\mathcal{S}_{u,v}$  ist damit proportional zur Güte (d.h. dem Abstand von  $\#L_f(u, v)$  zu  $2^{n-1}$ ) der durch  $u, v$  gegebenen linearen Approximation von  $f$ .

**Beispiel 2.44.** Für die bereits aus Beispiel 2.33 bekannte Funktion

$$f : \mathbb{F}_2^2 \rightarrow \mathbb{F}_2$$

$$(x_0, x_1) \mapsto (\neg x_0) \wedge x_1$$

ergeben sich die folgenden Wertetabellen der charakteristischen Funktion  $\vartheta_f$  sowie des Spektrums  $\widehat{\vartheta}_f$ :

$\vartheta_f(x, y)$	$y =$	
$x = 00$	0	1
01	1	0
10	0	1
11	1	0

$\widehat{\vartheta}_f(u, v)$	$v =$	
$u = 00$	0	1
01	4	2
10	0	-2
11	0	2

Für die dem gleichen Beispiel entstammende Funktion

$$g : \mathbb{F}_2^2 \rightarrow \mathbb{F}_2^2$$

$$\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \mapsto \begin{pmatrix} \neg x_0 \\ (\neg x_0) \wedge x_1 \end{pmatrix}$$

sehen charakteristische Funktion  $\vartheta_g$  und Spektrum  $\widehat{\vartheta}_g$  so aus:

$\vartheta_g(x, y)$	$y =$			
$x = 00$	00	01	10	11
01	0	0	1	0
10	0	0	0	1
11	1	0	0	0

$\widehat{\vartheta}_g(u, v)$	$v =$			
$u = 00$	00	01	10	11
01	4	2	0	2
10	0	2	0	-2
11	0	-2	-4	-2

□

Einige Eigenschaften des Spektrums fallen sofort auf:

**Satz 2.45.** Sei  $f : \mathbb{F}_2^n \rightarrow \mathbb{R}$  und  $\widehat{\vartheta}_f$  das Walsh-Spektrum von  $f$ . Dann gilt:

- (a) Alle Werte des Spektrums sind gerade.
- (b) Es gilt  $-2^n \leq \widehat{\vartheta}_f(u, v) \leq 2^n$  für beliebiges  $u, v$ .
- (c)  $\widehat{\vartheta}_f(0, 0) = 2^n$ .
- (d) Der Absolutbetrag jeder Spaltensumme des Spektrums ist  $2^n$ .

*Beweis.* (a) und (b) folgen direkt aus Satz 2.42, (c) wegen

$$L_f(0, 0) = \{x \in \mathbb{F}_2^n \mid 0 \bullet x = 0 \bullet f(x)\} = \mathbb{F}_2^n$$

ebenfalls. Zum Nachweis von (d) definieren wir die Funktion  $\nu : x \mapsto v \bullet x$  sowie das der Boole'schen Funktion  $g$  zugeordnete  $\chi_g : x \mapsto (-1)^{g(x)}$  und erhalten

$$\begin{aligned}
\sum_{u \in \mathbb{F}_2^n} \widehat{\vartheta}_f(u, v) &= \sum_{u \in \mathbb{F}_2^n} \left( \sum_{x \in \mathbb{F}_2^n} \sum_{y \in \mathbb{F}_2^q} \vartheta_f(x, y) \cdot (-1)^{u \bullet x + v \bullet y} \right) \\
&= \sum_{u \in \mathbb{F}_2^n} \sum_{x \in \mathbb{F}_2^n} (-1)^{u \bullet x + v \bullet f(x)} \\
&= \sum_{u \in \mathbb{F}_2^n} \sum_{x \in \mathbb{F}_2^n} (-1)^{(\nu \circ f)(x)} \cdot (-1)^{u \bullet x} \\
&= \sum_{u \in \mathbb{F}_2^n} \widehat{\chi_{\nu \circ f}}(u),
\end{aligned}$$

woraus mit Korollar 2.37 folgt

$$\begin{aligned}
\sum_{u \in \mathbb{F}_2^n} \widehat{\chi_{\nu \circ f}}(u) &= 2^n \cdot \chi_{\nu \circ f}(0) \\
&= 2^n \cdot (-1)^{(\nu \circ f)(0)} \\
&= 2^n \cdot (-1)^{v \bullet f(0)} \\
&= \begin{cases} +2^n & \text{falls } v \bullet f(0) = 0, \\ -2^n & \text{sonst.} \end{cases}
\end{aligned}$$

Damit ist die Behauptung (d) gezeigt. □

Über den erwähnten Zusammenhang (2.29) lassen sich die Beobachtungen (b)–(d) von Satz 2.45 ohne Weiteres auf die lineare Approximationstabelle  $\text{LAT}_f$  übertragen.

Die Spektren affiner Abbildungen  $f(x) = Ax + b$  mit  $A \in \mathbb{F}_2^{q \times n}$ ,  $b \in \mathbb{F}_2^q$  haben eine besondere Struktur:

**Satz 2.46.** *Die Boole'sche Abbildung  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q$  ist genau dann affin, wenn jede Spalte des Spektrums von  $f$  genau einen von Null verschiedenen Eintrag besitzt (nämlich  $\pm 2^n$ ).*

*Beweis.* Siehe [30], Satz 3.2. Dass der von Null verschiedene Eintrag einer Spalte gleich  $\pm 2^n$  sein muss, folgt aus Satz 2.45 (d). □

Aus diesem Zusammenhang wird deutlich, wie das Walsh-Spektrum bzw. die lineare Approximationstabelle als Linearitätsmaß verwendet werden können: Je näher eine Abbildung an der Linearität ist, desto näher werden die Absolutbeträge der Maximal- bzw. Minimaleinträge am für affine Abbildungen charakteristischen  $2^n$  liegen. Dabei ist es, wie wir in Abschnitt 2.3.3 gesehen haben, für die Durchführbarkeit der linearen Kryptoanalyse unerheblich, ob betragsmäßig möglichst große positive oder negative Werte vorliegen. Also ist es zweckmäßig, diese gleich zu behandeln. Nach Matsui [28] sowie Pommerening [30] erhält man

**Definition 2.47** (Linearitätsprofil). Für eine Boole'sche Abbildung  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q$  sowie  $u \in \mathbb{F}_2^n, v \in \mathbb{F}_2^q$  vereinbaren wir die Größen

$$p_f(u, v) := \frac{\#L_f(u, v)}{2^n} \quad (2.30)$$

$$\text{sowie} \quad \lambda_f(u, v) := (2p_f(u, v) - 1)^2. \quad (2.31)$$

$p_f(u, v)$  und  $\lambda_f(u, v)$  heißen **Wahrscheinlichkeit** bzw. **Potenzial** der durch  $u, v$  beschriebenen linearen Approximation von  $f$ . Die auf diese Weise erklärte Funktion

$$\lambda_f : \mathbb{F}_2^n \times \mathbb{F}_2^q \rightarrow \mathbb{Q}$$

nennen wir **Linearitätsprofil** von  $f$ . □

Das Linearitätsprofil entspricht dabei genau dem unnormierten Quadrat des Walsh-Spektrums:

$$\begin{aligned} \lambda_f(u, v) &= (2p_f(u, v) - 1)^2 \\ &= \left( 2 \cdot \frac{\#L_f(u, v)}{2^n} - 1 \right)^2 = \left( \frac{2^n + \widehat{\vartheta}_f(u, v)}{2^n} - 1 \right)^2 = \frac{1}{2^{2n}} \widehat{\vartheta}_f(u, v)^2 \end{aligned}$$

nach Satz 2.42 und der Definition des Linearitätsprofils, also:

**Korollar 2.48.** Sei  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q, u \in \mathbb{F}_2^n, v \in \mathbb{F}_2^q$ . Es gilt

$$\lambda_f(u, v) = \frac{1}{2^{2n}} \widehat{\vartheta}_f(u, v)^2.$$

□

Damit „vererben“ sich auch viele der in den Sätzen 2.45 sowie 2.46 formulierten Eigenschaften auf das Linearitätsprofil:

**Korollar 2.49.** Sei  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q, u \in \mathbb{F}_2^n, v \in \mathbb{F}_2^q$ . Dann gilt

(a)  $0 \leq \lambda_f(u, v) \leq 1$ .

(b)  $\lambda_f(0, 0) = 1$ .

(c) Alle Spaltensummen des Linearitätsprofils ergeben 1:  $\sum_{u \in \mathbb{F}_2^n} \lambda_f(u, v) = 1$ . Damit gibt es insbesondere zu jedem  $v$  ein  $u$ , so dass  $\lambda_f(u, v) \geq \frac{1}{2^n}$ .

(d)  $f$  ist genau dann affin, wenn es in jeder Spalte des Linearitätsprofils genau einen von Null verschiedenen Wert gibt (nämlich 1). □

Mit diesem Rüstzeug versehen kann nun das Linearitätsmaß der linearen Kryptoanalyse wie folgt definiert werden:

**Definition 2.50** (Lineares Potenzial). Für eine Boole'sche Abbildung  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q$  heißt

$$\Lambda_f := \max_{\substack{u \in \mathbb{F}_2^n, v \in \mathbb{F}_2^q \\ (u,v) \neq (0,0)}} \lambda_f(u, v) \quad (2.32)$$

lineares Potenzial von  $f$ . □

Klar ist  $\Lambda_f = 1$  für affine Abbildungen  $f$ . Je kleiner (d.h. näher bei 0) das lineare Potenzial einer Funktion ist, desto „ferner“ ist sie der Linearität und desto ineffektiver werden die für einen linearen Angriff notwendigen linearen Approximationen für  $f$ . Eine als S-Box eingesetzte Funktion sollte also möglichst kleines  $\Lambda_f$  besitzen.

#### 2.4.4 Differenzentabelle und differentielles Potenzial

Was die lineare Approximationstabelle für die lineare Kryptoanalyse, ist die Differenzentabelle für die differentielle Kryptoanalyse. Wie aus Kapitel 2.3.2 bekannt, ist sie für eine Boole'sche Abbildung  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q$  definiert als

$$\#D_f(u, v) = \#\{x \in \mathbb{F}_2^n \mid f(x + u) = f(x) + v\}.$$

$\#D_f(u, v)$  misst also die Zahl der Eingabewerte von  $f$ , für die die Eingabedifferenz  $u$  die Ausgabedifferenz  $v$  liefert. Je höher diese Anzahl ist, desto besser ist die differentielle Charakteristik  $(u, v)$  für einen differentiellen Angriff nutzbar.

Die Differenzentabelle weist viele dem Walsh-Spektrum vergleichbare Eigenschaften auf, die im Folgenden zusammengefasst werden sollen. Insbesondere erweisen sich (wie bei der linearen Kryptoanalyse) affine Abbildungen als besonders anfällig gegenüber differentiellen Angriffen:

**Satz 2.51.** *Sei  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q$  sowie  $u \in \mathbb{F}_2^n, v \in \mathbb{F}_2^q$ . Dann gilt:*

- (a) *Alle Einträge der Differenzentabelle sind gerade.*
- (b) *Es gilt  $0 \leq \#D_f(u, v) \leq 2^n$  für beliebiges  $u, v$ .*
- (c)  *$\#D_f(0, 0) = 2^n$ .*
- (d) *Jede Zeilensumme der Differenzentabelle beträgt  $2^n$ .*
- (e) *Ist  $f$  affin, dann gibt es in jeder Zeile der Differenzentabelle genau einen von Null verschiedenen Eintrag (nämlich  $2^n$ ).*

*Beweis.* Zu (a): Ist  $u \neq 0$  und  $x_1 \in D_f(u, v)$ , so ist auch  $x_2 := x_1 + u$  mit  $x_2 \neq x_1$  in  $D_f(u, v)$ , denn  $f(x_2 + u) = f((x_1 + u) + u) = f(x_1) = f(x_1 + u) + v = f(x_2) + v$  nach Voraussetzung. Für  $u = 0$  folgt die Behauptung aus (c) und (d).

(b) ist trivial, (c) folgt aus  $D_f(0, 0) = \{x \in \mathbb{F}_2^n \mid f(x + 0) = f(x) + 0\} = \mathbb{F}_2^n$ .

Zu (d): Für festes  $u, x \in \mathbb{F}_2^n$  gibt es genau ein  $v \in \mathbb{F}_2^q$  mit  $f(x + u) - f(x) = v$ . Also gilt  $\sum_{v \in \mathbb{F}_2^q} \#D_f(u, v) = \sum_{v \in \mathbb{F}_2^q} \#\{x \in \mathbb{F}_2^n \mid f(x + u) - f(x) = v\} = 2^n$ .

Zu (e): Ist  $f(x) = Ax + b$ , so gilt

$$\begin{aligned} D_f(u, v) &= \{x \in \mathbb{F}_2^n \mid A(x + u) + b = Ax + b + v\} \\ &= \{x \in \mathbb{F}_2^n \mid Au = v\} \\ &= \begin{cases} \mathbb{F}_2^n & \text{falls } Au = v, \\ \emptyset & \text{sonst.} \end{cases} \end{aligned}$$

Daraus folgt die Behauptung.  $\square$

Analog zum Linearitätsprofil kann man durch Normierung mit dem Faktor  $\frac{1}{2^n}$  das so genannte Differenzenprofil erhalten:

**Definition 2.52** (Differenzenprofil). Für eine Boole'sche Abbildung  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q$  sowie  $u \in \mathbb{F}_2^n, v \in \mathbb{F}_2^q$  heißt die Funktion

$$\begin{aligned} \delta_f : \mathbb{F}_2^n \times \mathbb{F}_2^q &\rightarrow \mathbb{Q} \\ \delta_f(u, v) &:= \frac{1}{2^n} \# D_f(u, v) \end{aligned}$$

Differenzenprofil von  $f$ .  $\square$

Die meisten der in Satz 2.51 aufgeführten Eigenschaften übertragen sich unmittelbar auf das Differenzenprofil:

**Korollar 2.53.** Sei  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q$  sowie  $u \in \mathbb{F}_2^n, v \in \mathbb{F}_2^q$ . Dann gilt:

- (a) Es gilt  $0 \leq \delta_f(u, v) \leq 1$  für beliebiges  $u, v$ .
- (b)  $\delta_f(0, 0) = 1$ .
- (c) Alle Zeilensummen des Differenzenprofils ergeben 1:  $\sum_{v \in \mathbb{F}_2^q} \delta_f(u, v) = 1$ . Damit gibt es insbesondere zu jedem  $u$  ein  $v$ , so dass  $\delta_f(u, v) \geq \frac{1}{2^q}$ .
- (d) Ist  $f$  affin, so gibt es in jeder Zeile des Differenzenprofils genau einen von Null verschiedenen Eintrag (nämlich 1).  $\square$

Das Linearitätsmaß der differentiellen Kryptoanalyse ergibt sich nun als nichttriviales Maximum des Differenzenprofils:

**Definition 2.54** (Differentielles Potenzial). Für eine Boole'sche Abbildung  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q$  heißt

$$\Omega_f := \max_{\substack{u \in \mathbb{F}_2^n, v \in \mathbb{F}_2^q \\ (u, v) \neq (0, 0)}} \delta_f(u, v) \quad (2.33)$$

differentielles Potenzial von  $f$ .  $\square$

Wie beim linearen Potenzial gilt  $\Omega_f = 1$  für affines  $f$ , und je kleiner das differentielle Potenzial einer Funktion ist, desto ineffektiver ist die beste differentielle Charakteristik, die man für sie formulieren kann. Neben möglichst kleinem linearem Potenzial sollte eine S-Box also auch möglichst kleines differentielles Potenzial aufweisen.

Angesichts der vielen Analogien ist es wenig überraschend, dass Differenzenprofil und Linearitätsprofil eng zusammenhängen:

**Satz 2.55** ([6]). *Für eine Boole'sche Abbildung  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q$  entspricht die Differenzentabelle der Faltung der charakteristischen Funktion von  $f$  mit sich selbst:*

$$\#D_f = \vartheta_f * \vartheta_f. \quad (2.34)$$

*Beweis.* Wir haben nach Definition der Faltung sowie  $\vartheta_f$ :

$$\begin{aligned} \vartheta_f * \vartheta_f(u, v) &= \sum_{x \in \mathbb{F}_2^n} \sum_{y \in \mathbb{F}_2^q} \vartheta_f(x, y) \cdot \vartheta_f(u + x, v + y) \\ &= \sum_{x \in \mathbb{F}_2^n} \vartheta_f(x, f(x)) \cdot \vartheta_f(u + x, v + f(x)) \\ &= \sum_{x \in \mathbb{F}_2^n} \vartheta_f(x + u, f(x) + v) \\ &= \#\{x \in \mathbb{F}_2^n \mid f(x + u) = f(x) + v\} \\ &= \#D_f(u, v), \end{aligned}$$

was zu zeigen war. □

**Korollar 2.56.** *Für eine Boole'sche Abbildung  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q$  ist das Linearitätsprofil die unnormierte Walsh-Transformierte des Differenzenprofils:*

$$\lambda_f = \frac{1}{2^n} \widehat{\delta}_f. \quad (2.35)$$

*Beweis.* Aus dem vorigen Satz folgt mit dem Faltungssatz (Satz 2.39) und Korollar 2.48

$$\begin{aligned} \widehat{\#D_f} &= \widehat{\vartheta_f * \vartheta_f} \\ &= \widehat{\vartheta_f} \cdot \widehat{\vartheta_f} \\ &= 2^{2n} \lambda_f, \end{aligned}$$

und da  $\mathcal{W}$  nach Satz 2.35 linear ist, haben wir

$$\begin{aligned} \lambda_f &= \frac{1}{2^{2n}} \widehat{\#D_f} \\ &= \frac{1}{2^{2n}} 2^{2n} \widehat{\delta}_f \\ &= \frac{1}{2^n} \widehat{\delta}_f, \end{aligned}$$

wie behauptet. □

Das Linearitätsprofil kann also durch Walsh-Transformation und Skalierung aus dem Differenzenprofil oder der Differenzentabelle errechnet werden.

# Kapitel 3

## T-Funktionen

T-Funktionen bilden den hauptsächlichen Untersuchungsgegenstand dieser Arbeit. In diesem Kapitel sollen sie formal definiert und in ihren wesentlichen Eigenschaften dargestellt werden. Dabei liegt ein besonderes Augenmerk auf der so genannten *Bit-Slice-Technik* zur Analyse von T-Funktionen, welche insbesondere die Untersuchung auf Invertierbarkeit oder *Maximum Distance Separability* sowie die systematische Konstruktion von T-Funktionen mit diesen Eigenschaften ermöglicht.

### 3.1 Motivation

Wie wir im vorigen Kapitel gesehen haben, werden Blockchiffren üblicherweise als iterierte Chiffren realisiert, bei denen eine vom Schlüssel abhängige Rundenfunktion iteriert auf das Ergebnis der letzten Runde angewandt wird; die Eingabe der ersten Runde ist dabei der Klartext, das Ergebnis der letzten Runde der Chiffretext. Diesen Ansatz haben sie mit vielen anderen kryptografischen Verfahren gemein, insbesondere Hashfunktionen werden häufig in ähnlicher Weise konstruiert, und auch bei Stromchiffren besteht der Schlüsselstrom häufig aus den aus einem schlüsselabhängigen Startwert iterierten Funktionswerten.

Anhand der in den Abschnitten 2.3.2 und 2.3.3 beschriebenen „generischen“ Angriffe ist offensichtlich, dass beim Entwurf einer guten Rundenfunktion für eine Blockchiffre auf möglichst hohe Nichtlinearität geachtet werden muss, da Chiffren mit linearer Rundenfunktion trivial und solche mit im Sinne der in den Abschnitten 2.4.3 und 2.4.4 vorgestellten Linearitätsmaße niedriger Nichtlinearität leicht angreifbar sind.

Da hochgradig nichtlineare Funktionen, die etwa kleine Eingabedifferenzen möglichst uniform auf die möglichen Ausgabedifferenzen verteilen, sich zumeist nur über komplexe Berechnungsvorschriften realisieren lassen, wird gleichzeitig beim Entwurf danach gestrebt, diese Funktionen nichtsdestotrotz möglichst effizient zu gestalten. Auch die Iterationsanzahl hängt unmittelbar von der Qualität der Rundenfunktion ab: Eine geringe Rundenanzahl wird häufig über eine aufwändige Rundenfunktion erkaufte, umgekehrt erfordern sehr schnelle Rundenfunktionen in der Regel höhere Rundenanzahlen, um immun gegen differentielle und lineare Kryptoanalyse zu sein [31].

Während sich etwa Permutationen individueller Bits in Hardware sehr leicht durch entsprechende Verbindungen realisieren lassen, sind sie in Software schwerer umzusetzen, da die kleinste adressierbare Einheit von Mikroprozessoren in der Regel aus einem Byte

besteht. Um hohe Effizienz zu erreichen, ist eine Rundenfunktion von Vorteil, welche sich in möglichst wenig Maschineninstruktionen umsetzen lässt und dabei idealerweise auch noch auf der vollen Wortbreite des Mikroprozessors arbeiten kann.

Die gängigen Mikroprozessorinstruktionen fallen in der Regel in zwei Kategorien: Zum einen die arithmetischen Operationen wie Addition oder Multiplikation, zum anderen boole'sche Operationen wie logisches Und oder bitweise Komplementierung.

Dadurch, dass diese verfügbaren Operationen verschiedenen algebraischen Strukturen entstammen, ist außerdem eine Erschwerung für den Kryptoanalytiker zu erhoffen; ein Ansatz, der bereits insbesondere von den Designern der Blockchiffre IDEA gewinnbringend eingesetzt worden ist [23, 24].

Gleichzeitig bringt diese Tatsache aber die Schwierigkeit mit sich, dass man sich bei der Konstruktion und der Untersuchung der Eigenschaften solcher Funktionen – allgemein wie eines bestimmten Exemplars – nur bedingt auf die Zusammenhänge und Resultate für die bekannten algebraischen Strukturen stützen kann. Klimov gibt dafür in [14] folgendes Beispiel an: Die Funktion

$$\begin{aligned} f : \mathbb{F}_2^n &\rightarrow \mathbb{F}_2^n \\ x &\mapsto x \oplus (x^2 \vee C), \end{aligned} \tag{3.1}$$

wobei  $C$  eine ungerade Konstante ist und  $n$  nach der Wortbreite des Mikroprozessors gewählt wird, verwendet nur übliche Maschineninstruktionen (exklusives Oder, Multiplikation, Oder), ließe sich also sehr effizient implementieren. Um jedoch etwa in der Rundenfunktion eines SP-Netzwerks verwendet werden zu können, muss  $f$  zumindest für alle in Frage kommenden Werte von  $n$  und  $C$  invertierbar sein, und aus theoretischer Sicht wäre eine Aussage für *beliebiges*  $n$  und  $C$  wünschenswert.

Klimov demonstriert weiterhin, dass die in  $f$  vorliegende Mischung arithmetischer und logischer Operationen dazu führt, dass alle bekannten Ansätze zur Invertierbarkeitsuntersuchung entweder nicht anwendbar sind, fehlschlagen oder nur für bestimmte Werte von  $C$  eine Aussage liefern. Will man also kryptografische Primitive durch Kombinationen dieser Operationen konstruieren, so ist die Entwicklung einer Theorie notwendig, mit deren Hilfe solche Funktionen systematisch und auf einheitliche Art und Weise untersucht werden können.

Als geeigneter Ansatz zur einheitlichen Behandlung hat sich folgende zentrale Beobachtung herausgestellt, die durch Klimov und Shamir in [15] zuerst formuliert worden ist: Die meisten dieser Operationen, und insbesondere alle bislang erwähnten, haben gemein, dass das  $i$ -te Bit der Ausgabe höchstens von den Bits  $0, \dots, i - 1$  der Eingabe abhängt. Mit anderen Worten: Durch den modulo  $2^i$  reduzierten Wert der Eingabe ist der Wert der Ausgabe modulo  $2^i$  ebenfalls bereits festgelegt; der Funktionswert der auf die niederwertigen  $i$  Bits eingeschränkten Eingabe entspricht der auf die niederwertigen  $i$  Bits eingeschränkten „gesamten“ Ausgabe. Für derartige Funktionen haben Klimov und Shamir den Begriff T-Funktion geprägt.

Wie im Folgenden dargestellt wird, erlaubt die Konzentration auf diese Eigenschaft den gewünschten Aufbau einer Systematik, mit deren Hilfe Funktionen wie (3.1) analysiert werden können.

## 3.2 Definition und Notation

Bevor der im vorigen Abschnitt eingeführte Begriff der T-Funktion formal definiert werden kann, sind noch einige notationelle Aspekte zu vereinbaren (in der Notationsweise folgen wir im Allgemeinen [14]).

Im allgemeinen Fall wollen wir vektorwertige Boole'sche Funktionen in mehreren Veränderlichen betrachten, die also einen oder mehrere  $n$ -Bit-Worte auf eines oder mehrere  $n$ -Bit-Worte abbilden. Zur Vermeidung von Mehrdeutigkeiten bezeichnen wir (vgl. Notation in Abschnitt 2.1) im Folgenden das  $i$ -te Bit des Bitvektors  $x = ([x]_{n-1}, \dots, [x]_0)$  mit  $[x]_{i-1}$  und die  $k$ -te Eingabevariable unserer Funktionen mit  $x_{k-1}$ .

$m$   $n$ -Bit-Worte können entweder als eine  $m \times n$ -Matrix mit Einträgen aus  $\mathbb{F}_2$ , als  $m$ -elementiger Spaltenvektor von  $n$ -Bit-Worten oder als  $n$ -elementiger Zeilenvektor von  $m$ -Bit-Spalten dargestellt werden. Bezeichnet  $x$  die gesamte Matrix, so schreiben wir  $x_{k-1}$  für die  $k$ -te Zeile und in Analogie zur Komponentenbildung beim einzeiligen Bitvektor  $[x]_{i-1}$  für die  $i$ -te Spalte:

$$\begin{aligned}
 x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{m-1} \end{pmatrix} &= \begin{pmatrix} [x_0]_{n-1} & \cdots & [x_0]_1 & [x_0]_0 \\ [x_1]_{n-1} & \cdots & [x_1]_1 & [x_1]_0 \\ \vdots & \ddots & \vdots & \vdots \\ [x_{m-1}]_{n-1} & \cdots & [x_{m-1}]_1 & [x_{m-1}]_0 \end{pmatrix} \\
 &= \begin{pmatrix} [x]_{0,n-1} & \cdots & [x]_{0,1} & [x]_{0,0} \\ [x]_{1,n-1} & \cdots & [x]_{1,1} & [x]_{1,0} \\ \vdots & \ddots & \vdots & \vdots \\ [x]_{m-1,n-1} & \cdots & [x]_{m-1,1} & [x]_{m-1,0} \end{pmatrix} = ([x]_{n-1}, \dots, [x]_1, [x]_0)
 \end{aligned} \tag{3.2}$$

Diese Schreibweisen werden zusätzlich in Abbildung 3.1 auf Seite 59 illustriert.

Eine Funktion  $f$ , die  $k$   $n$ -Bit-Worte auf  $l$   $n$ -Bit-Worte abbildet, operiert in dieser Notation also auf einer  $k \times n$ -Matrix von Bits und liefert das Funktionsergebnis als  $l \times n$ -Bitmatrix. Äquivalent dazu (und wesentlich leichter lesbar) kann  $f$  dargestellt werden als Funktion von  $k$  auf  $l$  Zeilen à  $n$  Bits:

$$\begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{k-1} \end{pmatrix} \mapsto \begin{pmatrix} f(x)_0 \\ f(x)_1 \\ \vdots \\ f(x)_{l-1} \end{pmatrix} \tag{3.3}$$

oder als Funktion von  $n$  auf  $n$  Spalten à  $k$  beziehungsweise  $l$  Bits:

$$\begin{pmatrix} [x]_0 \\ [x]_1 \\ \vdots \\ [x]_{n-1} \end{pmatrix}^T \mapsto \begin{pmatrix} [f(x)]_0 \\ [f(x)]_1 \\ \vdots \\ [f(x)]_{n-1} \end{pmatrix}^T. \tag{3.4}$$

**Beispiel 3.1.** Zur Illustration dieser beiden Sichtweisen betrachten wir die Funktion

$$g : \mathbb{F}_2^n \times \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$$

$$\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \mapsto x_0 \wedge (x_1 + 2), \quad (3.5)$$

welche in Matrixschreibweise eine  $2 \times n$ -Matrix von Bits auf einen  $n$ -Bit-Vektor abbildet. Ihre zeilenorientierte Darstellungsform ist bereits in der Definition verwendet worden, spaltenorientiert stellt sich  $g$  folgendermaßen dar:

$$\begin{pmatrix} [x]_0 \\ [x]_1 \\ [x]_2 \\ [x]_3 \\ [x]_4 \\ \vdots \\ [x]_{n-1} \end{pmatrix}^T \mapsto \begin{pmatrix} [x_0]_0 & \wedge & [x_1]_0 \\ [x_0]_1 & \wedge & ([x_1]_1 \oplus 1) \\ [x_0]_2 & \wedge & ([x_1]_2 \oplus [x_1]_1) \\ [x_0]_3 & \wedge & ([x_1]_3 \oplus [x_1]_1 [x_1]_2) \\ [x_0]_4 & \wedge & ([x_1]_4 \oplus [x_1]_1 [x_1]_2 [x_1]_3) \\ & & \vdots \\ [x_0]_{n-1} & \wedge & ([x_1]_{n-1} \oplus [x_1]_1 \cdots [x_1]_{n-2}) \end{pmatrix}^T \quad (3.6)$$

Obwohl (3.5) und (3.6) im Wesentlichen nichts als transponierte Darstellungen der gleichen Funktion sind, werden in der Spaltendarstellung die „bitorientierten“ Aspekte deutlicher ersichtlich, insbesondere die bitweise Und-Verknüpfung von  $[x_0]_i$  und  $[(x_1 + 2)]_i$  sowie die explizit vorhandenen Terme für die Carry-Bits.  $\square$

Wie im vorigen Beispiel zu sehen ist, verdeutlicht die spaltenorientierte Darstellungsform die Abhängigkeiten der Ausgabebits von den einzelnen Eingabebits wesentlich schärfer als die zeilenorientierte, welche dafür den algebraischen Charakter der Funktion besser erkennen lässt.

Mit Hilfe dieser spaltenorientierten Darstellungsweise können wir nun die von Klimov und Shamir betrachtete Abhängigkeitseigenschaft ( $i$ -tes Bit der Ausgabe nur von den ersten  $i$  Bits der Eingabe abhängig) zum definierenden Merkmal einer ganzen Klasse von Funktionen machen:

**Definition 3.2** (T-Funktion). Eine Funktion  $f : \mathbb{F}_2^{k \times n} \rightarrow \mathbb{F}_2^{l \times n}$ , bei der die  $i$ -te Spalte  $[f(x)]_{i-1}$  der Ausgabeworte nur von den ersten  $i$  Spalten  $[x]_{i-1}, \dots, [x]_0$  der Eingabeworte abhängig ist:

$$\begin{pmatrix} [x]_0 \\ [x]_1 \\ [x]_2 \\ \vdots \\ [x]_{n-1} \end{pmatrix}^T \mapsto \begin{pmatrix} f_0([x]_0) \\ f_1([x]_0, [x]_1) \\ f_2([x]_0, [x]_1, [x]_2) \\ \vdots \\ f_{n-1}([x]_0, \dots, [x]_{n-2}, [x]_{n-1}) \end{pmatrix}^T \quad (3.7)$$

wird T-Funktion genannt. Dieser Name rührt daher, dass (3.7) Dreiecksgestalt hat (englisch *triangular*).  $\square$

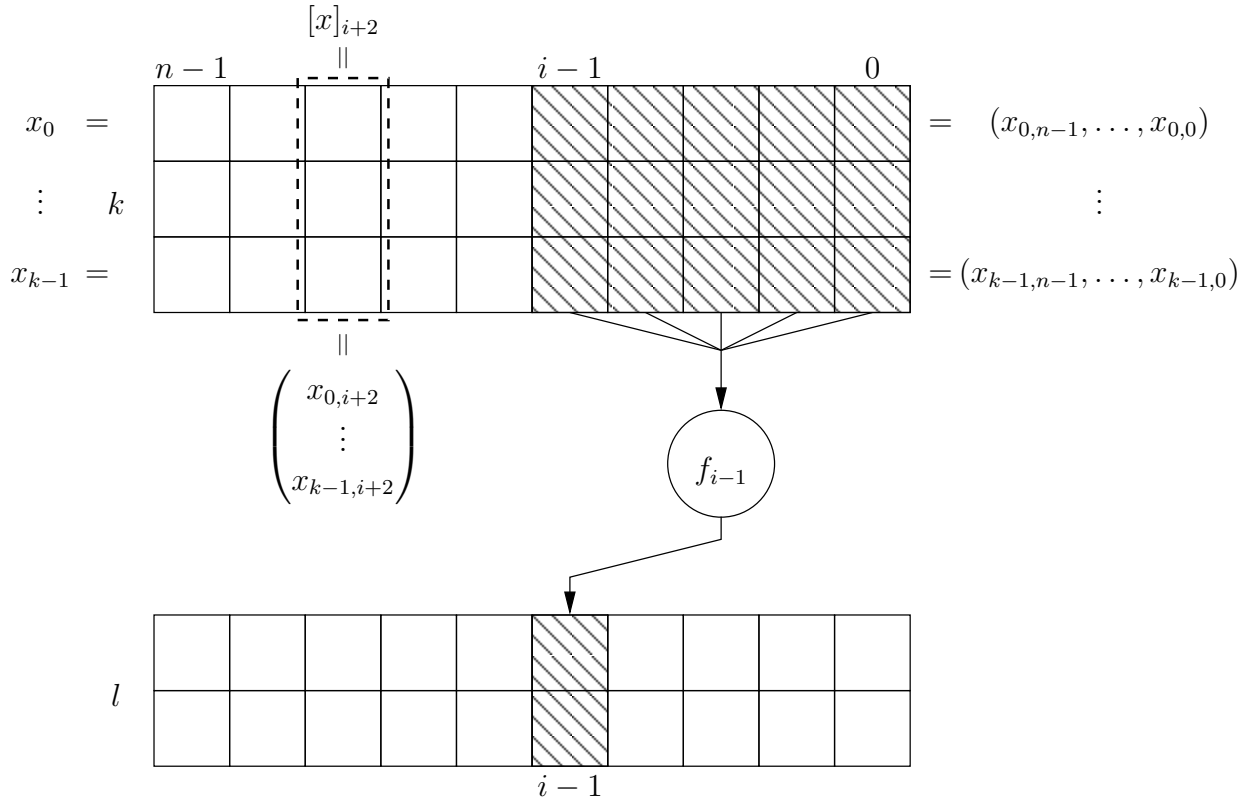


Abbildung 3.1: Notationsweise und Abhängigkeitsverhalten einer T-Funktion.

Dabei beschreibt die Funktion  $f_{i-1}$  genau die  $i$ -te Spalte des Funktionsergebnisses. In einer T-Funktion darf die  $i$ -te der  $n$  „Spaltenfunktionen“  $f_{i-1}$  also höchstens auf den ersten  $i$  Bits *eines jeden* Eingabewortes  $x_0, \dots, x_{k-1}$  operieren:

$$f_{i-1}([x]_{i-1}, \dots, [x]_0) = f_{i-1}\left(\begin{pmatrix} x_{0,i-1} \\ \vdots \\ x_{k-1,i-1} \end{pmatrix}, \dots, \begin{pmatrix} x_{0,0} \\ \vdots \\ x_{k-1,0} \end{pmatrix}\right) = [f(x)]_{i-1}$$

Dieser Abhängigkeitszusammenhang wird in Abbildung 3.1 veranschaulicht.

**Beispiel 3.3.** Betrachten wir unter diesem Aspekt (3.6) erneut, so stellen wir fest, dass jede Spaltenfunktion  $f_i$  lediglich auf den Bits  $x_{\cdot,0}$  bis  $x_{\cdot,i}$  operiert. Damit erfüllt (3.6) das Kriterium von Definition 3.2, also ist (3.5) aus Beispiel 3.1 eine T-Funktion.  $\square$

Unser ursprüngliches Ziel war es, Funktionen zu untersuchen, die sich aus den arithmetischen und logischen Operationen von Mikroprozessoren zusammensetzen. (3.5) ist eine solche Abbildung und wurde im vorigen Beispiel als T-Funktion klassifiziert, was nach unserer bisherigen Methodik durch direktes Anwenden von Definition 3.2 geschehen musste. Zur Beantwortung der Frage, ob auch andere Kompositionen von Prozessorinstruktionen, etwa (3.1) oder  $(x_0 + x_1)//2$ , wobei  $//$  die Ganzzahldivision bezeichnet,

T-Funktionen sind, müssten wir diese Funktionen analog (3.6) in Spaltenform darstellen und die Abhängigkeiten betrachten, was bei komplexeren Ausdrücken aufwändig werden kann. Daher soll nun eine Menge von Prozessorinstruktionen (so genannte **primitive Operationen**) zusammengestellt werden, deren Komposition per Konstruktion die T-Funktionseigenschaft erfüllt:

**Definition 3.4** (Primitive Operation,[14]). Seien  $x$  und  $y$  zwei  $n$ -Bit-Worte. Eine Funktion

$$\phi : \mathbb{F}_2^{k \times n} \rightarrow \mathbb{F}_2^n$$

wird **primitive Operation** genannt, wenn

- $k = 1$  und  $\phi(x)$  eine der folgenden Funktionen ist:

$$\begin{aligned} \phi(x) &= -x \pmod{2^n} && \text{(Negation),} \\ [\phi(x)]_i &= \overline{[x]_i} && \text{(Komplementierung, logisches Nicht)} \end{aligned}$$

oder

- $k = 2$  und  $\phi(x, y)$  eine der folgenden Funktionen ist:

$$\begin{aligned} \phi(x, y) &= x + y \pmod{2^n} && \text{(Addition),} \\ \phi(x, y) &= x - y \pmod{2^n} && \text{(Subtraktion),} \\ \phi(x, y) &= x \times y \pmod{2^n} && \text{(Multiplikation),} \\ [\phi(x, y)]_i &= [x]_i \oplus [y]_i && \text{(Exklusives Oder, XOR),} \\ [\phi(x, y)]_i &= [x]_i \wedge [y]_i && \text{(Logisches Und),} \\ [\phi(x, y)]_i &= [x]_i \vee [y]_i && \text{(Logisches Oder).} \end{aligned}$$

□

Von den auf gängigen Mikroprozessoren üblichen Operationen fehlen dabei die zirkuläre Links- ( $\circlearrowleft$ ) bzw. Rechtsrotation ( $\circlearrowright$ ), der Rechtsshift ( $\overset{\r}{\rightarrow}$ ) sowie die Ganzzahldivision, die alle keine T-Funktionen sind, da niederwertige von höherwertigen Bits abhängen. So ergibt sich etwa im Falle der oben erwähnten Funktion  $(x_0 + x_1)/2$  für die erste Spaltenfunktion  $f_0(x) = [x_0]_1 \oplus [x_1]_1 \oplus [x_0]_0 [x_1]_0$ , und damit eine nicht zulässige Abhängigkeit von  $[x]_1$ .

Die Linksschiebeoperation ( $\overset{\leftarrow}{\circlearrowleft}$ ) ist jedoch eine T-Funktion und wird von obiger Definition indirekt über eine Multiplikation mit einer Zweierpotenz erfasst.

Nachdem damit die Basisoperationen bekannt sind, können diese dazu verwendet werden, durch Komposition beliebige Ausdrücke zu formen:

**Definition 3.5** (Ausdruck). Sei  $\{x_j \mid 0 \leq j \leq m - 1\}$  eine Menge von Variablen. Dann ist ein **Ausdruck** über dieser Variablenmenge entweder

- eine Variable  $x_j$ ,
- eine Konstante  $C \in \mathbb{N}$  oder

- $\phi(y_0, \dots, y_{k-1})$ , wobei  $\phi$  eine  $k$ -stellige primitive Operation ist und die  $y_i$  wiederum Ausdrücke sind.  $\square$

Um einzusehen, dass bei dieser Konstruktion entstehende Abbildungen stets T-Funktionen sind, halten wir zunächst fest, dass alle in Definition 3.4 angegebenen primitiven Operationen T-Funktionen sind: Für die bitweise operierenden Funktionen  $\neg, \oplus, \wedge$  und  $\vee$  ist das trivial; im Falle der arithmetischen Operationen  $+, -$  und  $\times$  folgt das unmittelbar aus

$$\phi(x, y) \equiv \phi(x \bmod 2^i, y \bmod 2^i) \pmod{2^i}.$$

Damit haben wir

**Lemma 3.6.** *Jede primitive Operation ist eine T-Funktion.*  $\square$

Induktion über den Termaufbau liefert nun dank Lemma 3.6 für den Induktionsschritt sofort

**Satz 3.7.** *Jeder Ausdruck (im Sinne von Definition 3.5) ist eine T-Funktion.*  $\square$

Da unser erstes Beispiel (3.1) ein Ausdruck ist, ist er nach vorigem Satz konstruktionsbedingt eine T-Funktion, und wir müssen diese Eigenschaft nicht mehr explizit über die Aufstellung der Spaltenfunktionen nachweisen.

### 3.2.1 Parameter und Bit-Slice-Analyse

Betrachten wir nochmals unser altes Beispiel  $x_0 \wedge (x_1 + 2)$  und dessen Spaltendarstellung (3.6). In diesem Fall war das explizite Aufstellen der Spaltenfunktionen verhältnismäßig leicht, da nur die Konstante 2 zu  $x_1$  addiert wurde, welche die sehr einfache Binärdarstellung  $0 \dots 010$  besitzt, und somit das Carry-Bit in der ersten Spalte konstant war und danach in Spalte  $i$  nur noch im Falle  $[x_1]_1 = \dots = [x_1]_{i-1} = 1$  auftreten konnte. Wäre an Stelle von 2 ein Summand mit aufwändigerer Binärdarstellung gestanden, wären die Terme schnell unhandlich geworden – und zwar ohne dass sich dadurch etwas an der T-Funktionseigenschaft von  $g$  geändert hätte, da die Carry-Funktion in Spalte  $i$  auf jeden Fall nur von den Bits  $0, \dots, i-1$  abhängen kann und somit den Spielraum für T-Funktionen nicht einmal maximal ausnutzt.

Mit anderen Worten: Bei Termen wie den Carry-Bits aus (3.6), die in der  $i$ -ten Spalte nur von den vorigen Spalten abhängen, ist für die Frage, ob es sich um eine T-Funktion handelt, zumindest ihre exakte Form oft unerheblich. Wenn wir derartige Terme erkennen, von den „relevanten“ (von der  $i$ -ten Spalte abhängigen) Termen trennen und von ihrer konkreten Ausprägung abstrahieren könnten, würde dies die Spaltendarstellung unserer Funktionen wesentlich klarer gestalten. Klimov und Shamir ([15], [14]) haben für diese Terme den Begriff „Parameter“ eingeführt.

**Definition 3.8** (Parametrische Funktion, Parameter). Eine Funktion

$$f(x_1, \dots, x_m; \alpha_1, \dots, \alpha_n),$$

deren Argumente durch ein Semikolon in *Eingaben*  $x_i$  und *Parameter*  $\alpha_j$  eingeteilt werden, heißt **Parametrische Funktion (PF)**.

Ein **Parameter** ist eine parametrische Funktion, welche nicht von ihren Eingaben abhängt.  $\square$

Unter Parametern kann man sich unspezifizierte, aber feste Terme vorstellen (insbesondere bezeichnet ein Parameter innerhalb eines Ausdrucks immer den gleichen Term). Sie werden zumeist dazu genutzt, die Abhängigkeiten einer Funktion von ihren Eingaben und weiteren Parametern zu trennen. Parameter werden generell als griechische Kleinbuchstaben dargestellt.

**Beispiel 3.9.**  $f(x; \alpha) = \alpha^2$  ist ein Beispiel für einen Parameter. Die parametrische Funktion

$$f(x; \alpha_1, \alpha_2) = x - (\alpha_1 \alpha_2 \vee \alpha_1)$$

kann auf diese Weise dargestellt werden als

$$f(x; \beta) = x - \beta,$$

mit dem Parameter  $\beta(x; \alpha_1, \alpha_2) = \alpha_1 \alpha_2 \vee \alpha_1$ .  $\square$

Auf diese Weise können wir nun (3.6) getreu dem Vorhaben, in der  $i$ -ten Spalte von den Abhängigkeiten zu den vorigen Spalten zu abstrahieren, folgendermaßen umformulieren:

$$\begin{aligned} [g(x)]_0 &= g_0([x]_0; ) &&= [x]_0 \wedge [x]_0 \\ [g(x)]_1 &= g_1([x]_1; [x]_0) &&= [x]_1 \wedge ([x]_1 \oplus 1) \\ [g(x)]_2 &= g_2([x]_2; [x]_1, [x]_0) &&= [x]_2 \wedge ([x]_2 \oplus \alpha_2([x]_1)) \\ [g(x)]_3 &= g_3([x]_3; [x]_2, [x]_1, [x]_0) &&= [x]_3 \wedge ([x]_3 \oplus \alpha_3([x]_2, [x]_1)) \end{aligned} \quad (3.8)$$

$\vdots$

$$[g(x)]_{n-1} = g_{n-1}([x]_{n-1}; [x]_{n-2}, \dots, [x]_0) = [x]_{n-1} \wedge ([x]_{n-1} \oplus \alpha_{n-1}([x]_{n-2}, \dots, [x]_0))$$

Dabei stehen die Parameter  $\alpha_i$  für die Carry-Funktionen und hängen nur von  $[x]_{i-1}, \dots, [x]_0$  ab. Im konkreten Fall bringen sie keine wesentliche Vereinfachung mit sich, was, wie bereits erwähnt, bei „unbequemeren“ Summanden als 2 jedoch anders aussehen würde.

Inspiziert durch dieses Vorgehen, die  $i$ -te Spalte des Ergebnisses als Funktion der  $i$ -ten Spalte als Eingabe und der vorherigen Spalten als Parameter zu interpretieren, können wir T-Funktionen alternativ wie folgt definieren:

**Definition 3.10** (T-Funktion). Eine T-Funktion ist eine Abbildung der Form

$$\begin{pmatrix} [x]_0 \\ [x]_1 \\ [x]_2 \\ \vdots \\ [x]_{n-1} \end{pmatrix}^T \mapsto \begin{pmatrix} f_0([x]_0) \\ f_1([x]_1; [x]_0) \\ f_2([x]_2; [x]_1, [x]_0) \\ \vdots \\ f_{n-1}([x]_{n-1}; [x]_{n-2}, \dots, [x]_0) \end{pmatrix}^T,$$

wobei die Funktionen  $f_i$  beliebige parametrische Funktionen sind.  $\square$

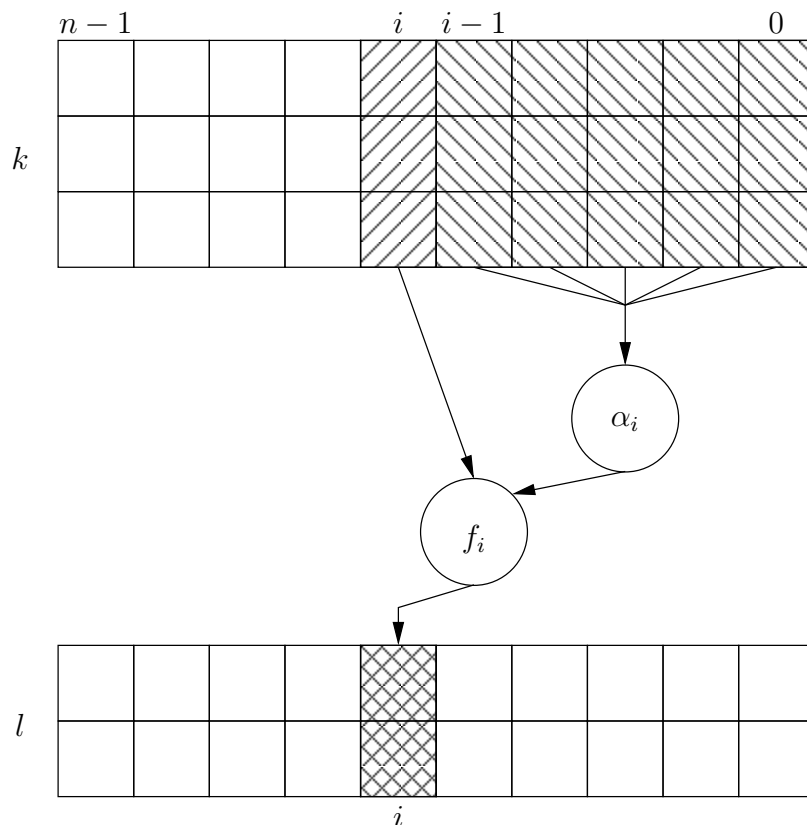


Abbildung 3.2: Parametrisierte Darstellung einer T-Funktion.

Diese gegenüber Definition 3.2 und Abbildung 3.1 leicht veränderte Sichtweise wird in Abbildung 3.2 illustriert.

In den letzten Abschnitten haben wir immer wieder die Spaltendarstellung einer T-Funktion  $f$  gebildet und die einzelnen „Spaltenfunktionen“  $f_i$  betrachtet. In der durch Klimov und Shamir geprägten Terminologie haben diese einen konkreten, einprägsamen Namen, der im Folgenden verwendet werden soll:

**Definition 3.11** (Bit-Slice). Sei  $f$  eine T-Funktion. Eine parametrische Funktion

$$[f(x)]_i = f_i([x]_i; [x]_{i-1}, \dots, [x]_0)$$

heißt die  $i$ -te Bit-Slice (deutsch etwa „Bit-Scheibe“) von  $f$ . □

Wie in [14] demonstriert wird, lassen sich viele Eigenschaften einer T-Funktion (etwa Invertierbarkeit oder Zyklenstruktur) von Eigenschaften ihrer Bit-Slices ableiten. Da in vielen Fällen diese Eigenschaften nicht von der exakten Form der Parameter abhängen, können wir diese wie zum Beispiel in (3.8) in abstrakter Form  $\alpha_i([x]_{i-1}, \dots, [x]_0)$  schreiben. Da darüber hinaus im Falle einer T-Funktion die Abhängigkeitsbeziehungen jedes Parameters implizit gegeben sind, kann die Darstellung von (3.8) weiter vereinfacht

werden zu

$$[g(x)]_i = [x_0 \wedge (x_1 + 2)]_i = [x_0]_i \wedge ([x_1]_i \oplus \alpha_{x_1+2}),$$

wobei  $\alpha_{x_1+2}$  den jeweiligen parametrischen Teil angibt. Diese Vorgehensweise erleichtert die Untersuchung der Bit-Slices enorm.

Unser Ziel ist es nun, von diesem Beispiel zu verallgemeinern und für beliebige Ausdrücke eine solche abstrakte parametrische Darstellungsform bestimmen zu können.

Zunächst stellen wir dazu fest, dass (zumindest) der Fall  $i = 0$  einer gesonderten Behandlung bedarf; in jenem gibt es noch keine vorhergehenden Spalten, die wir während der Auswertung der  $i$ -ten Bit-Slice als Parameter notieren. Sind  $x$  und  $y$  also zwei beliebige Ausdrücke, so gilt stets

$$[x \circ y]_0 = [x]_0 \bullet [y]_0 \quad \text{mit } \circ \in \{+, -, \times, \oplus, \wedge, \vee\},$$

wobei  $\bullet$  den für  $\circ$  geltenden funktionalen Zusammenhang der ersten Bits beschreibt. Bei den bitweise operierenden  $\oplus$ ,  $\wedge$  und  $\vee$  gilt  $\bullet = \circ$ . Nach den in Beispiel 2.5 auf Seite 12 angegebenen Tabellen ist die 0-te Bit-Slice bei Addition und Subtraktion genau dann 1, wenn die Operanden verschieden sind, also  $\bullet = \oplus$ ; und bei der Multiplikation müssen sie dazu simultan 1 sein, also gilt in diesem Fall  $\bullet = \wedge$ .

Für  $i > 0$  ändert sich bei den bitweisen Operationen nichts, die Regel

$$[x \circ y]_i = [x]_i \circ [y]_i \quad \text{mit } \circ \in \{\oplus, \wedge, \vee\}$$

lässt sich direkt aus Definition 3.4 ablesen.

Zusammengestellt ergibt sich die folgende rekursive Charakterisierung der Bit-Slices beliebiger Ausdrücke:

**Satz 3.12** ([14]). *Seien  $x$  und  $y$  zwei Ausdrücke. Dann gilt für  $i > 0$ :*

$$\begin{aligned} [x \pm y]_0 &= [x]_0 \oplus [y]_0, & [x \pm y]_i &= [x]_i \oplus [y]_i \oplus \alpha_{x \pm y}, \\ [x \times y]_0 &= [x]_0 \wedge [y]_0, & [x \times y]_i &= [x]_i \alpha_{[y]_0} \oplus \alpha_{[x]_0} [y]_i \oplus \alpha_{x \times y}, \\ [x \oplus y]_0 &= [x]_0 \oplus [y]_0, & [x \oplus y]_i &= [x]_i \oplus [y]_i, \\ [x \hat{\vee} y]_0 &= [x]_0 \hat{\vee} [y]_0, & [x \hat{\vee} y]_i &= [x]_i \hat{\vee} [y]_i \end{aligned} \quad (3.9)$$

wobei die  $\alpha_*$  Parameter bezeichnen.

*Beweis.* Für die bitweisen Operationen und die 0-ten Bit-Slices der arithmetischen Operationen folgen die angegebenen Beziehungen aus unseren vorigen Überlegungen.

Zum Nachweis von

$$[x + y]_i = [x]_i \oplus [y]_i \oplus \alpha_{x+y}$$

stellen wir  $x$  und  $y$  dar als

$$x = 2^i [x]_i + \dots + 2^0 [x]_0 \quad \text{und} \quad y = 2^i [y]_i + \dots + 2^0 [y]_0.$$

Damit ergibt sich

$$\begin{aligned}
(x + y) \bmod 2^{i+1} &= 2^i [x]_i + \cdots + 2^0 [x]_0 + 2^i [y]_i + \cdots + 2^0 [y]_0 \bmod 2^{i+1} \\
&= 2^i ([x]_i + [y]_i) + 2^{i-1} ([x]_{i-1} + [y]_{i-1}) + \cdots + [x]_0 + [y]_0 \bmod 2^{i+1} \\
&= 2^i (([x]_i + [y]_i + \alpha_{x+y}) \bmod 2) + \alpha \bmod 2^{i+1}
\end{aligned}$$

für einen geeigneten Parameter  $\alpha$ , da wir bei Betrachtung der  $i$ -ten Bit-Slice  $[x]_{i-1}, \dots, [x]_0$  und  $[y]_{i-1}, \dots, [y]_0$  als Parameter ansehen.  $\alpha_{x+y}$  gibt dabei den Wert des Übertragsbits an, welches 1 wird, wenn niederwertige Terme in der Summe zusammen  $2^i$  ergeben. Da  $[x + y]_i$  das höchstwertige Bit von  $x + y \bmod 2^{i+1}$  ist, folgt

$$[x + y]_i = [x]_i \oplus [y]_i \oplus \alpha_{x+y}$$

für einen geeigneten Parameter  $\alpha_{x+y}$ .

Für die Subtraktion kann der Nachweis in analoger Weise geführt werden. Eine Herleitung für die Charakterisierung der Bit-Slices der Multiplikation findet man im Beweis von Theorem 4.2 in [14].  $\square$

Auch für die unären primitiven Operationen lässt sich die allgemeine Form der Bit-Slices leicht angeben: Als bitweise Operation ist die  $i$ -te Bit-Slice der Komplementierung ( $\bar{\phantom{x}}$ ) einfach die Komplementierung derselben, also

$$[\bar{x}]_i = \overline{[x]_i} \quad \text{für alle } i \geq 0; \quad (3.10)$$

und im Falle der Negation folgt wegen  $[-x]_i = [0 - x]_i$  für alle  $i \geq 0$  aus der oben angegebenen Regel für die Subtraktion unmittelbar

$$\begin{aligned}
[-x]_0 &= [0 - x]_0 = [0]_0 \oplus [x]_0 = [x]_0, & \text{sowie} \\
[-x]_i &= [0 - x]_i = [0]_i \oplus [x]_i \oplus \alpha_{-x} = [x]_i \oplus \alpha_{-x}
\end{aligned} \quad (3.11)$$

für alle  $i > 0$ .

**Beispiel 3.13.** Mit Hilfe von Satz 3.12 können wir nun die abstrakte Form der Bit-Slices der T-Funktion  $f(x) = (\bar{x} \vee 3)^2 - x$  durch rekursive Anwendung der allgemeinen Regeln bestimmen:

$$\begin{aligned}
[(\bar{x} \vee 3)^2 - x]_0 &= [(\bar{x} \vee 3)^2]_0 \oplus [x]_0 \\
&= ([(\bar{x} \vee 3)]_0 \wedge [(\bar{x} \vee 3)]_0) \oplus [x]_0 \\
&= [(\bar{x} \vee 3)]_0 \oplus [x]_0 \\
&= ([\bar{x}]_0 \vee [3]_0) \oplus [x]_0 \\
&= (\overline{[x]_0} \vee 1) \oplus [x]_0 \\
&= 1 \oplus [x]_0,
\end{aligned}$$

und für  $i > 0$ :

$$\begin{aligned}
[(\bar{x} \vee 3)^2 - x]_i &= [(\bar{x} \vee 3)^2]_i \oplus [x]_i \oplus \alpha_{(\bar{x} \vee 3)^2 - x} \\
&= \left( \underbrace{[(\bar{x} \vee 3)]_i \alpha_{[(\bar{x} \vee 3)]_0} \oplus \alpha_{[(\bar{x} \vee 3)]_0} [(\bar{x} \vee 3)]_i}_{=0} \oplus \alpha_{(\bar{x} \vee 3)^2} \right) \oplus [x]_i \oplus \alpha_{(\bar{x} \vee 3)^2 - x} \\
&= \alpha_{(\bar{x} \vee 3)^2} \oplus [x]_i \oplus \alpha_{(\bar{x} \vee 3)^2 - x} \\
&= [x]_i \oplus \alpha.
\end{aligned}$$

Alle Bit-Slices von  $f$  haben also die Struktur  $[x]_i \oplus \beta$ , wobei  $\beta$  entweder ein (komplexerer) Parameter oder die Konstante 1 ist.

Klimov bezeichnet diesen rein syntaktischen Vorgang zur Bestimmung der Bit-Slices als „vergleichbar zur symbolischen Differentiation algebraischer Ausdrücke“ ([14], S. 36).  $\square$

### 3.2.2 Breite T-Funktionen

Bislang sind wir davon ausgegangen, die Länge  $n$  unserer Bitvektoren nach der Wortbreite der Mikroprozessoren zu wählen, also etwa  $n = 32$  oder  $n = 64$  für Standardprozessoren beziehungsweise  $n = 8$  oder  $n = 16$  für eingebettete Systeme. Dies liegt vor allem darin begründet, dass die primitiven Operationen nur bis zur nativen Wortbreite des Prozessors zu einer einzelnen Maschineninstruktion korrespondieren und für längere Bitvektoren (größere Zahlen) emuliert werden müssen.

In nicht wenigen kryptografischen Anwendungen sind Funktionen eines einzelnen Maschinenwortes jedoch nicht ausreichend. Zum Beispiel ist zur Konstruktion der Diffusionsebene in einem SP-Netzwerk mit einer Blocklänge von 128 Bit eine invertierbare Funktion erforderlich, die (insgesamt) auf 128-Bit-Vektoren operiert. Selbstverständlich könnte dazu bei  $n = 64$  eine zweidimensionale T-Funktion  $f : \mathbb{F}_2^{2 \times n} \rightarrow \mathbb{F}_2^{2 \times n}$  gewählt werden, wobei die höher- bzw. niederwertigen 64 Bits des internen Zustands  $s$  auf zwei als unabhängig angesehene Variablen  $x, y$  verteilt werden:  $s = 2^n x + y$ . Obwohl nahelegend, bringt es dieser Ansatz mit sich, dass die T-Funktionseigenschaft nur noch in Bezug auf die (streng genommen nicht unabhängigen) Worte  $x$  und  $y$  erfüllt ist:

$$[f(x)]_i = f_i([x]_{i,\dots,0}, [y]_{i,\dots,0}) = f_i([s]_{i+n,\dots,n}; i,\dots,0).$$

Damit ist die Diffusionsebene zwar *mittels* T-Funktionen implementiert worden, als Funktion von  $s$  betrachtet aber *keine* T-Funktion mehr.

Obwohl ähnliche Ansätze in Kapitel 4 an der Tagesordnung sein werden, ist dennoch der alternative Ansatz möglich, trotz der Überschreitung der physischen Wortbreite weiterhin auf einem „logischen“ Wort der Länge  $2n$  zu operieren. Solche T-Funktionen nennen wir *breite* T-Funktionen:

**Definition 3.14** (Breite T-Funktion). Eine T-Funktion  $f : \mathbb{F}_2^{k \times ln} \rightarrow \mathbb{F}_2^{m \times ln}$ , wobei  $l > 1$ , wird **breite T-Funktion** (englisch: *wide T-function*) genannt.  $\square$

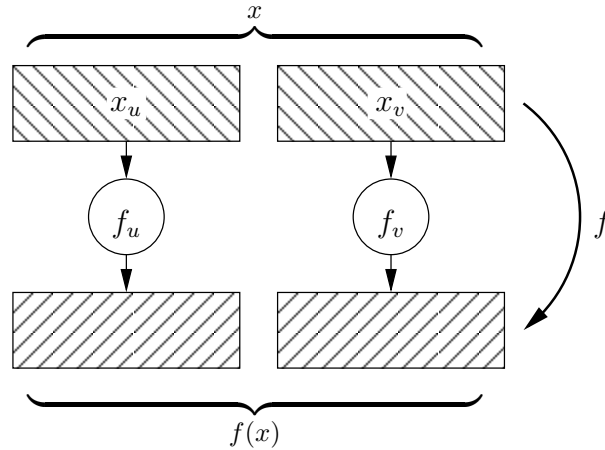


Abbildung 3.3: Aus unabhängigen Teilausdrücken konstruierte breite T-Funktion.

Wie bei gewöhnlichen T-Funktionen gilt dabei, dass die Bits der Spalte  $i$  des Funktionsergebnisses nur von Bits der Spalten  $0, \dots, i$  der Eingabematrix abhängen dürfen.

Eine breite T-Funktion  $f$  stellen wir üblicherweise als Funktion einer  $k \times l$ -Matrix von  $n$ -Bit-Worten auf eine  $m \times l$ -Matrix von  $n$ -Bit-Worten dar; die einzelnen Worte bezeichnen wir mit  $x_{i,b}$ , wobei  $b$  für einen lateinischen Kleinbuchstaben steht:

$$\begin{pmatrix} x_{0,u} & \cdots & x_{0,w} \\ \vdots & \ddots & \vdots \\ x_{k-1,u} & \cdots & x_{k-1,w} \end{pmatrix} \rightarrow \begin{pmatrix} f_{0,u} & \cdots & f_{0,w} \\ \vdots & \ddots & \vdots \\ f_{m-1,u} & \cdots & f_{m-1,w} \end{pmatrix}. \quad (3.12)$$

Dabei nennen wir die  $l$   $n$ -Bit-Worte  $x_{i,u}, \dots, x_{i,w}$  jeder Zeile **physische Variablen** und jede gesamte Zeile eine **logische Variable**.

Zur Beschreibung einer Funktion wie (3.12) benötigen wir allgemein  $m \cdot l$  Ausdrücke über  $n$ -Bit-Worten. Wählen wir die Ausdrücke dabei so, dass in  $f_{i,u}$  lediglich die physischen Variablen der eigenen Spalte  $u$  (also  $x_{0,u}, \dots, x_{k-1,u}$ ) verwendet werden, so ist die gesamte Funktion in jedem Fall wieder eine T-Funktion (siehe Abbildung 3.3).

Allerdings hängt dann  $f_{i,u}$  nicht mehr von den vorhergehenden Spalten ab, was für eine T-Funktion ja zulässig wäre. Folglich können, ohne die T-Funktionseigenschaft zu verletzen, im Ausdruck  $f_{i,u}$  zusätzlich die physischen Variablen  $x_{i,v}, \dots, x_{i,w}$  ( $0 \leq i < k$ ) verwendet werden. Das wird in Abbildung 3.4 auf der nächsten Seite illustriert.

Als Beispiel (leicht vereinfacht aus [17], S. 4) betrachten wir dazu den Fall  $l = 2$ ,  $k = m = 1$  und die Funktion

$$f(x) = (f_u, f_v) = (x_u \oplus x_v, x_v). \quad (3.13)$$

Für ihre Bit-Slices ermitteln wir mit Satz 3.12:

$$\begin{aligned} [f(x)]_0 &= [f_v]_0 = [x_v]_0 &= [x]_0 \\ [f(x)]_{i=1, \dots, n-1} &= [f_v]_i = [x_v]_i &= [x]_i \\ [f(x)]_n &= [f_u]_0 = [x_u \oplus x_v]_0 = [x_u]_0 \oplus [x_v]_0 &= [x]_n \oplus [x]_0 \\ [f(x)]_{i=n+1, \dots, 2n-1} &= [f_u]_{i-n} = [x_u \oplus x_v]_{i-n} = [x_u]_{i-n} \oplus [x_v]_{i-n} &= [x]_i \oplus [x]_{i-n}. \end{aligned}$$

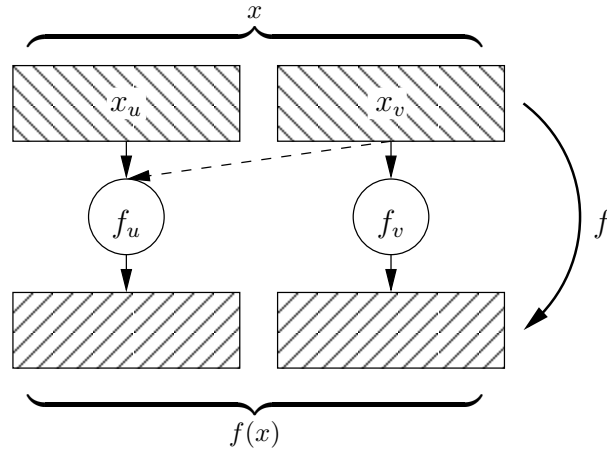


Abbildung 3.4: Aus abhängigen Teilausdrücken konstruierte breite T-Funktion.

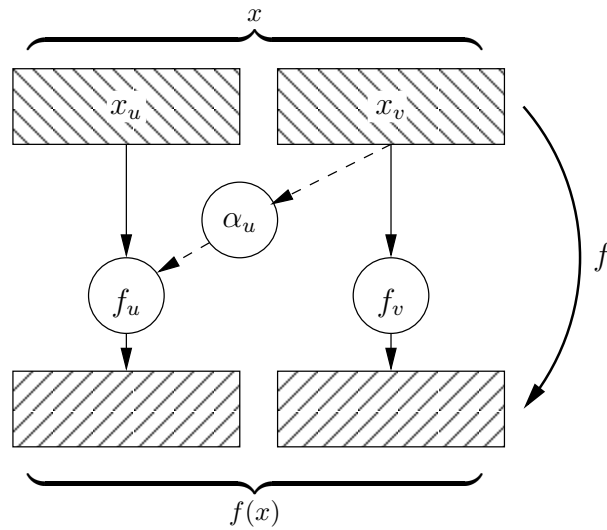


Abbildung 3.5: Nichtprimitive Operationen in breiten T-Funktionen.

Man sieht sofort, dass auf Ebene der logischen Variablen  $[f(x)]_i$  nur von  $[x]_i, \dots, [x]_0$  abhängt, so dass (3.13) tatsächlich eine T-Funktion ist.

Auf diese Weise können in  $f_{i,u}$  sogar beliebige Funktionen der physischen Variablen  $x_{,v}, \dots, x_{,w}$  als Parameter verwendet werden:

$$f_{i,u}(x_{,u}; \alpha_{i,u}(x_{,v}, \dots, x_{,w})).$$

Auch wenn die Parameter  $\alpha_{i,u}$  selbst keine T-Funktionen sind, ist dabei  $f$  insgesamt stets wieder eine T-Funktion, da lediglich auf vorhergehenden Spalten operiert wird (vgl. Abbildung 3.5).

Zur Illustration modifizieren wir das vorige Beispiel (3.13) wie folgt:

$$f(x) = (f_u(x_u; \alpha(x_v)), f_v(x)) = (x_u \oplus (x_v // 2), x_v). \quad (3.14)$$

Die Ganzzahldivision  $//$  ist keine T-Funktion. Da während der Auswertung von  $f_u$  jedoch  $x_v // 2$  lediglich auf ohnehin als Parameter angesehenen Bits operiert, ist  $f$  definitions-

gemäß eine T-Funktion:

$$\begin{aligned}
[f(x)]_0 &= [f_v]_0 = [x_v]_0 &= [x]_0 \\
[f(x)]_{i=1,\dots,n-1} &= [f_v]_i = [x_v]_i &= [x]_i \\
[f(x)]_n &= [f_u]_0 = [x_u \oplus (x_v//2)]_0 = [x_u]_0 \oplus [x_v]_1 &= [x]_n \oplus [x]_1 \\
[f(x)]_{i=n+1,\dots,2n-1} &= [f_u]_{i-n} = [x_u \oplus (x_v//2)]_{i-n} = [x_u]_{i-n} \oplus [x_v]_{i-n+1} &= [x]_i \oplus [x]_{i-n+1}.
\end{aligned}$$

### 3.3 Invertierbarkeit

Für den Einsatz von T-Funktionen in Blockchiffren ist in vielen Fällen deren Invertierbarkeit unabdingbare Voraussetzung oder doch zumindest sehr wünschenswert. Invertierbare totale Abbildungen zwischen endlichen Mengen sind stets Bijektionen, so dass ihre iterierte Anwendung den Wertebereich nicht einschränkt, was dem Kryptoanalytiker entgegen kommen könnte (siehe dazu etwa die *Non-Surjective Attack* von Rijmen in [31] sowie Rijmen und Preneel [32]).

Unser Invertierbarkeitsbegriff ist dabei der übliche:

**Definition 3.15** (Invertierbare Funktion). Sei  $A$  eine endliche Menge. Eine Funktion  $f : A \rightarrow A$  heißt invertierbar, wenn für alle  $x, y \in A$  gilt

$$f(x) = f(y) \Leftrightarrow x = y.$$

□

#### 3.3.1 Charakterisierung invertierbarer T-Funktionen

Die Einführung von Parametern und der Bit-Slice-Technik in Abschnitt 3.2.1 war nicht zuletzt dadurch motiviert, die Untersuchung von Eigenschaften einer T-Funktion auf die Analyse ihrer Bit-Slices zurückführen zu können. Tatsächlich können die invertierbaren T-Funktionen vollständig charakterisiert werden durch eine bestimmte Eigenschaft ihrer Bit-Slices:

**Definition 3.16** (Parametrisch invertierbare Funktion). Sei  $f(x; \alpha)$  eine parametrische Funktion. Ist  $f$  für jeden festen Wert ihrer Parameter  $\alpha$  invertierbar:

$$\forall \alpha. \forall x, y. f(x; \alpha) = f(y; \alpha) \Leftrightarrow x = y,$$

so heißt  $f$  parametrisch invertierbare Funktion (PIF). □

Die beiden wichtigsten Beispiele hierzu sind  $f(x; \alpha) = x \overset{+}{\oplus} \alpha$ , was für beliebiges  $\alpha$  eine PIF ist; und  $g(x; \alpha) = \alpha x$ , was nur im Falle von  $\alpha \neq 0$  für alle Parameterwerte invertierbar und somit eine PIF ist.

**Satz 3.17.** *Eine T-Funktion ist dann und nur dann invertierbar, wenn jede ihrer Bit-Slices eine parametrisch invertierbare Funktion ist.*

*Beweis.* Siehe [14], Theorem 5.1. □

Die Frage nach der Invertierbarkeit einer T-Funktion kann also heruntergebrochen werden auf die Untersuchung, ob alle ihre Bit-Slices parametrisch invertierbare Funktionen sind. Die parametrisierte Form der Bit-Slices wiederum kann mit Hilfe von Satz 3.12 hergeleitet werden. Da diese Form darüber hinaus allgemein für beliebiges  $i > 0$  angegeben wird, werden von der Wortgröße  $n$  unabhängige Invertierbarkeitsaussagen möglich. Zum Beispiel kann exakt nach der hier skizzierten Methode leicht gezeigt werden, dass unsere erste T-Funktion (3.1) für alle  $n$  und alle ungeraden Konstanten  $C$  invertierbar ist – ein Resultat, das ohne die von Klimov und Shamir entwickelte T-Funktions-Methodik ungleich schwerer zu erhalten war.

Einen Spezialfall bilden dabei die Polynome mit ganzzahligen Koeffizienten. Rivest gibt in [34] die folgende erschöpfende Charakterisierung der modulo  $2^n$  invertierbaren Polynome:

**Satz 3.18.** *Sei*

$$P(x) = a_0 + a_1x + \cdots + a_dx^d$$

*ein Polynom mit ganzzahligen Koeffizienten. Dann ist  $P$  modulo  $2^n$ ,  $n \geq 2$ , genau dann invertierbar, wenn  $a_1$  ungerade,  $(a_2 + a_4 + \cdots)$  gerade und  $(a_3 + a_5 + \cdots)$  gerade ist. □*

Als Komposition primitiver Operationen (Addition und Multiplikation) sind Polynome über  $n$ -Bit-Worten selbstverständlich T-Funktionen und ihre Invertierbarkeit kann damit über den Nachweis, dass ihre Bit-Slices PIFs sind, gezeigt werden. Da nach Satz 3.12 die Bit-Slices von  $x + y$  und  $x \oplus y$  bis auf einen additiven Parameter übereinstimmen, lässt sich damit die Aussage des obigen Satzes verallgemeinern zu

**Satz 3.19** ([14], Theorem 5.3). *Sei*

$$P(x) = a_0 \oplus a_1x \oplus \cdots \oplus a_dx^d$$

*ein verallgemeinertes Polynom mit ganzzahligen Koeffizienten (die  $\oplus$ -Operationen können dabei in beliebiger aber fester Weise geklammert sein). Dann ist  $P$  modulo  $2^n$ ,  $n \geq 2$ , genau dann invertierbar, wenn  $a_1$  ungerade,  $(a_2 + a_4 + \cdots)$  gerade und  $(a_3 + a_5 + \cdots)$  gerade ist. □*

### 3.3.2 Testen auf Invertierbarkeit

Wir wollen die im vorigen Abschnitt angegebenen Charakterisierungen invertierbarer T-Funktionen an einigen Beispielen anwenden.

(3.5) von Seite 58 kann als Abbildung von  $\mathbb{F}_2^{2 \times n}$  nach  $\mathbb{F}_2^n$  grundsätzlich nicht invertierbar sein. In der Tat ist bereits die erste Bit-Slice

$$[g(x)]_0 = [x_0]_0 \wedge [x_1]_0$$

keine (parametrisch) invertierbare Funktion, wie die Eingabepaare  $(x_0, x_1) = (1, 0)$  und  $(x_0, x_1) = (0, 1)$  zeigen:  $[g((1, 0)^T)]_0 = 0 = [g((0, 1)^T)]_0$ , im Widerspruch zu Definition 3.15.

Die Bit-Slices der Funktion  $f(x) = (\bar{x} \vee 3)^2 - x$  sind bereits in Beispiel 3.13 bestimmt worden als

$$[(\bar{x} \vee 3)^2 - x]_0 = [x]_0 \oplus 1,$$

und für  $i > 0$ :

$$[(\bar{x} \vee 3)^2 - x]_i = [x]_i \oplus \alpha.$$

Da jede Bit-Slice invertierbar ist, ist  $f$  nach Satz 3.17 für beliebiges  $n$  eine invertierbare T-Funktion.

Um die Invertierbarkeit des Polynoms  $p(x) = 2x + 5$  zu untersuchen, wenden wir (bequemerweise) Satz 3.18 an: Da  $a_1 = 2$  gerade ist, kann  $p$  nicht invertierbar sein. Tatsächlich liefern  $x' = 0$  und  $x'' = 2^{n-1}$  für jedes  $n \geq 2$  ein Gegenbeispiel:  $p(x'') \equiv 2x'' + 5 \equiv 2(2^{n-1}) + 5 \equiv 2^n + 5 \equiv 5 \equiv 2 \cdot 0 + 5 \equiv p(x') \pmod{2^n}$ . Anwendung von Satz 3.17:

$$\begin{aligned} [2x + 5]_0 &= [2x]_0 \oplus [5]_0 \\ &= ([2]_0 \wedge [x]_0) \oplus 1 \\ &= 1, \end{aligned}$$

was keine PIF ist, liefert erwartungsgemäß (und bereits für  $i = 0$ ) die gleiche Aussage. Im Gegensatz dazu ist  $q(x) = 2x^2 + x$  für alle  $n$  modulo  $2^n$  invertierbar, denn  $a_1 = 1$  ist ungerade,  $a_2 = 2$  gerade und die Summe  $(a_3 + a_5 + \dots)$  ist leer und damit gerade. Auch die Bit-Slice-Analyse bestätigt dieses Ergebnis:

$$\begin{aligned} [2x^2 + x]_0 &= [2x^2]_0 \oplus [x]_0 \\ &= ([2]_0 \wedge [x^2]_0) \oplus [x]_0 \\ &= [x]_0, \end{aligned}$$

und für  $i > 0$ :

$$\begin{aligned} [2x^2 + x]_i &= [2x^2]_i \oplus [x]_i \oplus \alpha_{2x^2+2} \\ &= ([2]_i \alpha_{[x^2]_0} \oplus \alpha_{[2]_0} [x^2]_i \oplus \alpha_{2x^2}) \oplus [x]_i \oplus \alpha_{2x^2+2} \\ &= [2]_i \alpha_{[x^2]_0} \oplus [x]_i \oplus \alpha_{2x^2} \oplus \alpha_{2x^2+2} \\ &= [x]_i \oplus \alpha. \end{aligned}$$

( $q$  ist das RC6-Polynom.)

Auch im mehrdimensionalen (multivariaten) Fall lässt sich die Invertierbarkeit einer T-Funktion auf diese Weise bestimmen. Betrachten wir dazu die Funktion

$$f : \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \mapsto \begin{pmatrix} (x_1^2 \vee 1) \oplus x_0 \\ x_1 - (5x_0x_1^2 \wedge 2) \end{pmatrix} = \begin{pmatrix} f_{x_0} \\ f_{x_1} \end{pmatrix}. \quad (3.15)$$

Zunächst berechnen wir getrennt die 0-te Bit-Slice von  $f_{x_0}$  und  $f_{x_1}$ :

$$\begin{aligned} [f_{x_0}]_0 &= [(x_1^2 \vee 1)]_0 \oplus [x_0]_0 \\ &= ([x_1^2]_0 \vee [1]_0) \oplus [x_0]_0 \\ &= 1 \oplus [x_0]_0, \end{aligned}$$

und

$$\begin{aligned} [f_{x_1}]_0 &= [x_1]_0 \oplus [5x_0x_1^2 \wedge 2]_0 \\ &= [x_1]_0 \oplus ([5x_0x_1^2]_0 \wedge [2]_0) \\ &= [x_1]_0 \oplus ([5x_0x_1^2]_0 \wedge 0) \\ &= [x_1]_0, \end{aligned}$$

zusammen also

$$[f(x)]_0 = \begin{pmatrix} [x_0]_0 \oplus 1 \\ [x_1]_0 \end{pmatrix}, \quad (3.16)$$

was offensichtlich invertierbar ist. Analog ermitteln wir für die  $i$ -te Bit-Slice ( $i > 0$ ) beider Ausdrücke:

$$\begin{aligned} [f_{x_0}]_i &= [(x_1^2 \vee 1)]_i \oplus [x_0]_i \\ &= ([x_1^2]_i \vee [1]_i) \oplus [x_0]_i \\ &= \left( [x_1]_i \alpha_{[x_1]_0} \oplus \alpha_{[x_1]_0} [x_1]_i \oplus \alpha_{x_1^2} \right) \oplus [x_0]_i \\ &= \alpha_{x_1^2} \oplus [x_0]_i, \end{aligned}$$

beziehungsweise

$$\begin{aligned} [f_{x_1}]_i &= [x_1]_i \oplus [5x_0x_1^2 \wedge 2]_i \oplus \alpha_{x_1 - (5x_0x_1^2 \wedge 2)} \\ &= [x_1]_i \oplus ([5(x_0x_1^2)]_i \wedge [2]_i) \oplus \alpha_{x_1 - (5x_0x_1^2 \wedge 2)} \\ &= [x_1]_i \oplus \left( ([5]_i \alpha_{[x_0x_1^2]_0} \oplus \alpha_{[5]_0} [x_0x_1^2]_i \oplus \alpha_{5x_0x_1^2}) [2]_i \right) \oplus \alpha_{x_1 - (5x_0x_1^2 \wedge 2)} \\ &= [x_1]_i \oplus [2]_i ([x_0x_1^2]_i) \oplus \alpha' \\ &= [x_1]_i \oplus [2]_i \left( [x_0]_i \alpha_{[x_1^2]_0} \oplus \alpha_{[x_0]_0} [x_1^2]_i \oplus \alpha_{x_0x_1^2} \right) \oplus \alpha' \\ &= [x_1]_i \oplus [2]_i \left( [x_0]_i \alpha_{[x_1^2]_0} \oplus \alpha_{[x_0]_0} \left( [x_1]_i \alpha_{[x_1]_0} \oplus \alpha_{[x_1]_0} [x_1]_i \oplus \alpha_{x_1^2} \right) \oplus \alpha_{x_0x_1^2} \right) \oplus \alpha' \\ &= [x_1]_i \oplus [2]_i \left( [x_0]_i \alpha_{[x_1^2]_0} \oplus \alpha_{[x_0]_0} \alpha_{x_1^2} \oplus \alpha_{x_0x_1^2} \right) \oplus \alpha' \\ &= [x_1]_i \oplus \beta [x_0]_i \oplus \gamma. \end{aligned}$$

Insgesamt ergibt sich

$$[f(x)]_i = \begin{pmatrix} [x_0]_i \oplus \alpha \\ [x_1]_i \oplus \beta [x_0]_i \oplus \gamma \end{pmatrix}, \quad (3.17)$$

wobei wir  $\alpha$  und  $\gamma$  als *additive* und  $\beta$  als *multiplikativen* Parameter bezeichnen. Etwas systematischer formuliert erhalten wir das lineare Gleichungssystem

$$\underbrace{\begin{pmatrix} 1 & 0 \\ \beta & 1 \end{pmatrix}}_A \underbrace{\begin{pmatrix} [x_0]_i \\ [x_1]_i \end{pmatrix}}_b = \underbrace{[f(x)]_i}_c \oplus \underbrace{\begin{pmatrix} \alpha \\ \gamma \end{pmatrix}}_d \quad (3.18)$$

bei dem für alle möglichen  $\alpha, \beta, \gamma \in \mathbb{F}_2$  sowohl der Rang der Koeffizientenmatrix  $A$  als auch derjenige der erweiterten Matrix  $(A, b)$  gleich 2 ist. Folglich ist (3.18) stets eindeutig lösbar, so dass (3.17) eine parametrisch invertierbare Funktion ist. Da auch die niederwertigste Bit-Slice (3.16) eine PIF ist, haben wir damit die Invertierbarkeit von (3.15) für beliebiges  $n$  etabliert.

Alle bisher untersuchten Funktionen waren entweder für alle Werte von  $n$  oder aber für keinen invertierbar. Es gibt jedoch auch Funktionen, die für einige  $1 \leq n < n_0$  invertierbar sind, für alle  $n \geq n_0$  jedoch nicht mehr. Ein solcher Fall liegt vor bei

$$f(x) = x + (4x \wedge x), \quad (3.19)$$

denn offensichtlich verschwindet modulo 2 und modulo 4 die Konstante 4, so dass  $f(x)$  für  $n = 1, 2$  der Identität entspricht und somit invertierbar ist. An der allgemeinen Form der  $i$ -ten Bit-Slice:

$$\begin{aligned} [f(x)]_i &= [x]_i \oplus [4x \wedge x]_i \oplus \alpha_{x+(4x \wedge x)} \\ &= [x]_i \oplus ([4x]_i \wedge [x]_i) \oplus \alpha_{x+(4x \wedge x)} \\ &= [x]_i \oplus (([4]_i \alpha_{[x]_0} \oplus \alpha_{[4]_0} [x]_i \oplus \alpha_{4x}) \wedge [x]_i) \oplus \alpha_{x+(4x \wedge x)} \\ &= [x]_i \oplus [4]_i \alpha_{[x]_0} [x]_i \oplus \alpha_{4x} [x]_i \oplus \alpha_{x+(4x \wedge x)} \\ &= \alpha [x]_i \oplus \beta, \end{aligned}$$

die nur bei  $\alpha \neq 0$  invertierbar und damit keine PIF ist, erkennen wir jedoch, dass  $f$  im allgemeinen Fall nicht invertierbar sein kann. In der Tat lässt sich für  $n \geq 3$  das Gegenbeispiel  $x' = 2^n - 1$  und  $x'' = 2^{n-1} - 1$  finden: Es gilt  $f(x') \equiv (2^n - 1) + (4 \cdot (2^n - 1) \wedge (2^n - 1)) \equiv -1 + (-4 \wedge -1) \equiv -5 \pmod{2^n}$  und  $f(x'') \equiv (2^{n-1} - 1) + (4 \cdot (2^{n-1} - 1) \wedge (2^{n-1} - 1)) \equiv 2^{n-1} - 1 + ((2^{n+1} - 4) \wedge (2^{n-1} - 1)) \equiv 2^{n-1} - 1 + 2^{n-1} - 4 \equiv -5 \pmod{2^n}$ , obwohl  $x' \not\equiv x'' \pmod{2^n}$ . Damit ist (3.19) nur für  $n < 3$  modulo  $2^n$  invertierbar.

Zu beachten ist, dass der umgekehrte Fall (für  $n_0$  nicht, für  $n > n_0$  jedoch invertierbar) nicht möglich ist: Ist  $f$  eine T-Funktion, so gilt  $f(x \bmod 2^n) \equiv f(x \bmod 2^i) \pmod{2^i}$  für alle  $n \geq i \geq 1$ . Gibt es also ein  $n_0$ , so dass  $f$  modulo  $2^{n_0}$  nicht invertierbar ist, so ist  $f$  auch für alle  $n > n_0$  nicht invertierbar.

Die an den vorhergehenden Beispielen demonstrierte Methode, die Invertierbarkeit einer T-Funktion durch den Test zu entscheiden, ob alle ihre Bit-Slices parametrisch invertierbare Funktionen sind, führt in sehr vielen Fällen auf parametrisierte lineare Gleichungssysteme über  $GF(2)$ , ähnlich (3.18). Darüber ist auch der Spezialfall einer

T-Funktion in einer Veränderlichen erfasst, in welchem Fall man eine einzelne lineare Gleichung erhält.

Gibt es in der allgemeinen Form der Bit-Slices nur additive Parameter, so enthält die Koeffizientenmatrix des Gleichungssystems keine Parameter, und die Funktion ist genau dann invertierbar, wenn die Koeffizientenmatrix regulär ist; die additiven Parameter spielen für die Invertierbarkeit keine Rolle. Im eindimensionalen Fall bedeutet dies, dass alle Bit-Slices die Form

$$[x]_i \oplus \alpha$$

haben müssen, wobei  $\alpha$  ein beliebiger Parameter ist (also auch konstant 0 oder 1 sein kann).

Treten einer oder mehrere multiplikative Parameter auf, sind die Bit-Slices genau dann eine PIF, wenn die parametrisierte Koeffizientenmatrix für alle *tatsächlich auftretenden* Wertekombinationen der multiplikativen Parameter regulär ist. Selbstverständlich ist dafür die Nichtsingularität der Matrix für alle möglichen  $\{0, 1\}$ -Kombinationen der multiplikativen Parameter hinreichend. Da die Anzahl der hierbei auf Regularität zu testenden Matrizen exponentiell in der Zahl  $p$  der Parameter wächst, kann dieses Verfahren nur bei kleinen Werten von  $p$  durchgeführt werden.

Dabei ist es wichtig, im Auge zu behalten, dass dieses Kriterium zwar hinreichend, aber nicht notwendig ist: Eine PIF liegt auch dann vor, wenn die Koeffizientenmatrix lediglich für solche Wertkombinationen der multiplikativen Parameter singular ist, die auf Grund der konkreten Beschaffenheit der Parameter nie auftreten können. Ein Beispiel dafür ist die folgende T-Funktion in zwei Veränderlichen:

$$f : \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \mapsto \begin{pmatrix} (x_0 \wedge (x_1 \vee 1)^2) + x_1 \\ x_0 - (x_1 \wedge \overline{(x_1 \vee 1)^2}) \end{pmatrix}. \quad (3.20)$$

Die niederwertigste Bit-Slice kann problemlos als invertierbar erkannt werden:

$$[f(x)]_0 = \begin{pmatrix} [x_0]_0 \oplus [x_1]_0 \\ [x_0]_0 \end{pmatrix}.$$

Für die  $i$ -te Bit-Slice ( $i > 0$ ) ergibt sich jedoch

$$[f(x)]_i = \begin{pmatrix} \alpha_{(x_1 \vee 1)^2} [x_0]_i \oplus [x_1]_i \oplus \gamma \\ [x_0]_i \oplus \overline{\alpha_{(x_1 \vee 1)^2}} [x_1]_i \oplus \delta \end{pmatrix} = \begin{pmatrix} \alpha [x_0]_i \oplus [x_1]_i \\ [x_0]_i \oplus \beta [x_1]_i \end{pmatrix} \oplus \begin{pmatrix} \gamma \\ \delta \end{pmatrix}, \quad (3.21)$$

was zunächst keine PIF zu sein scheint, denn für  $\alpha = \beta = 1$  ist die Matrix

$$\begin{pmatrix} \alpha & 1 \\ 1 & \beta \end{pmatrix} \quad (3.22)$$

nicht regulär. Da hier jedoch  $\beta = \bar{\alpha}$  gilt, kann der Fall  $\alpha = \beta = 1$  nie eintreten; und weil in allen anderen Fällen (3.22) invertierbar ist, ist (3.21) eine parametrisch invertierbare Funktion und damit (3.20) eine invertierbare T-Funktion.

Angesichts der in Satz 3.12 formulierten Regeln erkennt man, dass die  $i$ -ten Bit-Slices jeder T-Funktion, welche nur die primitiven Operationen Multiplikation, Addition, Subtraktion und XOR verwendet, linear sind (der Umkehrschluss gilt nicht, wie etwa die Funktion aus Beispiel 3.13 zeigt). Mithin können auch viele hochgradig nichtlineare T-Funktionen ausschließlich lineare Bit-Slices besitzen. Wie die in (3.9) angegebenen Regeln für die Operationen  $\vee$  und  $\wedge$  zeigen, sind aber auch nichtlineare Bit-Slices möglich. Ist dies der Fall, so lässt sich die Invertierbarkeitsuntersuchung nicht auf ein lineares Gleichungssystem modulo 2 zurückführen. Zum Beispiel ergibt sich für die Funktion

$$f : \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} (x_0 \oplus x_1) - (x_0 \wedge x_2) \\ x_0 + (x_2 \oplus (x_1 \wedge x_2)) \\ x_0 \oplus (x_1 \wedge x_2) \end{pmatrix} \quad (3.23)$$

die folgende Form der  $i$ -ten Bit-Slices:

$$[f(x)]_i = \begin{pmatrix} x'_0 \\ x'_1 \\ x'_2 \end{pmatrix} = \begin{pmatrix} [x_0]_i \oplus [x_1]_i \oplus [x_0]_i [x_2]_i \oplus \alpha \\ [x_0]_i \oplus [x_2]_i \oplus [x_1]_i [x_2]_i \oplus \beta \\ [x_0]_i \oplus [x_1]_i [x_2]_i \end{pmatrix}, \quad (3.24)$$

was nicht linear ist. Dennoch ist (3.24) eine PIF, denn für beliebige aber feste Werte von  $\alpha$  und  $\beta$  können die Werte von  $[x_0]_i$ ,  $[x_1]_i$  und  $[x_2]_i$  aus  $x'_0$ ,  $x'_1$ ,  $x'_2$  errechnet werden: Zunächst ergibt sich  $[x_2]_i = x'_2 \oplus x'_1 \oplus \beta$ , und falls  $[x_2]_i = 0$ , so reduziert sich (3.24) auf das lineare Gleichungssystem

$$\begin{pmatrix} x'_0 \\ x'_1 \\ x'_2 \end{pmatrix} = \begin{pmatrix} [x_0]_i \oplus [x_1]_i \oplus \alpha \\ [x_0]_i \oplus \beta \\ [x_0]_i \end{pmatrix},$$

aus dem  $[x_0]_i$  und  $[x_1]_i$  mühelos berechnet werden können. Andernfalls haben wir

$$\begin{pmatrix} x'_0 \\ x'_1 \\ x'_2 \end{pmatrix} = \begin{pmatrix} [x_1]_i \oplus \alpha \\ [x_0]_i \oplus [x_1]_i \oplus 1 \oplus \beta \\ [x_0]_i \oplus [x_1]_i \end{pmatrix},$$

was wiederum trivial nach  $[x_0]_i$  und  $[x_1]_i$  auflösbar ist. In der niederwertigsten Bit-Slice ist die Situation analog. Damit ist (3.23) also invertierbar.

Dennoch ist in solchen Fällen die Entscheidung, ob es sich um eine PIF handelt oder nicht, häufig nur durch Enumeration aller Eingaben und Parameterwerte möglich, da es schwierig ist, das Lösen der sich ergebenden parametrisierten nichtlinearen Gleichungssysteme für den allgemeinen Fall algorithmisch durchzuführen.

Darüber hinaus ergeben sich auch im Fall linearer Bit-Slices algorithmische Probleme bei der Herleitung der Bit-Slices sowie deren Untersuchung auf die PIF-Eigenschaft, die die Entwicklung eines allgemein anwendbaren automatisierten Invertierbarkeitstests für T-Funktionen verhindern. In der Frühphase dieser Diplomarbeit wurden Versuche unternommen, mit Hilfe der folgenden Manipulationen auf symbolische Weise die allgemeine Form der Bit-Slices einer beliebigen T-Funktion algorithmisch zu bestimmen:

- (a) Erkennung und Ausnutzung von Kommutativität ( $+$ ,  $\times$ ,  $\oplus$ ,  $\wedge$ ,  $\vee$ ) und Assoziativität ( $+$ ,  $\times$ ,  $\oplus$ ,  $\wedge$ ,  $\vee$ ) primitiver Operationen, um Terme zusammenzufassen oder als „gleich“ ansehen zu können;
- (b) Vereinfachung bei  $\oplus$ -Ketten: Für beliebige Ausdrücke  $x, y$  wird  $x \oplus y \oplus x$  (und jede Umordnung dieser Terme) zu  $y$  vereinfacht;
- (c) Vereinfachung von Spezialfällen: Für beliebige Ausdrücke  $x, y$  erlauben die Gleichungen  $x \vee x = x$ ,  $x \wedge x = x$ ,  $x \oplus 0 = x$ ,  $x \vee 1 = 1$ ,  $x \wedge 0 = 0$  eine Vereinfachung der Terme, dabei sind auch die logisch negierten Fälle analog zu behandeln;
- (d) Die 0-ten oder  $i$ -ten Bits von Konstanten  $C$  oder lediglich von solchen abhängige Parameter  $\alpha_{[C]_{\{0,i\}}}$  sind zu 0 oder 1 zu evaluieren, weitere Behandlung wie unter (c);
- (e) In den Bit-Slices  $\wedge$  zur Multiplikation in  $\mathbb{F}_2$  werden lassen;
- (f) Sammeln von Termen: Variablen  $[x_k]_i$  zusammenfassen; mehrere multiplikative und additive Parameter zu einem vereinfachen, wenn feststeht, dass sie sich nicht gegenseitig herauskürzen.

Neben der Tatsache, dass die vollständige Anwendung oder auch nur probeweise Untersuchung aller in einer bestimmten Situation möglichen Regelkombinationen sehr aufwändig werden kann, verbleibt selbst bei perfekt bestimmter symbolischer Form der Bit-Slices die Schwierigkeit, die entstehenden parametrisierten Gleichungssysteme allgemeingültig auf Invertierbarkeit zu untersuchen. Die oben angeführten Beispiele, in denen „unmögliche“ Parameterkombinationen algorithmisch auf effiziente Weise als solche erkannt oder nichtlineare Gleichungssysteme gelöst werden müssten, geben einen Begriff von der Komplexität des Problems. Nicht umsonst also gibt Klimov in [14] die Frage nach der automatisierten Untersuchung einer T-Funktion auf Invertierbarkeit als offenes Problem an.

Gleichwohl ist hervorzuheben, dass die in diesem Abschnitt beschriebene Methode zur Invertierbarkeitsuntersuchung im Regelfall problemlos *manuell* durchführbar ist. Dass Invertierbarkeitstests für T-Funktionen im allgemeinen Fall bislang nicht *automatisiert* werden konnten, liegt nicht an der fehlenden Methodik, sondern allein an der Schwierigkeit, diese algorithmisch so zu formulieren, dass der ausführende Computer nicht mehr von Beobachtungen und Vereinfachungsideen eines Menschen abhängig ist.

### 3.3.3 Konstruktion invertierbarer T-Funktionen

Im vorigen Abschnitt haben wir gesehen, dass die angegebenen Charakterisierungen invertierbarer T-Funktionen zwar einen manuellen, aber keine automatisierten Invertierbarkeitstest erlauben. Soll also eine größere Anzahl invertierbarer T-Funktionen erzeugt werden als manuell getestet werden können, müssen diese folglich *per Konstruktion* invertierbar sein. Im Rest dieses Kapitels sollen die von Klimov und Shamir (vgl. [14], [17], [18]) vorgeschlagenen Konstruktionstechniken vorgestellt werden.

### 3.3.3.1 Ansatz

Die zentrale Idee zur Konstruktion invertierbarer T-Funktionen besteht darin, die „Rückrichtung“ des Invertierbarkeitskriteriums aus Satz 3.17 anzuwenden: Eine T-Funktion, deren Bit-Slices für alle  $i \geq 0$  parametrisch invertierbare Funktionen sind, ist invertierbar. Haben wir dann für die Bit-Slices eine bestimmte Form gewählt, welche die PIF-Eigenschaft garantiert, so können wir durch „Rückwärtsanwendung“ von Satz 3.12 Funktionen mit solchen Bit-Slices konstruieren. Diese sind dann konstruktionsbedingt invertierbar.

Dieses Vorgehen lässt sich am besten durch ein einfaches Beispiel illustrieren:

**Beispiel 3.20.** Um eine invertierbare T-Funktion in einer Veränderlichen  $x$  zu konstruieren, müssen wir zunächst ihren Bit-Slices eine (parametrisch) invertierbare Form geben. Dafür kommen im eindimensionalen Fall nur die Varianten  $[x]_i \mapsto [x]_i$  oder  $[x]_i \mapsto [x]_i \oplus \alpha$  (mit einem beliebigen Parameter  $\alpha$ ) als PIFs in Frage. Betrachtet man die rekursiven Bit-Slice-Regeln von Satz 3.12, so kann der erste Fall nur durch Variationen der Identitätsfunktion erreicht werden (etwa  $x$ ,  $x \wedge (x \oplus \bar{x})$  oder  $x((x^2 + x) - x(x + 1) + 1)$ ). Da er überdies ein Spezialfall der zweiten ist, können wir uns auf jenen konzentrieren. Der Parameter  $\alpha$  kann etwa durch XOR, Addition oder Subtraktion mit einer Konstanten entstehen:  $x \oplus C$  und  $x \pm C$  haben jeweils die geforderte Form. An Hand von (3.9) sehen wir außerdem, dass Multiplikation mit einer geraden Konstanten den Term  $[x]_i$  verschwinden lassen würde, während ungerade Konstanten diesen stehen lassen:  $[x \times C_{\text{odd}}]_i = [x]_i \oplus \alpha_{[x]_0} [C_{\text{odd}}]_i \oplus \alpha_{x \times C_{\text{odd}}} = [x]_i \oplus \alpha$ .

Die Multiplikation mit einer geraden Konstanten zeigt, dass nicht nur T-Funktionen der Form  $\phi(x, C)$  die gewünschte Bit-Slice-Form annehmen können: Auch von  $x$  abhängige Terme, die jedoch in der  $i$ -ten Bit-Slice nur von den Bits  $i - 1, \dots, 0$  abhängen, erzeugen Parameter. Somit sind auch  $x \oplus 4x$  oder  $x + 2\bar{x}$  invertierbare T-Funktionen mit Bit-Slices der Form  $[x]_i \oplus \alpha$ .  $\square$

Das obige Beispiel zeigt, dass nach Auswahl einer PIF als gewünschte Form der Bit-Slices wesentlicher Spielraum bei der Ausgestaltung der Parameter existiert, durch dessen Ausnutzung ausgehend von einer einzigen PIF eine Vielzahl unterschiedlicher Funktionen konstruiert werden können. Bevor verschiedene Möglichkeiten zur Konstruktion von PIFs vorgestellt werden, wollen wir Ansätze zur Konstruktion von Parametern betrachten. Diese können dann in allen Varianten gleichermaßen verwendet werden.

### 3.3.3.2 Parameterkonstruktion

Parameter sind gemäß Definition 3.8 parametrische Funktionen, welche nicht von ihren Eingaben abhängen. Wir untersuchen mit Hilfe von Satz 3.12 für alle primitiven Operationen und deren Komposition, unter welchen Voraussetzungen ihre Bit-Slices Parameter ergeben. Dabei sind stets beide Fälle  $i = 0$  (die niederwertigste Bit-Slice oder auch **least significant bit-slice, LSB**) und  $i > 0$  zu berücksichtigen. Wie wir sehen werden, kann der Fall eintreten, dass ein Ausdruck für  $i > 0$  ein Parameter ist, jedoch nicht in der LSB. Solche Konstruktionen erlauben wir dennoch, da (wie weiter unten demonstriert

---

(R1)	LSB( $\mathcal{E} \times \mathcal{E}$ )
(R2)	LSB( $\mathcal{E}$ ) $\times$ LSB( $\mathcal{E}$ )
(R3)	LSB( $(x \pm y) \times (x \oplus y)$ )
(R3a)	LSB( $(x + y) \times (x - y)$ )
(R4)	LSB( $(\mathcal{E} \oplus \mathcal{P}) \pm \mathcal{E}$ )
(R5)	LSB( $(\mathcal{E} \pm \mathcal{P}) \oplus \mathcal{E}$ )
(R6)	$\mathcal{C}$
(R7)	$\mathcal{P}_0$
(R8)	LSB( $\mathcal{P}' \circ \mathcal{P}''$ )

---

Legende:

$\mathcal{C}$	Konstante
$\mathcal{E}$	Ausdruck
$x, y$	Variablen oder ihre Komplemente
$\mathcal{P}$	Parameter (evtl. bis auf LSB)
$\mathcal{P}_0$	Parameter mit 0 in der LSB
$\circ$	Zweistellige primitive Operation
LSB()	Funktion, die ggf. die LSB zum Parameter macht, siehe Tabelle 3.3

*In einer Zeile bezeichnen mehrfach verwendete Symbole den gleichen Wert; in (R2) etwa stehen beide LSB( $\mathcal{E}$ ) für die gleiche LSB-Korrekturtechnik, angewandt am gleichen Ausdruck.*

---

Tabelle 3.1: Konstruktionstechniken für Parameter.

wird) dieser „Defekt“ in der LSB leicht behoben werden kann. Die wichtigsten sich dabei ergebenden Konstruktionsregeln für Parameter werden in Tabelle 3.1 zusammengefasst.

Der denkbar einfachste Parameter ist die Konstante: Gilt  $\alpha(x) = C$ , so ist für alle  $i \geq 0$   $[\alpha(x)]_i = [C]_i$ , was nicht von der Eingabe  $x$  abhängt und somit für beliebiges  $C \in \mathbb{N}$  ein Parameter ist. Dies erklärt die Konstruktionsregel (R6) in Tabelle 3.1.

Seien  $x$  und  $y$  zwei beliebige Ausdrücke. Für die unären Operationen Komplementierung und Negation sind  $\bar{x}$  respektive  $-x$  genau dann ein Parameter, wenn  $x$  bereits ein Parameter ist, wie deren in (3.10) und (3.11) angegebenen Formen sowohl für die 0-te als auch die  $i$ -te Bit-Slice zeigen. Damit können wir also in jeder folgenden Konstruktion das Vorkommen eines Parameters  $\alpha$  beliebig durch  $\bar{\alpha}$  oder  $-\alpha$  ersetzen.

Bei Addition und Subtraktion gilt  $[x \pm y]_i = [x]_i \oplus [y]_i \oplus \alpha$ , was nur dann ein Parameter sein kann, wenn entweder  $x$  und  $y$  beide Parameter sind, oder aber sich die variablen Terme aus  $[x]_i$  und  $[y]_i$  gegenseitig herauskürzen, so dass  $[x]_i \oplus [y]_i$  ein Parameter wird. Dies ist insbesondere bei  $x = y$  der Fall, denn dann verschwindet dieser Term vollständig. Von den möglichen Konstruktionen, in denen eine solche Kürzung stattfindet, ergeben sich zunächst für einen Parameter  $\alpha$  und einen Ausdruck  $x$  mit  $x + \alpha + x$  und  $\alpha - x - x$  nichts als Synonyme für  $\alpha \pm 2x$  (was bei der Multiplikation behandelt wird), sowie  $x + \alpha - x$ , was sich zu  $\alpha$  vereinfacht. Da jedoch die Situation beim exklusiven Oder analog ist:  $[x \oplus y]_i = [x]_i \oplus [y]_i$ , erhalten wir durch Kombination von  $+$ ,  $-$  und  $\oplus$  die

folgenden beiden interessanten Konstruktionen:

$$(x \oplus \alpha) \pm x \quad \text{und} \\ (x \pm \alpha) \oplus x,$$

bei welchen sich für einen beliebig komplexen Ausdruck  $x$  die nichtparametrischen Terme gegenseitig herauskürzen. Sie sind in Tabelle 3.1 als (R4) und (R5) erfasst. Zu beachten ist hier, dass die beiden Regeln tatsächlich vier verschiedene Konstruktionen liefern, denn die Klammerung von  $\pm$  und  $\oplus$  lässt sich nicht vertauschen. Erfreulicherweise findet das Kürzen der nichtparametrischen Terme auch in der niederwertigsten Bit-Slice statt, so dass die später beschriebene Sonderbehandlung nur dann erfolgen muss, wenn  $\alpha$  in der LSB kein Parameter ist.

Für logisches Und und Oder gilt  $[x \diamond y]_i = [x]_i \diamond [y]_i$ , was nur dann ein Parameter ist, wenn entweder  $[x]_i$  und  $[y]_i$  beides Parameter sind oder einer der beiden Terme konstant 0 oder 1 wird:  $[x]_i \vee 1 = 1$  beziehungsweise  $[x]_i \wedge 0 = 0$ .  $\wedge$  und  $\vee$  können also im Gegensatz zu  $+$ ,  $-$  und  $\oplus$  nicht direkt dazu benutzt werden, aus einem beliebigen Ausdruck einen Parameter zu konstruieren. Sie erhalten jedoch (wie jene) die Parametereigenschaft.

Am interessantesten ist die Situation bei der Multiplikation: Wegen  $[x \times y]_i = [x]_i \alpha_{[y]_0} \oplus \alpha_{[x]_0} [y]_i \oplus \alpha_{x \times y}$  kann  $[x \times y]_i$  für  $i > 0$  nur dann ein Parameter sein, wenn entweder  $x$  und  $y$  bereits Parameter sind oder sich

$$[x]_i \alpha_{[y]_0} \oplus \alpha_{[x]_0} [y]_i \tag{3.25}$$

zu einem Parameter vereinfacht. Sind sowohl  $x$  als auch  $y$  keine Parameter, kann dieser Fall am leichtesten durch  $x = y$  herbeigeführt werden, denn dann verschwindet der Term (3.25) vollständig. Das Quadrieren eines beliebigen Ausdrucks liefert also einen Parameter, was in Regel (R1) von Tabelle 3.1 erfasst ist. Problematisch ist allerdings die Situation in der LSB, denn mit  $[x \times y]_0 = [x]_0 [y]_0$  ist klar, dass  $[x \times x]_0 = [x]_0$ , was für einen beliebigen Ausdruck  $x$  kein Parameter sein muss. Um in der LSB dennoch einen Parameter zu erhalten, muss eine der unten besprochenen Korrekturtechniken auf  $x^2$  angewandt werden. Da auch  $\text{LSB}(x)$  einen Ausdruck darstellt, erhalten wir mittels  $(\text{LSB}(x))^2$  ebenfalls einen Parameter, jedoch mit dem Unterschied, dass bei diesem  $[(\text{LSB}(x))^2]_0 = [\text{LSB}(x)]_0$  bereits konstruktionsbedingt ein Parameter ist und damit keiner weiteren Korrektur bedarf, siehe Regel (R2) in Tabelle 3.1.

Beschränken wir uns auf einzelne Variablen  $x_0, x_1$  anstatt vollständiger Ausdrücke, so ergibt sich außer beim Quadrieren auch durch die „binomische“ Multiplikation  $(x_0 + x_1)(x_0 - x_1)$  für  $i > 0$  ein Parameter, denn:

$$\begin{aligned} [(x_0 + x_1)(x_0 - x_1)]_i &= [x_0 + x_1]_i \alpha_{[x_0 - x_1]_0} \oplus \alpha_{[x_0 + x_1]_0} [x_0 - x_1]_i \oplus \alpha_{(x_0 + x_1)(x_0 - x_1)} \\ &= ([x_0]_i \oplus [x_1]_i \oplus \alpha_{x_0 + x_1}) \alpha_{[x_0 - x_1]_0} \oplus \alpha_{[x_0 + x_1]_0} ([x_0]_i \oplus [x_1]_i \oplus \alpha_{x_0 - x_1}) \\ &\quad \oplus \alpha_{(x_0 + x_1)(x_0 - x_1)} \\ &\stackrel{(*)}{=} \alpha_{x_0 + x_1} \alpha_{[x_0 - x_1]_0} \oplus \alpha_{[x_0 + x_1]_0} \alpha_{x_0 - x_1} \oplus \alpha_{(x_0 + x_1)(x_0 - x_1)} \\ &= \alpha, \end{aligned}$$

(P0a)	$\mathcal{C}_0$
(P0b)	$\mathcal{C}_0 \times \mathcal{E}$
(P0c)	$\mathcal{C}_0 \wedge \mathcal{P}$
(P0d)	$\bar{1} \wedge \mathcal{P}$
(P0e)	$\mathcal{P}_0 \times \mathcal{P}$
(P0f)	$\mathcal{P}'_0 \circ \mathcal{P}''_0$

Legende:	
$\mathcal{C}_0$	Gerade Konstante
$\mathcal{E}$	Ausdruck
$\mathcal{P}$	Parameter (evtl. bis auf LSB)
$\mathcal{P}_0$	Parameter mit 0 in der LSB
$\circ$	Zweistellige primitive Operation

Tabelle 3.2: Konstruktionsrezepte für Parameter mit Null in der LSB

wobei in Schritt (\*) verwendet wurde, dass wegen  $[x_0 \pm x_1]_0 = [x_0]_0 \oplus [x_1]_0$  auch  $\alpha_{[x_0-x_1]_0} = \alpha_{[x_0+x_1]_0}$  gelten muss. Fast die gleiche Herleitung kann zum Nachweis verwendet werden, dass die verwandte Formel  $(x_0 \pm x_1) \times (x_0 \oplus x_1)$  ebenfalls einen Parameter liefert. In beiden Fällen bleibt noch die LSB zu korrigieren, damit ergeben sich dann die Regeln (R3) und (R3a) in Tabelle 3.1.

Bleibt noch der Fall zu untersuchen, dass ein Parameter mit einem Ausdruck multipliziert wird. Ist  $x$  ein Parameter, so ist  $[x]_i, \alpha_{[y]_0}$  ebenfalls einer, und damit (3.25) insgesamt einen Parameter ergibt, muss offenbar  $\alpha_{[x]_0} = 0$  sein, da  $[y]_i$  ja nicht als Parameter vorausgesetzt werden kann. Damit erhalten wir folgende Regel: Ist  $\alpha_0$  ein Parameter mit  $[\alpha_0]_0 = 0$ , so ist

$$\alpha_0 \times y$$

ein Parameter für jeden Ausdruck (und damit auch Parameter)  $y$ . Diese Konstruktion ist auch in der LSB ein Parameter, denn  $[\alpha_0 \times y]_0 = [\alpha_0]_0 [y]_0 = 0$  nach Wahl von  $\alpha_0$ .

Parameter mit dieser Eigenschaft können auf verschiedene Weise aus Ausdrücken oder gewöhnlichen Parametern konstruiert werden; die wichtigsten Regeln dazu sind in Tabelle 3.2 zusammengefasst und sollen im Folgenden kurz erläutert werden: Als wichtiger Spezialfall der vorigen Konstruktion ergibt sich, dass  $Cy$  genau dann ein Parameter ist, wenn die Konstante  $C$  gerade ist (vgl. Regeln (P0a) und (P0b) in Tabelle 3.2). Ist  $y$  bereits ein Parameter, so entsteht durch logisches Und mit einer geraden Konstanten ein Parameter mit Null in der LSB (Regeln (P0c) und (P0d)). Gleiches geschieht bei der Multiplikation mit einem Parameter mit Null in der LSB (P0e). Aus der allgemeinen Form der 0-ten Bit-Slices aller zweistelligen primitiven Operationen  $\phi$  folgt schließlich, dass eine Null in der LSB von  $\alpha$  und  $\beta$  auch die LSB von  $\phi(\alpha, \beta)$  zu 0 werden lässt (P0f).

Insbesondere haben wir festgestellt, dass für Parameter  $\alpha, \beta$  die Anwendung einer beliebigen unären oder binären primitiven Operationen wiederum einen Parameter  $\phi(\alpha)$  bzw.  $\phi(\alpha, \beta)$  liefert, siehe Regel (R8) in Tabelle 3.1.

Als letzte Frage bleibt noch zu klären, auf welche Weise der bereits einige Male auf-

(L1)	$\mathcal{E} \times \mathcal{P}_0$
(L2)	$\mathcal{C}_0 \times \mathcal{E}$
(L3)	$\mathcal{E} \vee 1$
(L4)	$\mathcal{E} \vee \overline{\mathcal{P}_0}$
(L5)	$\mathcal{E} \wedge \bar{1}$
(L6)	$\mathcal{E} \wedge \mathcal{P}_0$

Legende:	
$\mathcal{C}_0$	Gerade Konstante
$\mathcal{E}$	Ausdruck
$\mathcal{P}_0$	Parameter mit 0 im LSB

Tabelle 3.3: Korrekturtechniken für das niederwertigste Bit

getretene „Defekt“ eines Parameters in der niederwertigsten Bit-Slice behoben werden kann. In einem solchen Fall haben wir die Situation, dass  $[\alpha(x)]_i$  für  $i > 0$  ein Parameter ist, für  $i = 0$  jedoch nicht. Bei T-Funktionen bedeutet die Eigenschaft, in der LSB ein Parameter zu sein, gerade, von *keiner* Eingabespalte mehr abzuhängen.  $[\alpha(x)]_0$  muss folglich unabhängig vom Wert von  $[x]_0$  konstant Null oder Eins sein. Die wesentlichen Möglichkeiten dazu sind in Tabelle 3.3 zusammengestellt.

Einige Möglichkeiten, die LSB konstant auf Null zu setzen, wurden bereits in Tabelle 3.2 aufgeführt und können direkt als LSB-Korrekturtechnik verwendet werden: So korrespondiert Regel (L1) zu (P0e), (L2) zu (P0b) und (L5) zu (P0d). Das logische Und mit einer geraden Konstanten (Regel (P0c)) wird jedoch nicht in allgemeiner Form als LSB-Korrekturtechnik angewandt: Eine solche sollte möglichst wenig invasiv sein, wohingegen durch (P0c) all diejenigen Bit-Slices zu 0 gelöscht würden, in denen  $[C_0]_i = 0$ . Bei dem in (L5) verwendeten  $C_0 = \bar{1} = \underbrace{1 \cdots 1}_0 0$  wird nur die LSB auf 0 gesetzt, während

alle anderen unverändert bleiben. Analog dazu wird eine LSB von 0 auch durch logisches Und mit einem Parameter mit 0 in der LSB erzwungen (L6).

Alternativ kann die LSB eines Parameters konstant auf 1 gesetzt werden, was entweder über logisches Oder mit der Konstanten 1 (siehe (L3)) oder mit einem logisch negierten  $\mathcal{P}_0$  geschehen kann (L4).

Zur Illustration dieser Konstruktionstechniken ein abschließendes Beispiel:

**Beispiel 3.21.** Um einen Parameter in den zwei Veränderlichen  $x_0, x_1$  zu konstruieren, beginnen wir mit dem Ausdruck  $3x_0 - x_1$ . Quadrieren, also Anwendung von Regel (R1), liefert  $(3x_0 - x_1)^2$ , was bis auf die LSB bereits ein Parameter ist. Zu deren Korrektur verwenden wir Regel (L4), wobei für  $\mathcal{P}_0$  gemäß Regel (P0b) der Ausdruck  $4x_0$  gewählt wird. Insgesamt ergibt sich

$$\alpha(x) = (3x_0 - x_1)^2 \vee \overline{4x_0},$$

was wegen Regel (R8) für  $i > 0$  in der  $i$ -ten Bit-Slice ein Parameter ist. Dass der „Defekt“

in der LSB behoben wurde, zeigt

$$\begin{aligned}
[(3x_0 - x_1)^2 \vee \overline{4x_0}]_0 &= [(3x_0 - x_1)^2]_0 \vee \overline{[4x_0]_0} \\
&= [3x_0 - x_1]_0 \vee \overline{[4x_0]_0} \\
&= [3x_0 - x_1]_0 \vee \overline{[4]_0 [x_0]_0} \\
&= [3x_0 - x_1]_0 \vee 1 \\
&= 1.
\end{aligned}$$

Dieses  $\alpha$  kann jetzt überall dort eingesetzt werden, wo die PIF-Konstruktion einen Parameter zulässt.  $\square$

### 3.3.3.3 Lineare Bit-Slices mit additiven Parametern

Nachdem nun allgemeine Konstruktionstechniken für Parameter beschrieben worden sind, benötigen wir noch Methoden zur Konstruktion von PIFs, die gemäß der in Abschnitt 3.3.3.1 beschriebenen Vorgehensweise zur Konstruktion invertierbarer T-Funktionen die Form der  $i$ -ten Bit-Slices beschreiben sollen.

Angenommen, es soll eine invertierbare T-Funktion in den  $k$  Variablen  $x_0, \dots, x_{k-1}$  erzeugt werden. Beim Invertierbarkeitstest (vgl. Seite 73 ff.) haben wir die Beobachtung gemacht, dass stets eine PIF vorliegt, wenn sich in der allgemeinen Form der Bit-Slices ein eindeutig lösbares lineares Gleichungssystem ergibt. Zur Konstruktion einer PIF, deren Veränderliche gerade die  $i$ -ten Bit-Slices von  $x_0, \dots, x_{k-1}$  sind, können wir im einfachsten Fall also folgenden Ansatz verfolgen: Wir wählen eine Matrix  $A = (a_{i,j}), 0 \leq i, j < k$  mit Einträgen aus  $\mathbb{F}_2$  und betrachten die lineare Abbildung

$$\begin{pmatrix} [x_0]_i \\ [x_1]_i \\ \vdots \\ [x_{k-1}]_i \end{pmatrix} \mapsto \begin{pmatrix} a_{0,0} [x_0]_i \oplus a_{0,1} [x_1]_i \oplus \cdots \oplus a_{0,k-1} [x_{k-1}]_i \\ a_{1,0} [x_0]_i \oplus a_{1,1} [x_1]_i \oplus \cdots \oplus a_{1,k-1} [x_{k-1}]_i \\ \vdots \\ a_{k-1,0} [x_0]_i \oplus a_{k-1,1} [x_1]_i \oplus \cdots \oplus a_{k-1,k-1} [x_{k-1}]_i \end{pmatrix}. \quad (3.26)$$

Ist  $A$  regulär, so ist (3.26) invertierbar und beschreibt damit (mangels Vorhandensein von Parametern) gewiss eine PIF. Jedoch ist aus den in (3.9) angegebenen Bit-Slice-Regeln ersichtlich, dass es für eine gegebene Matrix  $A$  nur eine einzige T-Funktion  $f$  gibt, deren Bit-Slices (3.26) entsprechen, und zwar:

$$f : \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{k-1} \end{pmatrix} \mapsto \begin{pmatrix} a_{0,0}x_0 \oplus a_{0,1}x_1 \oplus \cdots \oplus a_{0,k-1}x_{k-1} \\ a_{1,0}x_0 \oplus a_{1,1}x_1 \oplus \cdots \oplus a_{1,k-1}x_{k-1} \\ \vdots \\ a_{k-1,0}x_0 \oplus a_{k-1,1}x_1 \oplus \cdots \oplus a_{k-1,k-1}x_{k-1} \end{pmatrix},$$

zum Beispiel etwa  $g : \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \mapsto \begin{pmatrix} x_0 \oplus x_1 \\ x_1 \end{pmatrix}$  für  $k = 2$  und  $A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ . Um mehr Variationsmöglichkeiten zu erhalten, müssen wir in (3.26) Parameter einführen. Eine Möglichkeit

dazu ist, sie **additiv** hinzuzufügen, was für beliebige Parameter  $\alpha_0, \dots, \alpha_{k-1}$  (die insbesondere auch konstant Null sein können) zu der folgenden Konstruktion führt:

$$\begin{pmatrix} [x_0]_i \\ [x_1]_i \\ \vdots \\ [x_{k-1}]_i \end{pmatrix} \mapsto \begin{pmatrix} a_{0,0} [x_0]_i \oplus a_{0,1} [x_1]_i \oplus \cdots \oplus a_{0,k-1} [x_{k-1}]_i \\ a_{1,0} [x_0]_i \oplus a_{1,1} [x_1]_i \oplus \cdots \oplus a_{1,k-1} [x_{k-1}]_i \\ \vdots \\ a_{k-1,0} [x_0]_i \oplus a_{k-1,1} [x_1]_i \oplus \cdots \oplus a_{k-1,k-1} [x_{k-1}]_i \end{pmatrix} \oplus \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{k-1} \end{pmatrix}. \quad (3.27)$$

Für reguläres  $A$  gilt definitionsgemäß  $\text{rank}(A) = k$ , und da sich der Vektor  $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \dots, \alpha_{k-1})^T$  als Linearkombination der  $k$  linear unabhängigen Spaltenvektoren von  $A$  darstellen lässt, ist der Rang der erweiterten Koeffizientenmatrix  $(A, \boldsymbol{\alpha})$  für beliebige Werte der  $\alpha_i$  ebenfalls gleich  $k$ , womit die eindeutige Lösbarkeit des linearen Gleichungssystems gesichert ist. (3.27) beschreibt folglich eine PIF, falls  $A$  nicht singular ist.

Die Einführung von additiven Parametern ermöglicht es uns weiterhin, bei der Bestimmung von T-Funktionen mit Bit-Slices der Form (3.27) nicht nur XOR, sondern auch Addition und Subtraktion zur Verknüpfung der Einzelterme zu verwenden: Wegen  $[x \pm y]_i = [x \oplus y]_i \oplus \alpha_{x \pm y}$  entsprechen die Bit-Slices von  $+$  und  $-$  bis auf einen additiven Parameter denjenigen von XOR, und die sich in der  $j$ -ten Zeile dadurch ergebenden zusätzlichen additiven Parameter können in  $\alpha_j$  subsummiert werden, so dass sich die Form der Bit-Slices nicht ändert.

**Beispiel 3.22.** Für  $k = 3$  und die invertierbare Matrix  $A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$  ergibt sich folgende Form für die  $i$ -te Bit-Slice:

$$\begin{pmatrix} [x_0]_i \\ [x_1]_i \\ [x_2]_i \end{pmatrix} \mapsto \begin{pmatrix} [x_1]_i \oplus \alpha_0 \\ [x_0]_i \oplus [x_2]_i \oplus \alpha_1 \\ [x_2]_i \oplus \alpha_2 \end{pmatrix},$$

was konstruktionsbedingt für beliebige Werte der Parameter eine PIF ist. Eine mögliche invertierbare T-Funktion, die sich aus dieser Form ableiten lässt, ist

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} x_1 \oplus ((x_0 - 5)^2 \vee 1) \\ x_0 \oplus x_2 \\ x_2 \oplus (2x_0x_1) \end{pmatrix}$$

oder auch

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} x_1 \oplus ((x_0 - 5)^2 \vee 1) \\ x_0 + x_2 \\ x_2 - (2x_0x_1) \end{pmatrix}.$$

□

### 3.3.3.4 Multiplikative Parameter

Bei der im vorigen Abschnitt vorgestellten Konstruktion treten Parameter in den einzelnen Bit-Slices nur additiv auf. Sollen sie auch multiplikativ, also in der Form  $\alpha x$ ,

verwendet werden, so müssen wir sie in der Koeffizientenmatrix unterbringen. Die Einträge von  $A = (\mathbf{a}_{i,j})$ ,  $0 \leq i, j < k$ , können nun also entweder Elemente von  $\mathbb{F}_2$  oder aber Parameter  $\alpha_{i,j}$  sein:

$$\begin{pmatrix} [x_0]_i \\ [x_1]_i \\ \vdots \\ [x_{k-1}]_i \end{pmatrix} \mapsto \begin{pmatrix} \mathbf{a}_{0,0} [x_0]_i \oplus \mathbf{a}_{0,1} [x_1]_i \oplus \cdots \oplus \mathbf{a}_{0,k-1} [x_{k-1}]_i \\ \mathbf{a}_{1,0} [x_0]_i \oplus \mathbf{a}_{1,1} [x_1]_i \oplus \cdots \oplus \mathbf{a}_{1,k-1} [x_{k-1}]_i \\ \vdots \\ \mathbf{a}_{k-1,0} [x_0]_i \oplus \mathbf{a}_{k-1,1} [x_1]_i \oplus \cdots \oplus \mathbf{a}_{k-1,k-1} [x_{k-1}]_i \end{pmatrix} \oplus \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{k-1} \end{pmatrix}. \quad (3.28)$$

Ist  $A$  nun regulär für alle möglichen  $\{0, 1\}$ -Wertekombinationen der  $\alpha_{i,j}$ , so lässt sich unsere Argumentation aus Abschnitt 3.3.3.3 direkt übertragen, und (3.28) ist für beliebige Werte der additiven Parameter  $\beta_j$  eine PIF. Die auf Seite 74 ff. erläuterte Problematik, dass dieses Kriterium eigentlich zu stark (da nicht notwendig) ist, ignorieren wir hier, da dieses Verfahren trotz der Beschränkung auf für alle möglichen Werte der Parameter invertierbare Matrizen eine Vielzahl verschiedener PIFs für die allgemeine Form der Bit-Slices liefert.

**Beispiel 3.23.** Eine invertierbare T-Funktion in  $k = 3$  Variablen mit multiplikativen und additiven Parametern kann wie folgt konstruiert werden: Als Koeffizientenmatrix wählen wir  $A = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & \alpha_{1,2} \\ 1 & \alpha_{2,1} & \alpha_{2,2} \end{pmatrix}$ , welche als untere Dreiecksmatrix mit Einsen auf der Hauptdiagonale für alle möglichen Werte der Parameter regulär ist. Damit ergibt sich die folgende Form für die Bit-Slice Nummer  $i$ :

$$\begin{pmatrix} [x_0]_i \\ [x_1]_i \\ [x_2]_i \end{pmatrix} \mapsto \begin{pmatrix} [x_2]_i \oplus \beta_0 \\ [x_1]_i \oplus \alpha_{1,2} [x_2]_i \oplus \beta_1 \\ [x_0]_i \oplus \alpha_{2,1} [x_1]_i \oplus \alpha_{2,2} [x_2]_i \oplus \beta_2 \end{pmatrix}, \quad (3.29)$$

und dass es sich dabei um eine PIF handelt, folgt aus der Konstruktion. Nun können wir daraus eine T-Funktion erstellen, deren Bit-Slices diese Form haben. Dabei wird die boole'sche Multiplikation  $\alpha [x]_i$  in der Bit-Slice auf der Ebene der Variablen gemäß (3.9) zum logischen Und ( $\alpha \wedge x$ ), und die sechs Parameter  $\alpha_{i,j}$  und  $\beta_j$  können beliebig gewählt werden:

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} x_2 & - ((x_0 + x_1)(x_0 \oplus x_1) \vee 1) \\ x_1 & \oplus (6x_0^3) \wedge x_2 \\ x_0 + (2x_0x_1) \wedge x_1 & \oplus (x_2 \vee \overline{2x_0})^2 \end{pmatrix}.$$

Dass die Bit-Slices dieser Funktion tatsächlich (3.29) entsprechen, lässt sich mit Hilfe von Satz 3.12 leicht verifizieren. In unserem Beispiel wird ferner durch die Wahl  $\alpha_{2,2} = \beta_1 = 0$  verdeutlicht, dass sowohl multiplikative als auch additive Parameter verschwinden dürfen.  $\square$

### 3.3.3.5 Nichtlineare Bit-Slices

Obwohl durch die bislang vorgestellten Ansätze hochgradig nichtlineare invertierbare T-Funktionen erzeugt werden können, so haben alle dennoch konstruktionsbedingt aus-

schließlich lineare  $i$ -te Bit-Slices. Eine Möglichkeit zur Konstruktion einer (möglicherweise) nichtlinearen PIF in  $k$  Variablen sieht so aus:

- (a) Wähle eine Permutation  $\pi : \mathbb{F}_2^k \rightarrow \mathbb{F}_2^k$ .
- (b) Bezeichne mit  $[\pi]_i$ ,  $0 \leq i < k$ , die  $i$ -te Spalte der Wahrheitstabelle von  $\pi$ . Da die  $[\pi]_i$  Boole'sche Funktionen sind, können sie nach Satz 2.30 in algebraischer Normalform  $\text{ANF}([\pi]_i)$  in den Variablen  $[x_{k-1}]_i, \dots, [x_0]_i$  geschrieben werden. Damit ist

$$\begin{pmatrix} [x_0]_i \\ [x_1]_i \\ \vdots \\ [x_{k-1}]_i \end{pmatrix} \mapsto \begin{pmatrix} \text{ANF}([\pi]_0) \\ \text{ANF}([\pi]_1) \\ \vdots \\ \text{ANF}([\pi]_{k-1}) \end{pmatrix} \quad (3.30)$$

(und jede beliebige Umordnung der Zeilen davon) eine invertierbare Funktion.

- (c) Füge additive Parameter  $\alpha_0, \dots, \alpha_{k-1}$  hinzu:

$$\begin{pmatrix} [x_0]_i \\ [x_1]_i \\ \vdots \\ [x_{k-1}]_i \end{pmatrix} \mapsto \begin{pmatrix} \text{ANF}([\pi]_0) \\ \text{ANF}([\pi]_1) \\ \vdots \\ \text{ANF}([\pi]_{k-1}) \end{pmatrix} \oplus \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{k-1} \end{pmatrix}, \quad (3.31)$$

was nichts an der Invertierbarkeit der Konstruktion ändert.

Dabei ist zu beachten, dass für  $k = 1, 2$  die Bit-Slices aller Permutationen affin sind, so dass in (3.31) keine nichtlinearen Bit-Slices entstehen. Für  $k \geq 3$  können die Bit-Slices einer Permutation jedoch nichtlinear sein, wie das folgende Beispiel demonstriert.

**Beispiel 3.24.** Allgemein muss die Wahrheitstabelle von  $[\pi]_i$  als Bit-Slice einer Permutation von  $\mathbb{F}_2^k$  *balanciert* sein, also jeweils genau  $2^{k-1}$  Nullen und Einsen enthalten. (Dies kann man folgendermaßen einsehen: Die Wahrheitstabelle von  $\pi$  ist eine Zeilenpermutation der Wahrheitstabelle der Identitätsabbildung von  $\mathbb{F}_2^k$ , deren Spalten balanciert sind. Folglich müssen auch die Bit-Slices von  $\pi$  balancierte Boole'sche Funktionen sein.)

$k = 1$ : Die einzigen Möglichkeiten sind  $\pi_1 : 0 \mapsto 0, 1 \mapsto 1$  und  $\pi_2 : 0 \mapsto 1, 1 \mapsto 0$ . Die algebraischen Normalformen von  $\pi_1$  und  $\pi_2$  sind  $[x_0]_i$  beziehungsweise  $[x_0]_i \oplus 1$ , was beides affin ist.

$k = 2$ : In diesem Fall gibt es für jedes der beiden  $[\pi]_i$  gerade  $\binom{4}{2} = 6$  mögliche balancierte Boole'sche Funktionen, deren algebraische Normalformen  $[x_0]_i, [x_0]_i \oplus 1, [x_1]_i, [x_1]_i \oplus 1, [x_0]_i \oplus [x_1]_i$  und  $[x_0]_i \oplus [x_1]_i \oplus 1$  allesamt affin sind.

$k = 3$ : Hier sind nichtlineare Bit-Slices möglich. Wir wählen die Permutation  $\pi = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 5 & 7 & 0 & 2 & 4 & 6 & 1 \end{pmatrix}$ , deren Wahrheitstabelle so aussieht:

$x_2$	$x_1$	$x_0$	$[\pi]_2$	$[\pi]_1$	$[\pi]_0$
0	0	0	0	1	1
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	0
1	0	0	0	1	0
1	0	1	1	0	0
1	1	0	1	1	0
1	1	1	0	0	1

Daraus ergeben sich die folgenden algebraischen Normalformen:

$$\begin{aligned} [\pi]_2 &= x_0 \oplus x_1, \\ [\pi]_1 &= 1 \oplus x_0, \\ [\pi]_0 &= 1 \oplus x_0x_1 \oplus x_2, \end{aligned}$$

von denen die letzte tatsächlich nichtlinear ist. In die Variablen  $[x_0]_i, [x_1]_i, [x_2]_i$  umgeschrieben ist also

$$\begin{pmatrix} [x_0]_i \\ [x_1]_i \\ [x_2]_i \end{pmatrix} \mapsto \begin{pmatrix} [x_0]_i \oplus [x_1]_i \\ 1 \oplus [x_0]_i \\ 1 \oplus [x_0]_i [x_1]_i \oplus [x_2]_i \end{pmatrix} \oplus \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{pmatrix}$$

die gesuchte nichtlineare Bit-Slice-Form. Bei der Erstellung einer Funktion mit diesen Bit-Slices ist zu beachten, dass auf Bit-Ebene  $x \oplus 1$  äquivalent zu  $\bar{x}$  ist. Dank Kommutativität und Assoziativität von XOR können wir also eine Kette  $1 \oplus x \oplus y$  beliebig als  $\bar{x} \oplus y$ ,  $x \oplus \bar{y}$  oder  $\overline{x \oplus y}$  umsetzen. Beispielsweise entspricht

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} x_0 \oplus x_1 & \oplus & 2(x_0 \wedge x_2) \\ \bar{x}_0 & + & 5 \\ ((x_0 \wedge x_1) - \bar{x}_2) & \oplus & (x_1^2 \vee 1) \end{pmatrix}$$

dieser Form und ist damit konstruktionsgemäß eine invertierbare T-Funktion mit nichtlinearen Bit-Slices. □

### 3.3.3.6 Breite T-Funktionen und nichtprimitive Operationen

Die in den letzten Abschnitten beschriebenen Verfahren zur Konstruktion von PIFs sowie Parametern können direkt zur Konstruktion invertierbarer breiter T-Funktionen verwendet werden. Wie gewohnt stellen wir zunächst eine bestimmte parametrisierte Bit-Slice-Form zusammen, aus der dann Funktionen abgeleitet werden, deren Bit-Slices dieser Form entsprechen.

Bei einer breiten T-Funktion in  $k \times l$  physischen Variablen haben wir  $l$  Spalten je logischer Variable. Für jede zu den physischen Variablen  $x_{.,u}$  gehörende Spalte wählen wir eine Matrix  $A^{(u)} = (a_{ij}^{(u)})$ ,  $0 \leq i, j < k$  mit Einträgen aus  $\mathbb{F}_2$ , insgesamt also  $l$  solche Matrizen. Zusätzlich führen wir in jeder Spalte  $k$  additive Parameter ein:

$$\begin{pmatrix} [x_{0,u}]_i & \cdots & [x_{0,w}]_i \\ \vdots & \ddots & \vdots \\ [x_{k-1,u}]_i & \cdots & [x_{k-1,w}]_i \end{pmatrix} \mapsto \left( \underbrace{A^{(u)} \begin{pmatrix} [x_{0,u}]_i \\ \vdots \\ [x_{k-1,u}]_i \end{pmatrix} \oplus \begin{pmatrix} \alpha_{0,u} \\ \vdots \\ \alpha_{k-1,u} \end{pmatrix}}_{\text{Spalte } 0} \cdots \underbrace{A^{(w)} \begin{pmatrix} [x_{0,w}]_i \\ \vdots \\ [x_{k-1,w}]_i \end{pmatrix} \oplus \begin{pmatrix} \alpha_{0,w} \\ \vdots \\ \alpha_{k-1,w} \end{pmatrix}}_{\text{Spalte } l-1} \right),$$

beziehungsweise explizit notiert:

$$\begin{pmatrix} [x_{0,u}]_i & \cdots & [x_{0,w}]_i \\ \vdots & \ddots & \vdots \\ [x_{k-1,u}]_i & \cdots & [x_{k-1,w}]_i \end{pmatrix} \mapsto \begin{pmatrix} \bigoplus_{j=0}^{k-1} a_{0,j}^{(u)} [x_{j,u}]_i \oplus \alpha_{0,u} & \cdots & \bigoplus_{j=0}^{k-1} a_{0,j}^{(w)} [x_{j,w}]_i \oplus \alpha_{0,w} \\ \vdots & \ddots & \vdots \\ \bigoplus_{j=0}^{k-1} a_{k-1,j}^{(u)} [x_{j,u}]_i \oplus \alpha_{k-1,u} & \cdots & \bigoplus_{j=0}^{k-1} a_{k-1,j}^{(w)} [x_{j,w}]_i \oplus \alpha_{k-1,w} \end{pmatrix}. \quad (3.32)$$

Es ist klar, dass (3.32) invertierbar ist, wenn alle  $l$  Matrizen  $A^{(u)}, \dots, A^{(w)}$  regulär sind, denn dann können die Bit-Slices aller physischen Variablen eindeutig aus den Bit-Slices des Funktionsergebnisses bestimmt werden.

Die additiven Parameter  $\alpha_{i,j}$  können nun beliebig gemäß den Konstruktionsregeln aus Abschnitt 3.3.3.2 gewählt werden, wobei als Variablen in dem Parameter  $\alpha_{.,v}$  zwecks Erhaltung der T-Funktionseigenschaft nur die  $x_{.,v}$  aus der gleichen Spalte sowie die  $x_{.,w}$  aus allen niederwertigeren Spalten verwendet werden dürfen. Zusätzlich dazu können, wie in Abbildung 3.5 auf Seite 68 illustriert, beliebige Funktionen (also auch Nicht-T-Funktionen) der niederwertigeren physischen Variablen als Parameter verwendet werden. Das erlaubt es uns, in den additiven Parametern auch diejenigen gängigen Prozessorinstruktionen einzusetzen, welche keine T-Funktionen sind, insbesondere also

- Rechtsshift: Mit  $0 \leq y < n$  gilt

$$[x \curvearrowright y]_i = \begin{cases} [x]_{i+y}, & i = 0, \dots, n - y - 1 \\ 0, & i = n - y, \dots, n - 1; \end{cases}$$

- Zyklische Linksrotation:

$$[x \circlearrowleft y]_i = [x]_{(i-y) \bmod n} \quad \text{für } i \geq 0;$$

sowie

- Zyklische Rechtsrotation:

$$[x \circlearrowright y]_i = [x]_{(i+y) \bmod n} \quad \text{für } i \geq 0;$$

wobei  $n$  wie üblich die Wortgröße bezeichnet.

**Beispiel 3.25.** Um eine invertierbare breite T-Funktion in  $k = 2$  logischen Variablen à  $l = 2$  physischen Worten zu konstruieren (hier gilt  $k = l$  nur, um das Beispiel klein zu halten), brauchen wir nach vorigen Überlegungen zwei invertierbare  $2 \times 2$ -Matrizen, etwa  $A^{(u)} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$  und  $A^{(v)} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ . Damit ergibt sich die folgende schematische Form für die Bit-Slices:

$$\begin{pmatrix} [x_{0,u}]_i & [x_{0,v}]_i \\ [x_{1,u}]_i & [x_{1,v}]_i \end{pmatrix} \mapsto \left( A^{(u)} \begin{pmatrix} [x_{0,u}]_i \\ [x_{1,u}]_i \end{pmatrix} \oplus \begin{pmatrix} \alpha_{0,u} \\ \alpha_{1,u} \end{pmatrix} \quad A^{(v)} \begin{pmatrix} [x_{0,v}]_i \\ [x_{1,v}]_i \end{pmatrix} \oplus \begin{pmatrix} \alpha_{0,v} \\ \alpha_{1,v} \end{pmatrix} \right),$$

ausmultipliziert also:

$$\begin{aligned} \begin{pmatrix} [x_{0,u}]_i & [x_{0,v}]_i \\ [x_{1,u}]_i & [x_{1,v}]_i \end{pmatrix} &\mapsto \begin{pmatrix} a_{0,0}^{(u)} [x_{0,u}]_i \oplus a_{0,1}^{(u)} [x_{1,u}]_i \oplus \alpha_{0,u} & a_{0,0}^{(v)} [x_{0,v}]_i \oplus a_{0,1}^{(v)} [x_{1,v}]_i \oplus \alpha_{0,v} \\ a_{1,0}^{(u)} [x_{0,u}]_i \oplus a_{1,1}^{(u)} [x_{1,u}]_i \oplus \alpha_{1,u} & a_{1,0}^{(v)} [x_{0,v}]_i \oplus a_{1,1}^{(v)} [x_{1,v}]_i \oplus \alpha_{1,v} \end{pmatrix} \\ &= \begin{pmatrix} [x_{1,u}]_i \oplus \alpha_{0,u} & [x_{0,v}]_i \oplus [x_{1,v}]_i \oplus \alpha_{0,v} \\ [x_{0,u}]_i \oplus \alpha_{1,u} & [x_{1,v}]_i \oplus \alpha_{1,v} \end{pmatrix}. \end{aligned}$$

Die Parameter  $\alpha_{\cdot,v}$  müssen dabei T-Funktionen sein, während wir  $\alpha_{\cdot,u}$  auch als beliebige Funktionen der Variablen  $x_{\cdot,v}$  wählen dürfen, etwa so:

$$\begin{pmatrix} x_{0,u} & x_{0,v} \\ x_{1,u} & x_{1,v} \end{pmatrix} \mapsto \begin{pmatrix} x_{1,u} \oplus (x_{0,v} \uparrow 3) & x_{0,v} - x_{1,v} \\ x_{0,u} + (x_{0,v} x_{1,v} \circlearrowleft 7) & x_{1,v} \oplus 4(x_{0,v} \vee x_{1,v}) \end{pmatrix}. \quad (3.33)$$

Aus der Konstruktion folgt, dass (3.33) eine invertierbare T-Funktion ist, obwohl sie nicht ausschließlich aus T-Funktionen zusammengesetzt ist.  $\square$

# Kapitel 4

## T-Funktionsbasierte Blockchiffren

Mit Hilfe der Konstruktionstechniken aus Abschnitt 3.3.3 können komplexe nichtlineare und dabei dennoch konstruktionsbedingt invertierbare Abbildungen für beliebige Wortbreiten konstruiert werden. Folglich liegt es nahe, den Einsatz von T-Funktionen in der Rundenfunktion von iterierten Blockchiffren zu erwägen, und zwar sowohl in der Konfusions-Ebene (als S-Box) als auch in der Diffusions-Ebene (an Stelle von Bitpermutationen oder linearen Transformationen).

In der Blockchiffre RC6, einem AES-Finalisten, wurde die T-Funktion  $f(x) = 2x^2 + x$  als Bestandteil der Rundenfunktion verwendet [33], allerdings zusammen mit datenabhängigen zyklischen Rotationen, welche bekanntlich keine T-Funktionen sind. Im Folgenden sollen Möglichkeiten zum Einsatz reiner T-Funktionen als Konfusions- sowie Diffusionselemente in Blockchiffren untersucht werden.

### 4.1 T-Funktionen als Konfusionelemente

Zunächst betrachten wir die Konstruktion von S-Boxen aus nichtlinearen T-Funktionen. Für diesen Einsatz sollte eine Funktion  $f$  besonders gute lineare und differentielle Eigenschaften aufweisen, d.h.  $\Lambda_f$  und  $\Omega_f$  sollten so klein wie möglich sein. Dabei konzentrieren wir uns auf bijektive S-Boxen, d.h. invertierbare T-Funktionen. Gründe dafür sind in erster Linie die Einsetzbarkeit in Substitutions-Permutations-Netzwerken sowie die potenziell kryptoanalytisch ausnutzbare inkrementelle Einschränkung des Wertebereichs bei nicht surjektiven Rundenfunktionen [31, 32].

Als Konfusionelemente besitzen T-Funktionen einen unmittelbar aus ihrer Definition ableitbaren Nachteil gegenüber allgemeinen Abbildungen: Da das  $i$ -te Bit der Ausgabe nur von den Bits  $i, \dots, 0$  der Eingabe abhängen kann, propagieren Nulldifferenzen in den niederwertigen Eingabebits auf die Ausgabedifferenz durch:

**Satz 4.1** ([14]). *Sei  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$  eine T-Funktion,  $u, v \in \mathbb{F}_2^n$ , und die der Differenzbildung zu Grunde liegende Gruppenoperation  $\odot$  eine T-Funktion (etwa  $\oplus$  oder  $+$ ). Gilt  $[u]_{0,\dots,i} = 0$ , so folgt mit*

$$v = \Delta(f(x \odot u), f(x))$$

$[v]_{0,\dots,i} = 0$  für alle  $x \in \mathbb{F}_2^n$ . □

Werden also die Eingabebits  $0, \dots, i$  nicht verändert, so verändern sich diese Bits in der Ausgabe ebenfalls nicht. Die Aussage dieses Satzes gilt analog auch für T-Funktionen

in mehreren Variablen; bei der in Abschnitt 3.3.3.6 diskutierten Konstruktionsvariante für breite T-Funktionen können zumindest in den höherwertigen Spalten nichtprimitive Operationen wie zyklische Rotationen oder Rechtsshifts der niederwertigen Spalten als Parameter eingesetzt werden, was ein komplexeres Abhängigkeitsverhalten verspricht.

Sollen T-Funktionen in mehreren Veränderlichen oder breite T-Funktionen als S-Box verwendet werden, so ergibt sich dabei jedoch eine kleine technische Schwierigkeit: Die Komponenten der Rundenfunktion einer iterierten Blockchiffre bilden von  $\mathbb{F}_2^n$  auf  $\mathbb{F}_2^m$ , also Bitvektoren auf Bitvektoren ab, während mehrdimensionale T-Funktionen  $\phi : \mathbb{F}_2^{k \times n} \rightarrow \mathbb{F}_2^{l \times n}$  Bitmatrizen auf Bitmatrizen abbilden. Des Weiteren sind auch die Differenzen- und die lineare Approximationstabelle und damit die Linearitätsmaße  $\Lambda_f, \Omega_f$  nur für Funktionen der Form  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^q$  erklärt. Es ist also eine Konversion nötig: Wir identifizieren mit den  $k$   $n$ -Bit-Worten  $x \in \mathbb{F}_2^{k \times n}$  den Vektor  $x' \in \mathbb{F}_2^{kn}$  vermöge

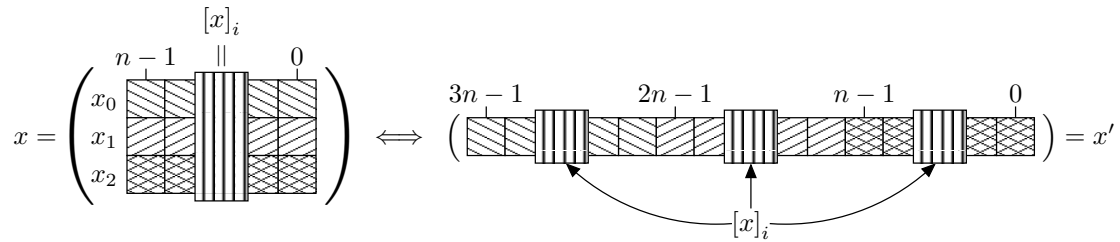
$$\begin{aligned} x &= \begin{pmatrix} x_0 \\ \vdots \\ x_{k-1} \end{pmatrix} \in \mathbb{F}_2^{k \times n} \\ \Updownarrow & \\ x' &= ([x']_{kn-1}, \dots, [x']_0) \\ &= ([x_0]_{n-1}, \dots, [x_0]_0, \dots, [x_{k-1}]_{n-1}, \dots, [x_{k-1}]_0) \in \mathbb{F}_2^{kn}. \end{aligned} \tag{4.1}$$

Die Bit-Slice  $[x]_i = ([x_0]_i, [x_1]_i, \dots, [x_{k-1}]_i)^T$  entspricht also  $([x']_{(k-1)n+i}, [x']_{(k-2)n+i}, \dots, [x']_i)$  (siehe Abbildung 4.1a auf der nächsten Seite), und das Skalarprodukt  $x \bullet y$  für  $x, y \in \mathbb{F}_2^{k \times n}$  ist erklärt als  $x' \bullet y'$ . Für breite T-Funktionen in  $l$  physischen Variablen wird spaltenweise verfahren, wie in Abbildung 4.1b auf der nächsten Seite illustriert. Da die durch (4.1) beschriebene Zuordnung bijektiv ist, wird die Konversion im Folgenden wo nötig implizit vorausgesetzt und nicht mehr explizit notiert.

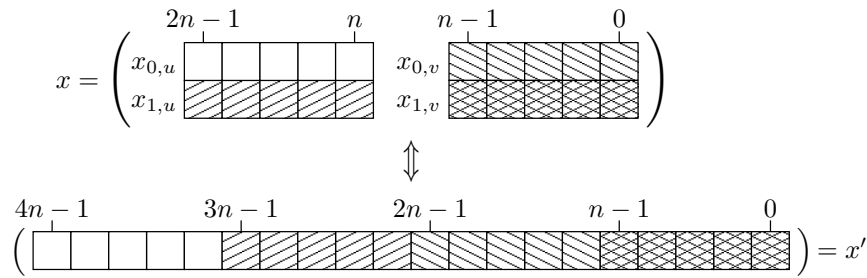
### 4.1.1 Untersuchungsansatz

Zur Ermittlung von invertierbaren T-Funktionen mit guten Konfusioneigenschaften bietet sich die Vorgehensweise an, alle (oder möglichst viele) Funktionen in einer bestimmten Anzahl von Variablen systematisch aufzuzählen und jeweils für einige kleine Wortbreiten  $n$  eine Berechnung des differentiellen sowie linearen Potenzials vorzunehmen. Auf Basis der experimentellen Ergebnisse können dann geeignet erscheinende T-Funktionen ausgewählt und einer allgemeinen analytischen Untersuchung unterzogen werden.

Generell besteht das Ziel einer solchen dann darin, entweder die für einige kleine  $n$  experimentell bestimmten Eigenschaften konkreter Funktionen auf allgemeines  $n$  zu übertragen, oder besser noch Aussagen über die Konfusionsgüte ganzer Klassen von T-Funktionen zu erhalten. Ein weiterer Ansatz besteht darin, einen Versuch zu unternehmen, die Bestimmung der Linearitätsmaße einer Funktionsklasse auf eine Analyse ihrer Bit-Slices herunterzubrechen (was von Klimov und Shamir bereits für Eigenschaften wie Invertierbarkeit oder Zyklenstruktur erfolgreich durchgeführt wurde). Einige Ergebnisse in dieser Richtung sind in Abschnitt 4.1.4 zu finden.



(a) T-Funktion in 3 Variablen.



(b) Breite T-Funktion in 2 logischen à 2 physischen Variablen.

Abbildung 4.1: Repräsentation von T-Funktionen in mehreren Veränderlichen als vektorielle Abbildungen.

Da der Invertierbarkeitstest nicht automatisierbar ist, lässt sich der Suchraum bei der Enumeration invertierbarer T-Funktionen nicht komplett ausschöpfen, sondern immer nur bezüglich einer bestimmten Konstruktionsform (siehe Abschnitte 3.3.3.3–3.3.3.6). Die drei Suchdimensionen für eine Enumeration – Konstruktionstyp, Variablenanzahl, Wortbreite – sind in Abbildung 4.2 auf der nächsten Seite illustriert.

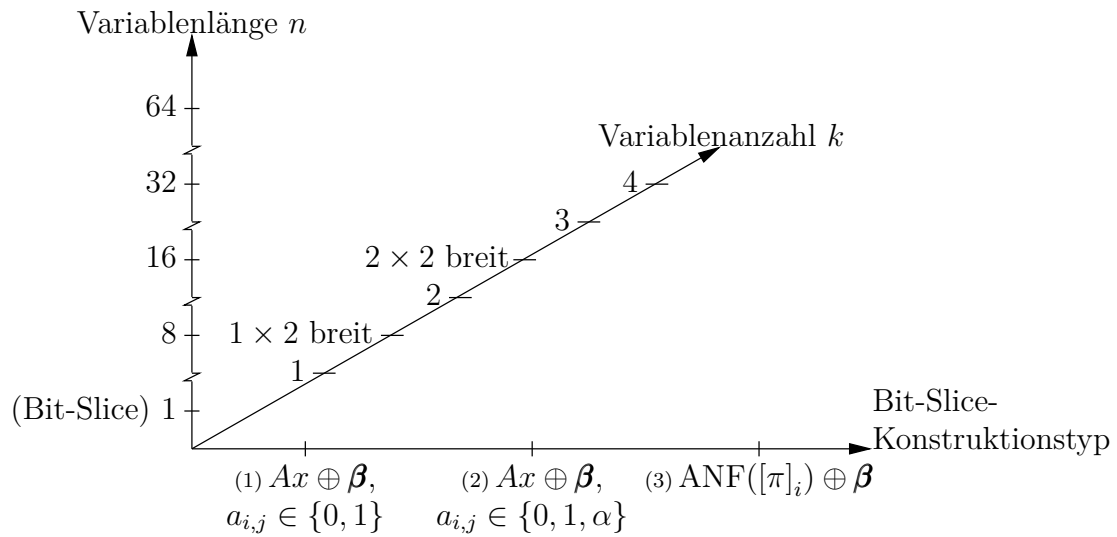
### 4.1.2 Linearitätsmaße

Die zur Bewertung der Konfusionseigenschaften der enumerierten T-Funktionen herangezogenen Linearitätsmaße sind diejenigen der differentiellen und linearen Kryptoanalyse, also das differentielle und lineare Potenzial, welche sich jeweils als nichttriviale Maxima des Differenzenprofils bzw. des Linearitätsprofils bestimmen lassen.

Das Differenzenprofil selbst ist durch Umnormierung leicht aus der Differenzentabelle zu erhalten; um Ganzzahlarithmetik verwenden zu können, beschränken wir uns sogar auf die Differenzentabelle  $\#D_f$ . Sie wird nach Algorithmus 4.1 auf der nächsten Seite berechnet (welcher hier für unsere Zwecke nur für Funktionen zwischen gleichdimensionalen  $\mathbb{F}_2$ -Vektorräumen formuliert ist). Er erfordert  $\Theta(2^{2n})$  Zeit und ebensoviel Speicherplatz.

Eine ähnlich direkte Berechnung des Linearitätsprofils über dessen Definition wäre sehr ineffizient. Zur Bestimmung von  $\#L_f$  (aus dem dann  $\lambda_f$  abgeleitet werden kann) müsste für jedes  $\alpha, \beta \in \mathbb{F}_2^n$  für alle  $x \in \mathbb{F}_2^n$  gezählt werden, wie oft  $\alpha \bullet x = \beta \bullet f(x)$  gilt. Dies erfordert mindestens<sup>1</sup>  $\Theta(2^{3n})$  Operationen. Wesentlich effizienter ist die Berechnung

<sup>1</sup>Die Skalarprodukte sind in der Regel nicht in  $\mathcal{O}(1)$  berechenbar.



Die unterbrochene Hochachse signalisiert, dass jede Ganzzahl zwischen den angegebenen prinzipiell als Variablenlänge in Frage kommt. Bei den Konstruktionstypen bezeichnet  $\beta$  jeweils einen Vektor von additiven Parametern,  $\alpha$  einen multiplikativen Parameter. Im Falle breiter T-Funktionen dürfen in den höheren Spalten auch Nicht-T-Funktionen als Parameter auftreten. Auch wenn die Bit-Slices der Formen (1) und (2) linear sind, können sie nichtlineare Funktionen beschreiben; die ANF-basierte Konstruktion aus Abschnitt 3.3.3.5 lässt nichtlineare Bit-Slices zu. Eine konkrete Ausprägung der Suchparameter kann damit etwa so aussehen: „2 Variablen,  $n = 8$ , Konstruktionstyp (2)“.

Abbildung 4.2: Zusammenfassung des Entwurfs- und Suchraums für invertierbare T-Funktionen.

---

**Algorithmus 4.1** Berechnung der Differenzentabelle einer Boole’schen Abbildung.

---

**Input:** Funktion  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ .

**Output:** Differenzentabelle  $\#D_f$ .

- 1:  $\forall u, v \in \mathbb{F}_2^n : \#D_f(u, v) \leftarrow 0$
  - 2: **for all**  $x \in \mathbb{F}_2^n$  **do**
  - 3:   **for all**  $x' \in \mathbb{F}_2^n$  **do**
  - 4:      $x^* \leftarrow x \oplus x'$
  - 5:      $y' \leftarrow f(x) \oplus f(x^*)$
  - 6:      $\#D_f(x', y') \leftarrow \#D_f(x', y') + 1$
  - 7:   **end for**
  - 8: **end for**
  - 9: **return**  $\#D_f$
-

mit Hilfe der schnellen Walsh-Transformation, wobei Korollar 2.56 angewandt wird. Da Algorithmus 2.3 in-place arbeitet, erfordert dieser Weg zusätzlich zur Berechnung der Differenzentabelle lediglich  $\Theta(2n \cdot 2^{2n})$  Zeit und keinen weiteren Speicher. Um auch hier auf Ganzzahlen operieren zu können,<sup>2</sup> wird an Stelle des Linearitätsprofils der Absolutbetrag der linearen Approximationstabelle errechnet (aus welchem sich  $\lambda_f$  bestimmen lässt), siehe Algorithmus 4.2. Dessen Umrechnungsschritt kann selbstverständlich ohne Umkopieren erfolgen.

---

**Algorithmus 4.2** Effiziente Berechnung der linearen Approximationstabelle.

---

**Input:** Differenzentabelle  $\#D_f$  der Funktion  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ .

**Output:** Lineare Approximationstabelle von  $f$  (betragsmäßig, also  $|\text{LAT}_f|$ ).

1: Berechne die Walsh-Transformierte von  $\#D_f$  in-place mit Algorithmus 2.3.

2: **for all**  $\alpha \in \mathbb{F}_2^n$  **do**

3:   **for all**  $\beta \in \mathbb{F}_2^n$  **do**

4:      $L(\alpha, \beta) \leftarrow \frac{1}{2} \sqrt{\widehat{\#D_f}(\alpha, \beta)}$

5:   **end for**

6: **end for**

7: **return**  $L$

---

Aus den angegebenen Komplexitäten ist klar ersichtlich, dass diese Berechnung auf Privatrechnern nur für etwa  $n \leq 14$  durchführbar ist, insbesondere infolge des Speicherbedarfs. Infolgedessen müssen wir uns bei der experimentellen Untersuchung auf T-Funktionen in wenigen Variablen kleiner Wortbreite beschränken, da für eine T-Funktion in  $k$  Variablen à  $n$  Bit  $2^{2kn}$  Tabelleneinträge erforderlich sind.

### 4.1.3 Funktions- und Parameter-Enumeration

Die Komplexität der Linearitätsmaßberechnung schränkt den Suchraum, wie er in Abbildung 4.2 dargestellt ist, zwar in den Parametern  $k$  und  $n$ , jedoch nicht bezüglich der Bit-Slice-Formen ein. Tabelle 4.1 auf der nächsten Seite gibt eine Übersicht, welche Kombinationen aus Variablenanzahl und Wortbreite innerhalb der verfügbaren Ressourcen zur experimentellen Untersuchung verwendet wurden. Da der Konstruktionstyp unabhängig von der Wortbreite ist, kann ein und dieselbe enumerierte T-Funktion mit mehreren Wortbreiten untersucht werden.

Innerhalb der Konstruktionstypen schränkt die Variablenanzahl die Anzahl aufzuzählender Ausprägungen ein, zum Beispiel sind für  $k = 2$  und die Bit-Slice-Form „Lineare Bit-Slices mit additiven Parametern“ bis auf die Parameter nur so viele Ausprägungen möglich, wie es invertierbare binäre  $2 \times 2$ -Matrizen gibt, also 6. Auf Einschränkungen bei der Aufzählung der Parameter wird weiter unten eingegangen.

**Technische Vorgehensweise** Eine T-Funktion wird über ihren abstrakten Syntaxbaum dargestellt, wobei jeder innere Knoten Instanz einer Operationsklasse (**Xor**,

---

<sup>2</sup>Die Quadratwurzel in Algorithmus 4.2 wird nur auf ganze Quadratzahlen angewandt.

Variablenanzahl	Bits je Variable
1	4, 8
1 × 2 breit	2, 4
2	2, 4
2 × 2 breit	1, 2, 3
3	2, 3, 4
4	1, 2, 3

Tabelle 4.1: Untersuchte Kombinationen aus Variablenanzahl und Wortbreite.

**Subtraction, ...**) ist und die Blätter Instanzen einer Variablen- oder Konstanten-Klasse sind. Jede Klasse verfügt neben der Fähigkeit zur Auswertung des bei ihr beginnenden Teilbaums (`evaluate()`) über eine Möglichkeit, diesen als gültigen Ausdruck in der Programmiersprache C auszugeben. Dies wurde dazu genutzt, die Objektmodellierung und Funktions-Enumeration in Python implementieren zu können, wohingegen die Berechnung der Linearitätsmaße aus Effizienzgründen in C erfolgt. Ein Python-Programm zur Aufzählung der bestimmten Suchparametern genügenden T-Funktionen erzeugt also systematisch alle Syntaxbäume und gibt diese in Form von C-Ausdrücken aus. Auf diese Weise entsteht eine alle Funktionen beinhaltende C-Quelldatei, die mit dem C-Code zur Linearitätsmaßberechnung zusammengelinkt und ausgeführt werden kann. Die Auswertung der dabei entstehenden Logdatei erfolgt dann wiederum in Python bzw. mit Standard-Unix-Werkzeugen zur Textbearbeitung.

**Matrizen** Zur Enumeration von T-Funktionen mit linearer Bit-Slice-Form werden alle invertierbaren  $k \times k$ -Matrizen benötigt. In Ermangelung einer Vorschrift zur *gezielten* Aufzählung regulärer Matrizen werden dazu alle binären  $k \times k$ -Matrizen aufgezählt und auf Invertierbarkeit getestet. Die Aufzählung selbst erfolgt dabei über eine Identifizierung der Zahlen  $0, \dots, 2^{k^2} - 1$  mit den binären  $k \times k$ -Matrizen; der Invertierbarkeitstest bringt die Matrix  $A$  mittels Gauss-Jordan-Elimination in Zeilenstufenform und berechnet das Produkt deren Hauptdiagonale, welches genau dann 0 beträgt, wenn  $A$  singular ist.

Für multiplikative Parameter werden parametrisierte invertierbare Matrizen benötigt, also Matrizen mit Einträgen aus  $\{0, 1, \alpha\}$ , welche für alle 0/1-Wertkombinationen der  $\alpha$ -Einträge regulär sind. Solche  $k \times k$ -Matrizen mit  $p$  Platzhaltern für Parameter können nur für  $p \leq k \cdot (k - 1) / 2$  parametrisch invertierbar sein, da ansonsten in Zeilenstufenform ein Element der Hauptdiagonale ein Platzhalter sein müsste, welcher auch den Wert 0 annehmen kann. Allgemein gibt es  $\binom{k^2}{p}$  mögliche Positionierungen der  $p$  Platzhalter und in einer solchen Positionierung jeweils  $2^{k^2-p}$  Varianten für die restlichen 0/1-Einträge. Für festes  $k$  und  $p$  werden also  $\binom{k^2}{p} \cdot 2^{k^2-p}$  Matrizen aufgezählt, welche für alle  $2^p$  Wertkombinationen der  $p$  Parameter auf Invertierbarkeit getestet werden müssen. Für unsere Zwecke wird jedoch maximal  $k = 3$  benötigt, was  $p \leq 3$  nach sich zieht. Ein Beispiel: Für  $k = 3, p = 2$  gibt es  $\binom{9}{2} \cdot 2^{9-2} = 4608$  Matrizenformen, von denen 180 parametrisch invertierbar sind. Die Aufzählung selbst erfolgt nach einem einfachen rekursiven Verfahren,

das in Algorithmus 4.3 dargestellt ist.

---

**Algorithmus 4.3** Aufzählung parametrisierter  $k \times k$ -Matrizen.

---

**Input:** Dimension  $k$ , Parameteranzahl  $p$ .

**Output:** Binäre  $k \times k$ -Matrizen mit  $p$  Platzhaltern (in linearisierter Darstellung als  $k \cdot k$ -Vektor  $V$ ).

1: ENUM-PMATRICES( $p, \emptyset$ )     $\triangleright$  Rekursionsstart: Noch  $p$  übrig, keine Position belegt

ENUM-PMATRICES( $p, ppos$ ):

1:  $freepos \leftarrow \{0, \dots, k^2 - 1\} \setminus ppos$

2: **if**  $p = 0$  **then**     $\triangleright$  Rekursionsabbruch,  $p$  Plätze gefüllt

3:    **for all**  $i \in ppos$  **do**

4:         $V[i] \leftarrow 'a'$

5:    **end for**

$\triangleright$  Alle Variationen der übrigen Plätze zurückliefern:

6:    **for**  $i = 0 \dots 2^{\#freepos} - 1$  **do**

7:         $j \leftarrow 0$

8:        **for all**  $f \in freepos$  **do**

9:             $V[f] \leftarrow (i \uparrow j) \bmod 2$

10:           $j \leftarrow j + 1$

11:        **end for**

12:        **yield**  $V$

13:    **end for**

14: **else**

15:     $\triangleright$  Nur freie Positionen an höheren Indizes besetzen, um Duplikate zu vermeiden:

16:     $maxpos \leftarrow \max_{x \in ppos} x$

17:    **for all**  $pos \in \{x \in freepos \mid x > maxpos\}$  **do**

18:        **for all**  $V \in \text{ENUM-PMATRICES}(p - 1, ppos \cup \{pos\})$  **do**

19:            **yield**  $V$

20:        **end for**

21:    **end for**

22: **end if**

---

**Permutationen** Für die ANF-basierte Konstruktion nichtlinearer Bit-Slice-Formen (siehe Abschnitt 3.3.3.5) müssen systematisch alle Permutationen des  $\mathbb{F}_2^k$  aufgezählt werden. Setzt man  $n := 2^k$ , so erfolgt dies mittels einer Identifizierung der Zahlen von 0 bis  $n! - 1$  mit den  $n!$  möglichen Permutationen der Zahlen  $0, \dots, n - 1$ : Jede Zahl  $f$  mit  $0 \leq f < n!$  lässt sich eindeutig im „faktoriellen Zahlensystem“ als

$$f = (n - 1)!c_{n-1} + (n - 2)!c_{n-2} + \dots + 1!c_1$$

schreiben, wobei  $0 \leq c_j \leq j$  für  $1 \leq j < n$  [22]. Für die Bestimmung der zur Zahl  $f$  korrespondierenden Permutation  $\pi$  lässt sich dabei die Rückwärtsvariante von Knuths

„Algorithm P“ verwenden, die in Algorithmus 4.4 formuliert ist (siehe [22] Abschnitt 3.3.2 F).

---

**Algorithmus 4.4** NUMBER-TO-PERMUTATION

---

**Input:** Zahl  $f$  mit  $0 \leq f < n!$ .

**Output:** Wertetabelle der  $f$  zugeordneten Permutation  $\pi \in S(\{0, \dots, n-1\})$ ,

also  $\pi[i] = \pi(i)$  für  $0 \leq i < n$ .

1:  $\pi \leftarrow [0, 1, \dots, n-1]$   $\triangleright$  Setze  $\pi$  auf die Identitätspermutation

2: **for**  $r = 2 \dots n$  **do**

3:  $s \leftarrow f \bmod r$

4:  $f \leftarrow \lfloor f/r \rfloor$

$\triangleright$  Vertausche  $\pi[r-1]$  und  $\pi[s]$ :

5:  $tmp \leftarrow \pi[r-1]$

6:  $\pi[r-1] \leftarrow \pi[s]$

7:  $\pi[s] \leftarrow tmp$

8: **end for**

9: **return**  $\pi$

---

Die eigentliche Aufzählung der Permutationen ist damit leicht möglich. In Algorithmus 4.5 auf der nächsten Seite wird zusätzlich noch die Möglichkeit berücksichtigt, lediglich eine zufällige Auswahl von  $r < n!$  verschiedenen Permutationen aufzählen zu lassen.

**Parameter** Unabhängig von der Konstruktion des Funktionsgerüsts aus der Bit-Slice-Form kann eine Menge von Parametern in einer bestimmten Anzahl Variablen erzeugt werden, welche dann beliebig an all diejenigen Stellen des Gerüsts eingesetzt werden können, wo dieses Parameter vorsieht (additiv wie multiplikativ).

„Rezepte“ für die Konstruktion von Parametern aus Ausdrücken wurden bereits in Abschnitt 3.3.3.2 ausführlich geschildert. Die Aufzählung von Parametern erfolgt direkt anhand dieser rekursiven Regeln, und da Parameter T-Funktionen sind, können für ihre Konstruktion die gleichen Syntaxbaumklassen eingesetzt werden. Zwei Aspekte sind dabei jedoch näher zu betrachten: Erstens muss ein Abbruchkriterium für die ansonsten endlos rekursiv anwendbaren Regeln festgelegt werden. Die Anzahl verschiedener Variablen ist dabei allein noch keine wirksame Beschränkung, da diese in Parametern der Form  $2x_0x_1x_0\dots$  beliebig oft vorkommen können. Zusätzlich ist also die Anzahl der *Verwendungen* der einzelnen Variablen zu limitieren; diese „Variablenbilanz“ wird als zusätzlicher Rekursionsparameter mitgeführt und verwaltet.

Als sinnvolle Beschränkung hat sich die Regel „bis zu 2 der  $k$  Variablen jeweils maximal 2 mal verwendet“ erwiesen. Auf diese Weise werden durch das Regelsystem 37 verschiedene Parameter in einer Variablen und 960 verschiedene Parameter in (bis zu) zwei Variablen aufgezählt, was bereits für lineare Bit-Slices in zwei Variablen mit additiven Parametern knapp eine Million Variationsmöglichkeiten ergibt.

Als zweiter Aspekt ist zu beachten, dass nicht alle auf diese Weise erzeugten Parameter auch Parameter in der LSB sind. Um die Korrekturtechniken aus Tabelle 3.3 nicht

---

**Algorithmus 4.5** Aufzählung von Permutationen

---

**Input:** Zahl  $n$ , Zahl  $random$ .

**Output:** Wenn  $random = 0$ : Alle Permutationen der Zahlen  $0, \dots, n - 1$  aufzählen, sonst  $random$  viele zufällig ausgewählte.

```
1:  $F \leftarrow n!$ 
2: if  $random > 0$  then    ▷ Nur  $random$  viele aufzählen
3:    $used \leftarrow \{\}$ 
4:   for  $i = 1 \dots random$  do
5:     repeat
6:        $x \leftarrow \text{RANDOM}(0, F)$     ▷  $x \in_R \{0, \dots, F - 1\}$ 
7:     until  $x \notin used$ 
8:      $used \leftarrow used \cup \{x\}$ 
9:     yield NUMBER-TO-PERMUTATION( $x$ )
10:  end for
11: else    ▷ Alle aufzählen
12:  assert  $n \leq 12$ 
13:  for  $f = 0 \dots F - 1$  do
14:    yield NUMBER-TO-PERMUTATION( $f$ )
15:  end for
16: end if
```

---

unnötigerweise anzuwenden, testen wir die niederwertigste Bit-Slice jedes Parameters mit Algorithmus 4.6 auf der nächsten Seite und wenden gegebenenfalls eine Korrekturtechnik an. Der Algorithmus nutzt aus, dass die LSB eines Ausdrucks  $\alpha$  genau dann ein Parameter ist, wenn  $[\alpha(x_0, \dots, x_{k-1})]_0$  konstant ist, also für alle 1-Bit-Belegungen der Variablen  $x_0, \dots, x_{k-1}$  modulo 2 den gleichen Wert annimmt.

Neben den normalen Parametern können in den vorderen Spalten von breiten T-Funktionen auch nichtprimitive Operationen der hinteren Spalten eingesetzt werden. Dieser Spielraum wird stets voll ausgeschöpft; die zusätzlichen Operationen sind exakt diejenigen aus Abschnitt 3.3.3.6, nämlich zyklische Links- und Rechtsrotation sowie Rechtsshift.

#### 4.1.4 Differentielle und lineare Eigenschaften

In diesem Abschnitt werden die Ergebnisse der experimentellen Untersuchung der Konfusionsseigenschaften von T-Funktionen vorgestellt und besprochen. Darüber hinaus werden einige analytische Resultate formuliert und bewiesen. Dabei gehen wir nach Konstruktionstyp und Variablenanzahl geordnet vor.

##### 4.1.4.1 Lineare Bit-Slices und breite T-Funktionen

**1 Variable, lineare Bit-Slices** Für T-Funktionen in einer Veränderlichen kommen von den in Abschnitt 3.3.3 vorgestellten Konstruktionstechniken nur lineare Bit-Slices

---

**Algorithmus 4.6** Test auf Parametereigenschaft der LSB.

---

**Input:** Ausdruck  $\alpha$

**Output:** „True“, wenn  $[\alpha]_0$  ein Parameter ist, „False“ sonst.

```
1: variables  $\leftarrow$  VARIABLES( $\alpha$ )
2: if #variables = 0 then  $\triangleright$  Ausdrücke ohne Variablen sind stets Parameter
3:   return „True“
4: end if
5: values  $\leftarrow$   $\emptyset$ 
6: for  $i = 0 \dots 2^{\#variables} - 1$  do
7:    $v \leftarrow \alpha(x) \Big|_{x_j=[i]_j}$ 
8:   values  $\leftarrow values \cup \{v \bmod 2\}$ 
9: end for
10: return (#values = 1)
```

---

mit additiven Parametern in Frage: Die einzige invertierbare  $1 \times 1$ -Matrix ist die Zahl 1, so dass keine multiplikativen Parameter möglich sind; und wie in Beispiel 3.24 verdeutlicht, sind die Bit-Slices der beiden Permutationen auf  $\mathbb{F}_2^1$  affin, so dass die ANF-basierte Konstruktionsmethode hier keine nichtlinearen Bit-Slices liefern kann.

Die Bit-Slices aller aufgezählten T-Funktionen in einer Variablen  $x$  haben also die Form

$$[x]_i \mapsto [x]_i \oplus \alpha,$$

was bedeutet, dass die Funktionen selbst stets die Gestalt

$$f(x) = x \oplus \alpha(x) \tag{4.2}$$

besitzen, wobei  $\alpha$  ein beliebiger (und insbesondere auch nichtlinearer) Parameter sein kann.

Das in Abschnitt 4.1.3 beschriebene Verfahren zur Aufzählung von Parametern in einer Variablen liefert 37 solche Parameter. In Verbindung mit den drei Verknüpfungsoperationen  $+$ ,  $-$  und  $\oplus$  und dem Trivialparameter 0 ergeben sich also insgesamt  $3 \cdot 37 + 1 = 112$  Funktionen, von denen 63 nichtlinear sind, da sie Multiplikationen enthalten. Die anschließende Untersuchung auf deren lineare und differentielle Eigenschaften wurde mit einer Wortbreite von  $n = 4$  und  $n = 8$  Bit durchgeführt. Dabei stellt sich heraus, dass bei allen untersuchten Funktionen – auch den nichtlinearen – im Falle von  $n = 4$  die maximalen nichttrivialen Einträge in den Differenzen- bzw. linearen Approximationstabellen ausnahmslos  $2^4 = 16$  bzw.  $2^4 - 2^3 = 8$  betragen und damit  $\Lambda_f = \Omega_f = 1$  gilt. Diese Situation wiederholt sich bei  $n = 8$  mit Maximaleinträgen von  $2^8 = 256$  bzw.  $2^8 - 2^7 = 128$ , so dass auch hier  $\Lambda_f = \Omega_f = 1$  vorliegt. Damit haben alle Funktionen die für ihre Dimension schlechtestmöglichen differentiellen sowie linearen Potenziale, was zumindest etwas überraschend ist, da nichtlinearen Funktionen wie  $f(x) = x - ((\bar{x})^2 \wedge 2x)$  durchaus bessere Eigenschaften zugetraut werden konnten als einer schlichten affinen Funktion wie  $g(x) = x + a$ , welche das gleiche differentielle und lineare Potenzial besitzt. (Es ist jedoch festzuhalten, dass die Anzahl der Maximaleinträge meist klein ist,

im Gegensatz zu affinen Funktionen, wo in jeder Zeile bzw. Spalte ein Maximaleintrag vorkommt, siehe Korollare 2.49 und 2.53.)

In diesem Zusammenhang ist nun die Frage interessant, wie  $\Lambda_f$  und  $\Omega_f$  für allgemeines  $n$ , insbesondere  $n > 8$ , aussehen. Die experimentellen Ergebnisse für  $n = 4, 8$  legen die Vermutung nahe, dass T-Funktionen vom Typ (4.2) allgemein das schlechtestmögliche differentielle bzw. lineare Potenzial haben, was sie für die Konstruktion von S-Boxen vollkommen untauglich macht. In der Tat lässt sich diese Vermutung für beliebiges  $n$  bestätigen:

**Proposition 4.2.** *Sei  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$  eine T-Funktion der Form*

$$f(x) = x \oplus \alpha(x)$$

*mit einem beliebigen Parameter  $\alpha$ . Dann hat  $f$  das lineare Potenzial*

$$\Lambda_f = 1.$$

*Beweis.* Wir betrachten die durch die Vektoren  $u = v = (0 \ \cdots \ 0 \ 1) \in \mathbb{F}_2^n$  gegebene lineare Approximation  $u \bullet x = v \bullet f(x)$  von  $f$ :

$$(0 \ \cdots \ 0 \ 1) \bullet x = (0 \ \cdots \ 0 \ 1) \bullet f(x),$$

ausmultipliziert also

$$[x]_0 = [f(x)]_0 = [x]_0 \oplus [\alpha(x)]_0,$$

wegen der speziellen Form von  $f$ . Da nun  $[\alpha(x)]_0$  als LSB eines Parameters konstant 0 oder 1 sein muss, ergibt sich je nach Ausgestaltung dieses Parameters eine der beiden folgenden Relationen:

$$[x]_0 = [x]_0 \quad \text{oder} \quad [x]_0 = [x]_0 \oplus 1,$$

wovon die erste für alle, die zweite für kein  $x \in \mathbb{F}_2^n$  erfüllt ist. Folglich ist  $\#L_f(u, v)$  entweder  $2^n$  oder 0, was in beiden Fällen

$$\lambda_f(u, v) = \left( 2 \frac{\#L_f(u, v)}{2^n} - 1 \right)^2 = 1$$

bedeutet. Damit sind Vektoren  $u, v \neq 0$  gefunden, für die  $\lambda_f(u, v)$  den maximal möglichen Wert 1 annimmt. Also gilt für Funktionen vom Typ (4.2) generell  $\Lambda_f = 1$ .  $\square$

Beispiele für Parameter, die die beiden möglichen Fälle illustrieren, sind das oben erwähnte  $\alpha(x) = (\bar{x})^2 \wedge 2x$ , für welches sich wegen  $[\alpha(x)]_0 = [(\bar{x})^2]_0 \wedge [2]_0 \wedge [x]_0 = 0$  die mit Wahrscheinlichkeit 1 erfüllte Relation  $[x]_0 = [x]_0$  ergibt; sowie  $\beta(x) = x^2 \vee 1$ , bei dem wegen  $[\beta(x)]_0 = [x^2]_0 \vee [1]_0 = 1$  die mit Wahrscheinlichkeit 0 erfüllte Relation  $[x]_0 = [x]_0 \oplus 1 = \overline{[x]_0}$  entsteht.

Intuitiv besagt Proposition 4.2, dass sich im Falle linearer Bit-Slices mit additiven Parametern selbst für nichtlineare T-Funktionen eine lineare Approximation der LSB mit maximalem Bias angeben lässt – eine direkte Folge des eingeschränkten Abhängigkeitsverhaltens von T-Funktionen. Da nach Satz 3.17 eine T-Funktion  $f$  in einer Variablen genau dann invertierbar ist, wenn  $[f(x)]_i = [x]_i \oplus \alpha$  gilt, lässt sich die Aussage von Proposition 4.2 verschärfen zu

**Proposition 4.3.** *Jede invertierbare T-Funktion in einer Veränderlichen hat das lineare Potenzial  $\Lambda_f = 1$ .*  $\square$

Das Ergebnis im einvariablen Fall lässt sich direkt auf den Fall breiter T-Funktionen in einer logischen à zwei physischen Variablen übertragen, bei dem die Bit-Slices der Spalten ebenfalls als lineare Bit-Slices mit additiven Parametern konstruiert werden (vgl. Abschnitt 3.3.3.6): Ist  $f : \mathbb{F}_2^{2n} \rightarrow \mathbb{F}_2^{2n}$  eine breite T-Funktion der Form

$$f(x_u, x_v) = \left( x_u \overset{+}{\oplus} \alpha(x_v) \quad x_v \overset{+}{\oplus} \beta(x_v) \right) \quad (4.3)$$

mit einem regulären Parameter  $\beta$  sowie einer beliebigen Funktion  $\alpha$ , die insbesondere Rechtsschiebe- und zyklische Rotationen beinhalten kann, so blendet mit  $u = v = (0 \ \cdots \ 0 \ 1) \in \mathbb{F}_2^{2n}$  die lineare Approximation  $u \bullet x = v \bullet f(x)$  alle zur ersten Spalte gehörenden Terme aus, so dass exakt die im nicht breiten Fall betrachtete Relation der niederwertigsten Bit-Slices entsteht und die Herleitung von Proposition 4.2 sofort ergibt:

**Proposition 4.4.** *Sei  $f : \mathbb{F}_2^{2n} \rightarrow \mathbb{F}_2^{2n}$  eine breite T-Funktion der Form*

$$f(x_u, x_v) = \left( x_u \overset{+}{\oplus} \alpha(x_v) \quad x_v \overset{+}{\oplus} \beta(x_v) \right)$$

*mit einer beliebigen Funktion  $\alpha$  und einem Parameter  $\beta$ . Dann hat  $f$  das lineare Potenzial*

$$\Lambda_f = 1.$$

$\square$

Damit sind also auch breite T-Funktionen der Form (4.3) für die Konstruktion von S-Boxen ungeeignet.

**2 Variablen, lineare Bit-Slices** Bei T-Funktionen in 2 Variablen bestehen mehr Variationsmöglichkeiten bei der Anwendung der Konstruktionsprinzipien: Im Falle linearer Bit-Slices mit additiven Parametern kommen alle invertierbaren  $2 \times 2$ -Matrizen in Frage, und im Gegensatz zum univariaten Fall werden auch multiplikative Parameter möglich. Von den 16  $2 \times 2$ -Matrizen über  $\mathbb{F}_2$  sind 6 invertierbar, zusätzlich ergeben sich Variationsmöglichkeiten durch die Wahl der zwei additiven Parameter.

Ähnlich wie beim univariaten Fall ergeben Experimente für  $2n = 4$  und  $2n = 8$  die schlechtestmöglichen differentiellen und linearen Eigenschaften: Bei allen knapp 20'000 untersuchten T-Funktionen dieses Typs galt sowohl für 2 Variablen à 2 als auch für 2 Variablen à 4 Bit  $\Omega_f = \Lambda_f = 1$ . Auch hier stellt sich heraus, dass diese Resultate kein Zufall sind, denn unabhängig von der Wortbreite  $n$  hat keine nach dem Konstruktionsprinzip aus Abschnitt 3.3.3.3 erzeugte T-Funktion in 2 Variablen (auch nicht die nichtlinearen) ein besseres lineares Potenzial als affine Funktionen:

**Proposition 4.5.** *Sei  $f : \mathbb{F}_2^{2 \times n} \rightarrow \mathbb{F}_2^{2 \times n}$  eine T-Funktion mit Bit-Slices der Form*

$$\begin{pmatrix} [x_0]_i \\ [x_1]_i \end{pmatrix} \mapsto A \begin{pmatrix} [x_0]_i \\ [x_1]_i \end{pmatrix} \oplus \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}$$

mit einer invertierbaren Matrix  $A \in \mathbb{F}_2^{2 \times 2}$  und beliebigen additiven Parametern  $\beta_0, \beta_1$ . Dann hat  $f$  das lineare Potenzial

$$\Lambda_f = 1.$$

*Beweis.* Wie zuvor beschrieben, interpretieren wir  $f$  als Funktion auf  $\mathbb{F}_2^{2n}$ , indem wir den Vektor  $(x, y)^T \in \mathbb{F}_2^{2 \times n}$  als Vektor  $(x \ y) \in \mathbb{F}_2^{2n}$  schreiben. Zum Nachweis der Behauptung geben wir lineare Approximationen maximaler Wahrscheinlichkeit für  $f$  an. Dazu werden die invertierbaren Matrizen aus  $\mathbb{F}_2^{2 \times 2}$  in drei Äquivalenzklassen à jeweils 2 Elemente eingeteilt:

1. Fall:  $A \in \{A_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, A_2 = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}\}$ . In diesem Fall betrachten wir die durch die Vektoren  $u = (0 \ \cdots \ 0 \ 1)$ ,  $v = (\underbrace{0 \ \cdots \ 0}_{n-1} \ 1 \ \underbrace{0 \ \cdots \ 0}_n) \in \mathbb{F}_2^{2n}$  gegebene lineare

Approximation von  $f$ :

$$(0 \ \cdots \ 0 \ 1) \bullet (x_0 \ x_1) = (\underbrace{0 \ \cdots \ 0}_{n-1} \ 1 \ \underbrace{0 \ \cdots \ 0}_n) \bullet f(x_0, x_1),$$

was gleichbedeutend ist zu

$$\begin{aligned} [x_1]_0 &= [f(x_0, x_1)]_{n+1} \\ &= [x_1]_0 \oplus [\beta_0]_0. \end{aligned}$$

2. Fall:  $A \in \{A_3 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, A_4 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}\}$ . Hier wählen wir die Vektoren  $u = (0 \ \cdots \ 0 \ 1)$ ,  $v = (\underbrace{0 \ \cdots \ 0}_{n-1} \ 1 \ \underbrace{0 \ \cdots \ 0}_{n-1} \ 1) \in \mathbb{F}_2^{2n}$ , welche folgende lineare Approximation von  $f$  beschreiben:

$$(0 \ \cdots \ 0 \ 1) \bullet (x_0 \ x_1) = (\underbrace{0 \ \cdots \ 0}_{n-1} \ 1 \ \underbrace{0 \ \cdots \ 0}_{n-1} \ 1) \bullet f(x_0, x_1),$$

beziehungsweise ausmultipliziert:

$$\begin{aligned} [x_1]_0 &= [f(x_0, x_1)]_{n+1} \oplus [f(x_0, x_1)]_0 \\ &= \begin{cases} [x_0 \oplus x_1 \oplus \beta_0]_0 \oplus [x_0 \oplus \beta_1]_0, & \text{falls } A = A_3 \\ [x_0 \oplus \beta_0]_0 \oplus [x_0 \oplus x_1 \oplus \beta_1]_0, & \text{falls } A = A_4 \end{cases} \\ &= [x_1]_0 \oplus [\beta_0]_0 \oplus [\beta_1]_0. \end{aligned}$$

3. Fall:  $A \in \{A_5 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, A_6 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}\}$ . Dieser Fall ist vollkommen analog zur Situation bei einer Variablen. Wie dort betrachten wir die durch die Vektoren  $u = v = (0 \ \cdots \ 0 \ 1) \in \mathbb{F}_2^{2n}$  beschriebene lineare Approximation von  $f$ :

$$(0 \ \cdots \ 0 \ 1) \bullet (x_0 \ x_1) = (0 \ \cdots \ 0 \ 1) \bullet f(x_0, x_1),$$

also

$$\begin{aligned} [x_1]_0 &= [f(x_0, x_1)]_0 \\ &= [x_1]_0 \oplus [\beta_1]_0. \end{aligned}$$

Als LSB eines Parameters sind  $[\beta_0]_0$  und  $[\beta_1]_0$  (und damit auch ihr XOR) konstant 0 oder 1, so dass in allen drei Fällen die sich ergebende Relation entweder für alle oder für kein  $x \in \mathbb{F}_2^{2n}$  erfüllt ist. Damit ist stets  $\lambda_f(u, v) = 1$  für wie beschrieben gewählte  $u, v \neq 0$ , also  $\Lambda_f = 1$ , was zu zeigen war.  $\square$

Proposition 4.5 besagt, dass auch im Fall zweier Variablen die lineare Bit-Slice-Konstruktion für beliebiges  $n$  keine für den Einsatz als S-Box tauglichen T-Funktionen hervorbringt. Auch wenn die niederwertigsten Bit-Slices jetzt von den LSBs zweier Variablen abhängen können, so erlauben passend gewählte Vektoren dennoch die Angabe linearer Approximationen mit bestmöglichem Bias.

Wie im einvariablen Fall lässt sich Proposition 4.5 sofort auf breite T-Funktionen in zwei logischen à jeweils 2 physischen Variablen übertragen, die durch die Konstruktionsvorschrift aus Abschnitt 3.3.3.6 entstanden sind. Die Bit-Slices einer solchen Funktion haben die Form

$$\begin{pmatrix} [x_{0,u}]_i & [x_{0,v}]_i \\ [x_{1,u}]_i & [x_{1,v}]_i \end{pmatrix} \mapsto \left( A^{(u)} \begin{pmatrix} [x_{0,u}]_i \\ [x_{1,u}]_i \end{pmatrix} \oplus \begin{pmatrix} \beta_0([x_{\cdot,v}]_i) \\ \beta_1([x_{\cdot,v}]_i) \end{pmatrix} \quad A^{(v)} \begin{pmatrix} [x_{0,v}]_i \\ [x_{1,v}]_i \end{pmatrix} \oplus \begin{pmatrix} \gamma_0([x_{\cdot,v}]_i) \\ \gamma_1([x_{\cdot,v}]_i) \end{pmatrix} \right)$$

mit regulären  $2 \times 2$ -Matrizen  $A^{(u)}$  und  $A^{(v)}$ , bei denen die  $\beta_j$  beliebige Funktionen von  $[x_{\cdot,v}]_i$  sein können, während die  $\gamma_j$  normale Parameter und damit T-Funktionen sein müssen. Indem wir die im Beweis von Proposition 4.5 gewählten Vektoren  $u, v$  durch führende Nullen auf  $\mathbb{F}_2^{4n}$  erweitern, blenden wir in den linearen Approximationen die gesamte erste Spalte aus, in der nichtprimitive Operationen wie etwa Rotationen vorkommen können. Damit ergeben sich exakt die gleichen Relationen wie in Proposition 4.5 und folglich

**Proposition 4.6.** *Sei  $f : \mathbb{F}_2^{2 \times 2n} \rightarrow \mathbb{F}_2^{2 \times 2n}$  eine breite T-Funktion mit Bit-Slices der Form*

$$\begin{pmatrix} [x_{0,u}]_i & [x_{0,v}]_i \\ [x_{1,u}]_i & [x_{1,v}]_i \end{pmatrix} \mapsto \left( A^{(u)} \begin{pmatrix} [x_{0,u}]_i \\ [x_{1,u}]_i \end{pmatrix} \oplus \begin{pmatrix} \beta_0([x_{\cdot,v}]_i) \\ \beta_1([x_{\cdot,v}]_i) \end{pmatrix} \quad A^{(v)} \begin{pmatrix} [x_{0,v}]_i \\ [x_{1,v}]_i \end{pmatrix} \oplus \begin{pmatrix} \gamma_0([x_{\cdot,v}]_i) \\ \gamma_1([x_{\cdot,v}]_i) \end{pmatrix} \right)$$

mit invertierbaren Matrizen  $A^{(u)}, A^{(v)} \in \mathbb{F}_2^{2 \times 2}$ , beliebigen Funktionen  $\beta_0, \beta_1$  und Parametern  $\gamma_0, \gamma_1$ . Dann hat  $f$  das lineare Potenzial

$$\Lambda_f = 1.$$

$\square$

**3 Variablen, lineare Bit-Slices** Auch in 3 Variablen bessern sich die experimentell ermittelten differentiellen und linearen Potenziale von T-Funktionen mit linearen Bit-Slices nicht; bei allen untersuchten Funktionen gilt stets  $\Omega_f = \Lambda_f = 1$ . Die 168 invertierbaren binären  $3 \times 3$ -Matrizen lassen sich ähnlich wie im Beweis von Proposition 4.5 in 7 Äquivalenzklassen von jeweils 24 Matrizen einteilen. Eine direkte Übertragung der für  $k = 1, 2$  verwendeten Herleitung, welche die explizite Angabe linearer Approximationen erfordert, gestaltet sich folglich sehr aufwändig, so dass hier auf eine Verallgemeinerung der experimentellen Ergebnisse verzichtet wurde.

Anzahl Funktionen	6 Bit		9 Bit		12 Bit	
	DT	LAT	DT	LAT	DT	LAT
10752	16	16	128	128	1024	1024
18816	32	32	256	256	2048	2048
10752	64	32	512	256	4096	2048

Tabelle 4.2: Maximaleinträge der Differenzen- (DT) und linearen Approximationstabelle (LAT, betragsmäßig) von T-Funktionen mit nichtlinearen Bit-Slices in 3 Variablen.

**Multiplikative Parameter** Bislang wurde die lineare Bit-Slice-Form nur mit additiven Parametern betrachtet, für multiplikative Parameter ergibt sich jedoch exakt das gleiche Bild. Für zwei und eine Variable bestand die Idee darin, lineare Approximationen maximalem Bias zu erhalten, indem lediglich die LSBs gewisser Variablen und Parameter für die Approximation ausgewählt werden. Da auch die LSBs eines multiplikativen Parameters konstant 0 oder 1 sind, entstehen dabei die gleichen Relationen wie im Fall rein additiver Parameter. Multiplikative Parameter können die Konfusionseigenschaften dieser Konstruktionsform also nicht verbessern.

#### 4.1.4.2 Nichtlineare Bit-Slices

Nachdem kein Konstruktionsprinzip für invertierbare T-Funktionen, das auf linearen Bit-Slices aufbaut, eine Funktion mit akzeptablen differentiellen und linearen Eigenschaften liefern konnte, sollen nun die Eigenschaften von T-Funktionen untersucht werden, die durch das in Abschnitt 3.3.3.5 beschriebene Verfahren entstehen, welches ja für mehr als zwei Variablen nichtlineare Bit-Slices erlaubt.

Betrachten wir zunächst den Fall  $k = 3$ , also 3 Variablen. Hier gibt es  $2^3! = 40320$  Permutationen auf  $\mathbb{F}_2^3$ , deren algebraische Normalformen als Komponenten einer invertierbaren T-Funktion  $f : \mathbb{F}_2^{3 \times n} \rightarrow \mathbb{F}_2^{3 \times n}$  verwendet werden können. Zusätzlich können in einer Bit-Slice bis zu drei additive Parameter vorhanden sein. Eine vollständige Aufzählung aller dieser Funktionen (ohne Parameter) und eine Bestimmung der differentiellen und linearen Eigenschaften für  $n = 2, 3, 4$  (also S-Boxen der Größe 6, 9 und 12 Bit) ergibt die in Tabelle 4.2 dargestellten Ergebnisse. Dabei fällt auf, dass eine bestimmte Funktion  $f_i$  bei steigender Wortbreite stets „in ihrer Kategorie“ bleibt; ist etwa  $f_{405}$  bei 6 Bit in der Kategorie (16, 16), also  $\Lambda_{f_{405}} = (2 \cdot \frac{16}{64})^2 = \frac{1}{4}$  sowie  $\Omega_{f_{405}} = \frac{16}{64} = \frac{1}{4}$ , so fällt  $f_{405}$  bei 9 bzw. 12 Bit in die Kategorie (128, 128) bzw. (1024, 1024), was ebenfalls  $\Lambda_{f_{405}} = \Omega_{f_{405}} = \frac{1}{4}$  bedeutet.

Diese Beobachtung legt die Vermutung nahe, dass sich bei diesen Funktionen das lineare bzw. differentielle Potenzial mit zunehmender Wortbreite nicht ändert, also im Umkehrschluss bereits durch Untersuchung der Funktion für kleine  $n$  zu bestimmen wäre. Da in den Experimenten kein Unterschied in den Potenzialen festgestellt werden konnte, wenn additive Parameter variiert oder weggelassen wurden, konzentrieren wir uns auf die „Reinform“ (3.30) dieses Funktionstyps. Da es zu jeder Permutation  $\pi$  bis auf Umordnung der Zeilen genau eine Bit-Slice-Form vom Typ (3.30) gibt, ist aus Satz 3.12

ersichtlich, dass es für diese Bit-Slice-Form genau eine T-Funktion gibt, deren Bit-Slices der Form entsprechen. Diese T-Funktionen haben die Eigenschaft, dass ihre  $i$ -te Bit-Slice nur von den  $i$ -ten Bits der Eingabewerte abhängig ist, also alle Bit-Slices gewissermaßen unabhängig voneinander operieren – eine Eigenschaft, die es uns erlaubt, die differentiellen Eigenschaften einer solchen Funktion aus den differentiellen Eigenschaften ihrer Bit-Slices zu bestimmen:

**Proposition 4.7.** *Sei  $f : \mathbb{F}_2^{k \times n} \rightarrow \mathbb{F}_2^{k \times n}$  eine T-Funktion in  $k$  Variablen, deren Bit-Slices algebraische Normalformen in den Variablen  $[x_j]_i$  sind:*

$$\begin{pmatrix} [x_0]_i \\ \vdots \\ [x_{k-1}]_i \end{pmatrix} \mapsto \begin{pmatrix} \bigoplus_{(i_0, \dots, i_{k-1})=(0, \dots, 0)}^{(1, \dots, 1)} a_{0(i_0, \dots, i_{k-1})} [x_0]_i^{i_0} \cdots [x_{k-1}]_i^{i_{k-1}} \\ \vdots \\ \bigoplus_{(i_0, \dots, i_{k-1})=(0, \dots, 0)}^{(1, \dots, 1)} a_{k-1(i_0, \dots, i_{k-1})} [x_0]_i^{i_0} \cdots [x_{k-1}]_i^{i_{k-1}} \end{pmatrix}. \quad (4.4)$$

Dann gilt für  $u, v \in \mathbb{F}_2^{k \times n}$ :

$$\begin{aligned} \#D_f(u, v) &= \prod_{i=0}^{n-1} \#D_{[f]_i}([u]_i, [v]_i) \\ &= \prod_{i=0}^{n-1} (\#\{x \in \mathbb{F}_2^k \mid [f(x \oplus [u]_i)]_i = [f(x)]_i \oplus [v]_i\}). \end{aligned} \quad (4.5)$$

*Beweis.* Aus der Definition der Differenzentabelle (Def. 2.23) folgt

$$\begin{aligned} \#D_f(u, v) &= \#\{x \in \mathbb{F}_2^{k \times n} \mid f(x \oplus u) = f(x) \oplus v\} \\ &\stackrel{(2)}{=} \#\left\{x \in \mathbb{F}_2^{k \times n} \mid \begin{array}{l} [f(x \oplus u)]_0 = [f(x)]_0 \oplus [v]_0 \\ \vdots \\ [f(x \oplus u)]_{n-1} = [f(x)]_{n-1} \oplus [v]_{n-1} \end{array}\right\} \\ &\stackrel{(3)}{=} \#\left\{x \in \mathbb{F}_2^{k \times n} \mid \begin{array}{l} [f([x]_0 \oplus [u]_0)]_0 = [f([x]_0)]_0 \oplus [v]_0 \\ \vdots \\ [f([x]_{n-1} \oplus [u]_{n-1})]_{n-1} = [f([x]_{n-1})]_{n-1} \oplus [v]_{n-1} \end{array}\right\} \\ &\stackrel{(4)}{=} \#\{[x]_0 \in \mathbb{F}_2^k \mid [f([x]_0 \oplus [u]_0)]_0 = [f([x]_0)]_0 \oplus [v]_0\} \cdot \dots \\ &\quad \cdot \#\{[x]_{n-1} \in \mathbb{F}_2^k \mid [f([x]_{n-1} \oplus [u]_{n-1})]_{n-1} = [f([x]_{n-1})]_{n-1} \oplus [v]_{n-1}\} \\ &\stackrel{(5)}{=} \prod_{i=0}^{n-1} (\#\{x \in \mathbb{F}_2^k \mid [f(x \oplus [u]_i)]_i = [f(x)]_i \oplus [v]_i\}) \\ &\stackrel{(6)}{=} \prod_{i=0}^{n-1} \#D_{[f]_i}([u]_i, [v]_i). \end{aligned}$$

Die Umformungen (2) – (6) verdienen einige Rechtfertigungen. In (2) wird die Gleichung auf  $n$  Gleichungen für die einzelnen Bit-Slices heruntergebrochen. Schritt (3) nutzt aus, dass  $f$  eine T-Funktion ist, deren  $i$ -te Bit-Slice  $[f(x)]_i$  nur von  $[x]_i$  abhängt. Da auf diese Weise  $n$  unabhängige Gleichungen für die  $n$  Spalten von  $x$  entstanden sind, entspricht die Anzahl der  $x \in \mathbb{F}_2^{k \times n}$ , die sämtliche Gleichungen erfüllen, genau dem Produkt der jeweiligen Anzahl der  $[x]_i \in \mathbb{F}_2^k$ , welche ihre jeweilige Spaltengleichung erfüllen; daraus folgt (4). Schritt (5) fasst dann lediglich alle Faktoren zusammen, und (6) folgt aus der Definition der Differenzentabelle. Damit ist (4.5) etabliert.  $\square$

Proposition 4.7 erlaubt es uns, Einträge der Differenzentabelle von  $f$  als Produkt von Einträgen der Differenzentabelle von  $[f]_i$  zu berechnen. Da jene nur  $2^{2k}$  statt  $2^{2kn}$  Einträge umfasst, ermöglicht dies eine effizientere und insbesondere erheblich speicherplatzsparendere Berechnung von  $\#D_f$  als der direkte Ansatz.

Bei der Betrachtung von  $\#D_{[f]_i}$  fällt weiterhin auf, dass die Einträge dieser Tabelle unabhängig von  $i$  sind: Die Bit-Slices  $[f(x)]_i$  und  $[f(x)]_j$  enthalten die gleichen algebraischen Normalformen, lediglich in anderen Variablen, folglich sind für  $i \geq 0$  alle  $\#D_{[f]_i}$  gleich. Insbesondere braucht also zur Berechnung von  $\#D_f(u, v)$  nur  $\#D_{[f]_0}$ , die Differenzentabelle der LSB, erstellt werden.

Die soeben erwähnte Eigenschaft von  $\#D_{[f]_i}$  ermöglicht es uns, die nichttrivialen Maximaleinträge der Differenzentabelle von  $f$  und damit auch das differentielle Potenzial aus der Differenzentabelle von  $\#D_{[f]_0}$  zu bestimmen:

**Proposition 4.8.** *Sei  $f : \mathbb{F}_2^{k \times n} \rightarrow \mathbb{F}_2^{k \times n}$  eine T-Funktion mit Bit-Slices der Form (4.4). Dann gilt*

$$\max_{\substack{u, v \in \mathbb{F}_2^{k \times n} \\ (u, v) \neq (0, 0)}} \#D_f(u, v) = (2^k)^{n-1} \cdot \max_{\substack{u, v \in \mathbb{F}_2^k \\ (u, v) \neq (0, 0)}} \#D_{[f]_i}(u, v), \quad (4.6)$$

und  $f$  hat das differentielle Potenzial

$$\Omega_f = \Omega_{[f]_0}.$$

*Beweis.* Nach Proposition 4.7 haben wir

$$\max_{\substack{u, v \in \mathbb{F}_2^{k \times n} \\ (u, v) \neq (0, 0)}} \#D_f(u, v) = \max_{\substack{u, v \in \mathbb{F}_2^{k \times n} \\ (u, v) \neq (0, 0)}} \left( \prod_{i=0}^{n-1} \#D_{[f]_i}([u]_i, [v]_i) \right),$$

mit  $0 \leq \#D_{[f]_i}([u]_i, [v]_i) \leq 2^k$  nach Definition der Differenzentabelle. Gibt es nun  $(a, b) \neq (0, 0)$  mit  $\#D_{[f]_i}(a, b) = 2^k$ , so ist der maximale Eintrag der Differenzentabelle von  $f$  gleich  $(2^k)^n$ , wie in (4.6) behauptet. Andernfalls müssen im Maximum wegen  $\#D_{[f]_i}(0, 0) = 2^k$  mindestens  $n - 1$  der  $n$  Faktoren gleich  $2^k$  sein, während der verbleibende Faktor dem nichttrivialen Maximaleintrag von  $\#D_{[f]_i}$  entsprechen muss, um insgesamt  $(u, v) \neq (0, 0)$  zu garantieren. Damit ist in beiden Fällen gezeigt:

$$\max_{\substack{u, v \in \mathbb{F}_2^{k \times n} \\ (u, v) \neq (0, 0)}} \left( \prod_{i=0}^{n-1} \#D_{[f]_i}([u]_i, [v]_i) \right) = (2^k)^{n-1} \cdot \max_{\substack{u, v \in \mathbb{F}_2^k \\ (u, v) \neq (0, 0)}} \#D_{[f]_i}(u, v);$$

Anzahl Funktionen	Differentielles Potenzial	Lineares Potenzial
10752	$\frac{1}{4}$	$\frac{1}{4}$
18816	$\frac{1}{2}$	1
10752	1	1

Tabelle 4.3: Differentielles und lineares Potenzial von T-Funktionen mit nichtlinearen Bit-Slices in 3 Variablen.

und da  $\#D_{[f]_i} = \#D_{[f]_0}$ , hat  $f$  das differentielle Potenzial

$$\begin{aligned}
\Omega_f &= \max_{\substack{u,v \in \mathbb{F}_2^{k \times n} \\ (u,v) \neq (0,0)}} \delta_f(u,v) \\
&= \frac{1}{2^{kn}} \cdot \max_{\substack{u,v \in \mathbb{F}_2^{k \times n} \\ (u,v) \neq (0,0)}} \#D_f(u,v) \\
&= \frac{1}{2^{kn}} \cdot (2^k)^{n-1} \cdot \max_{\substack{u,v \in \mathbb{F}_2^k \\ (u,v) \neq (0,0)}} \#D_{[f]_0}(u,v) \\
&= \frac{1}{2^k} \cdot \max_{\substack{u,v \in \mathbb{F}_2^k \\ (u,v) \neq (0,0)}} \#D_{[f]_0}(u,v) \\
&= \max_{\substack{u,v \in \mathbb{F}_2^k \\ (u,v) \neq (0,0)}} \delta_{[f]_0}(u,v) \\
&= \Omega_{[f]_0},
\end{aligned}$$

was zu zeigen war. □

Dieses Resultat besagt, dass das differentielle Potenzial einer T-Funktion der Form (4.4) bereits vollständig durch das differentielle Potenzial ihrer niederwertigsten Bit-Slice festgelegt ist, sich also mit zunehmender Wortgröße  $n$  nicht verändert.

Mit Hilfe von Proposition 4.8 können wir nun auch die für  $n = 2, 3, 4$  ermittelten experimentellen Ergebnisse für solche T-Funktionen in 3 Variablen (dargestellt in Tabelle 4.2 auf Seite 103) für beliebiges  $n$  bestätigen: Diejenigen Funktionen  $f_i$ , für die bei  $n = 2$  die Maximaleinträge der Differenzen- bzw. der linearen Approximationstabelle (16, 16) betragen, haben für alle  $n$  differentielles und lineares Potenzial von  $\frac{1}{4}$  usw. Tabelle 4.3 fasst alle möglichen Potenziale solcher T-Funktionen in 3 Variablen zusammen.

Damit sind bei der nichtlinearen Bit-Slice-Konstruktion bei 3 Variablen bestenfalls Funktionen mit  $\Omega_f = \Lambda_f = \frac{1}{4}$  möglich. Eine solche Funktion ist  $f_{12465}$ , deren Bit-Slices auf den algebraischen Normalformen der 12466. von Algorithmus 4.5 aufgezählten Permutation  $\sigma : \mathbb{F}_2^3 \rightarrow \mathbb{F}_2^3$  basieren:

$$\sigma = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 5 & 7 & 6 & 4 & 3 & 2 & 1 \end{pmatrix},$$

		$v$							
		000	001	010	011	100	101	110	111
$u$	000	8	0	0	0	0	0	0	0
	001	0	2	0	2	0	2	0	2
	010	0	0	2	2	0	0	2	2
	011	0	2	2	0	0	2	2	0
	100	0	0	0	0	2	2	2	2
	101	0	2	0	2	2	0	2	0
	110	0	0	2	2	2	2	0	0
	111	0	2	2	0	2	0	0	2

Tabelle 4.4: Differenzentabelle einer Bit-Slice von  $f_{12465}$ .

woraus nach dem Konstruktionsverfahren aus Abschnitt 3.3.3.5 die folgende Funktion entsteht:

$$f_{12465} : \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} x_0 \oplus x_1 \oplus (x_1 \wedge x_2) \\ x_1 \oplus (x_0 \wedge x_2) \\ x_0 \oplus x_1 \oplus (x_0 \wedge x_1) \oplus x_2 \end{pmatrix}.$$

Nach Proposition 4.8 ist der nichttriviale Maximaleintrag der Differenzentabelle von  $f_{12465}$  gegeben durch

$$(2^3)^{n-1} \cdot \max_{\substack{u,v \in \mathbb{F}_2^3 \\ (u,v) \neq (0,0)}} \#D_{[f_{12465}]_i}(u,v), \quad (4.7)$$

so dass zu dessen Bestimmung der nichttriviale Maximaleintrag der Differenzentabelle einer Bit-Slice von  $f_{12465}$  erforderlich ist. Jene errechnet sich für  $u, v \in \mathbb{F}_2^3$  zu

$$\begin{aligned} \#D_{[f_{12465}]_i}(u,v) &= \# \left\{ \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} \in \mathbb{F}_2^3 \mid \left[ f_{12465} \left( \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} \oplus \begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix} \right) \right]_i = \left[ f_{12465} \left( \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} \right) \right]_i \oplus \begin{pmatrix} v_0 \\ v_1 \\ v_2 \end{pmatrix} \right\} \\ &= \# \left\{ \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} \in \mathbb{F}_2^3 \mid \begin{array}{l} u_0 \oplus u_1 \oplus x_1 u_2 \oplus u_1 x_2 \oplus u_1 u_2 = v_0 \\ u_1 \oplus x_0 u_2 \oplus u_0 x_2 \oplus u_0 u_2 = v_1 \\ u_0 \oplus u_1 \oplus x_0 u_1 \oplus u_0 x_1 \oplus u_0 u_1 \oplus u_2 = v_2 \end{array} \right\}. \end{aligned}$$

Ihre Werte sind für alle  $u, v \in \mathbb{F}_2^3$  in Tabelle 4.4 dargestellt. Aus ihr ist ersichtlich, dass der maximale nichttriviale Eintrag 2 beträgt, also das differentielle Potenzial von  $f_{12465}$  nach Proposition 4.8 gleich dem von  $[f_{12465}]_0$  ist, nämlich  $\frac{2}{2^3} = \frac{1}{4}$ . Übereinstimmend dazu ergibt sich in (4.7) der Wert  $(2^3)^{n-1} \cdot 2 = 2^{3n-2}$  als maximaler nichttrivialer Eintrag der Differenzentabelle von  $f_{12465}$ .

Neben dieser Funktion gibt es noch 10751 weitere dieser Bauart mit bestmöglichen linearen und differentiellen Potenzialen. Zwei bei  $f_{12465}$  auffällige Merkmale:

- (a) Jede der drei Komponentenfunktionen enthält mindestens einen nichtlinearen Term;
- (b) Die nichtlinearen Terme kürzen sich bei der Bildung des XORs von Zeilen nicht weg;

eignen sich jedoch nicht zu einer Charakterisierung der „guten“ Funktionen: Zwar haben alle 10752 Funktionen mit  $\Omega_f = \Lambda_f = \frac{1}{4}$  ausschließlich nichtlineare Komponentenfunktionen, jedoch gibt es solche auch in den beiden weiteren in Tabelle 4.3 dargestellten Kategorien. Auch Eigenschaft (b) trifft zwar auf alle Funktionen der Kategorie  $(\frac{1}{4}, \frac{1}{4})$  zu, tritt jedoch auch in der Kategorie  $(\frac{1}{2}, 1)$  auf.

Auch bei den linearen Eigenschaften der Funktionen ist eine ähnliche Kategorienbildung wie bei den differentiellen Eigenschaften erkennbar. Unser Ziel ist es nun, für diese den Propositionen 4.7 und 4.8 vergleichbare Aussagen zu erhalten. Anstatt direkt über die lineare Approximationstabelle oder die Menge  $\#L_f$  vorzugehen, erinnern wir uns daran, dass das Linearitätsprofil die unnormierte Walsh-Transformierte der Differenzentabelle ist (Korollar 2.56). Für jene notieren wir als Hilfsresultat die folgende Produktformel:

**Lemma 4.9.** *Sei  $f : \mathbb{F}_2^{k \times n} \rightarrow \mathbb{F}_2^{k \times n}$  eine T-Funktion in  $k$  Variablen mit Bit-Slices der Form (4.4). Dann gilt für  $u, v \in \mathbb{F}_2^{k \times n}$ :*

$$\widehat{\#D}_f(u, v) = \prod_{i=0}^{n-1} \widehat{\#D}_{[f]_i}([u]_i, [v]_i). \quad (4.8)$$

*Beweis.* Es gilt

$$\begin{aligned} \widehat{\#D}_f(u, v) &\stackrel{(1)}{=} \sum_{x \in \mathbb{F}_2^{k \times n}} \sum_{y \in \mathbb{F}_2^{k \times n}} \#D_f(x, y) \cdot (-1)^{u \bullet x + v \bullet y} \\ &\stackrel{(2)}{=} \sum_{x \in \mathbb{F}_2^{k \times n}} \sum_{y \in \mathbb{F}_2^{k \times n}} \left( \prod_{i=0}^{n-1} \#D_{[f]_i}([x]_i, [y]_i) \right) \cdot (-1)^{u \bullet x + v \bullet y} \\ &\stackrel{(3)}{=} \sum_{\substack{[x]_{n-1} \\ \in \mathbb{F}_2^k}} \cdots \sum_{\substack{[x]_0 \\ \in \mathbb{F}_2^k}} \sum_{\substack{[y]_{n-1} \\ \in \mathbb{F}_2^k}} \cdots \sum_{\substack{[y]_0 \\ \in \mathbb{F}_2^k}} \left( \prod_{i=0}^{n-1} \#D_{[f]_i}([x]_i, [y]_i) \right) (-1)^{[u]_{n-1} \bullet [x]_{n-1} + \cdots + [v]_0 \bullet [y]_0} \\ &\stackrel{(4)}{=} \left( \sum_{[x]_{n-1} \in \mathbb{F}_2^k} \sum_{[y]_{n-1} \in \mathbb{F}_2^k} \#D_{[f]_{n-1}}([x]_{n-1}, [y]_{n-1}) (-1)^{[u]_{n-1} \bullet [x]_{n-1} + [v]_{n-1} \bullet [y]_{n-1}} \right) \\ &\quad \cdots \cdot \left( \sum_{[x]_0 \in \mathbb{F}_2^k} \sum_{[y]_0 \in \mathbb{F}_2^k} \#D_{[f]_0}([x]_0, [y]_0) (-1)^{[u]_0 \bullet [x]_0 + [v]_0 \bullet [y]_0} \right) \\ &\stackrel{(5)}{=} \prod_{i=0}^{n-1} \left( \sum_{[x]_i \in \mathbb{F}_2^k} \sum_{[y]_i \in \mathbb{F}_2^k} \#D_{[f]_i}([x]_i, [y]_i) \cdot (-1)^{[u]_i \bullet [x]_i + [v]_i \bullet [y]_i} \right) \\ &\stackrel{(6)}{=} \prod_{i=0}^{n-1} \widehat{\#D}_{[f]_i}([u]_i, [v]_i), \end{aligned}$$

was zu zeigen war. (1) entspricht der Definition der Walsh-Transformation, in (2) wird die Produktformel aus Proposition 4.7 angewandt. Schritt (3) zieht die Summationsbereiche und das Skalarprodukt bis auf die Ebene der einzelnen Bit-Slices auseinander. (4) resultiert aus einer Änderung der Summationsreihenfolge und der  $n$ -fachen Anwendung des Distributivgesetzes. Die auf diese Weise geeignet gruppierten Faktoren werden in Schritt (5) in einer Produktbildung zusammengefasst; (6) folgt wieder aus der Definition der Walsh-Transformation.  $\square$

Mit Hilfe dieses Lemmas folgt für die abgeleiteten Größen  $|\text{LAT}_f|$  sowie  $\lambda_f$

**Proposition 4.10.** *Sei  $f : \mathbb{F}_2^{k \times n} \rightarrow \mathbb{F}_2^{k \times n}$  eine  $T$ -Funktion mit Bit-Slices der Form (4.4). Dann gilt für  $u, v \in \mathbb{F}_2^{k \times n}$*

$$|\text{LAT}_f(u, v)| = \prod_{i=0}^{n-1} |\text{LAT}_{[f]_i}([u]_i, [v]_i)| \quad (4.9)$$

sowie

$$\lambda_f(u, v) = \prod_{i=0}^{n-1} \lambda_{[f]_i}([u]_i, [v]_i). \quad (4.10)$$

*Beweis.* Mit Korollar 2.56 sowie Gleichung (2.29) gilt  $|\text{LAT}_f| = \frac{1}{2} \sqrt{\widehat{\#D}_f}$ , also haben wir

$$\begin{aligned} |\text{LAT}_f(u, v)| &= \frac{1}{2} \sqrt{\widehat{\#D}_f(u, v)} \\ &= \frac{1}{2} \sqrt{\prod_{i=0}^{n-1} \widehat{\#D}_{[f]_i}([u]_i, [v]_i)} && \text{(nach Lemma 4.9)} \\ &= \frac{1}{2} \prod_{i=0}^{n-1} \sqrt{\widehat{\#D}_{[f]_i}([u]_i, [v]_i)} \\ &= \frac{1}{2} \prod_{i=0}^{n-1} 2 \cdot |\text{LAT}_{[f]_i}([u]_i, [v]_i)| \\ &= \prod_{i=0}^{n-1} |\text{LAT}_{[f]_i}([u]_i, [v]_i)|, \end{aligned}$$

beziehungsweise

$$\begin{aligned}
\lambda_f(u, v) &= \frac{1}{2^{2kn}} \widehat{\#D}_f(u, v) \\
&= \frac{1}{2^{2kn}} \prod_{i=0}^{n-1} \widehat{\#D}_{[f]_i}([u]_i, [v]_i) && \text{(Lemma 4.9)} \\
&= \prod_{i=0}^{n-1} \frac{1}{2^{2k}} \widehat{\#D}_{[f]_i}([u]_i, [v]_i) \\
&= \prod_{i=0}^{n-1} \lambda_{[f]_i}([u]_i, [v]_i),
\end{aligned}$$

wie behauptet. □

Damit sind also auch der Betrag der linearen Approximationstabelle sowie das Linearitätsprofil von  $f$  auf eine einzelne Bit-Slice zurückführbar. Insgesamt können wir nun das lineare Potenzial einer solchen Funktion wie folgt charakterisieren:

**Proposition 4.11.** *Sei  $f : \mathbb{F}_2^{k \times n} \rightarrow \mathbb{F}_2^{k \times n}$  eine T-Funktion mit Bit-Slices der Form (4.4). Dann hat  $f$  das lineare Potenzial*

$$\Lambda_f = \Lambda_{[f]_0}.$$

*Beweis.* Nach Lemma 4.9 gilt

$$\begin{aligned}
\Lambda_f &= \max_{\substack{u, v \in \mathbb{F}_2^{k \times n} \\ (u, v) \neq (0, 0)}} \lambda_f(u, v) \\
&= \max_{\substack{u, v \in \mathbb{F}_2^{k \times n} \\ (u, v) \neq (0, 0)}} \prod_{i=0}^{n-1} \lambda_{[f]_i}([u]_i, [v]_i),
\end{aligned}$$

wobei wir an Stelle von  $[f]_i$  wegen der identischen Struktur aller Bit-Slices ebenso  $[f]_0$  betrachten können, und stets  $0 \leq \lambda_{[f]_0}([u]_i, [v]_i) \leq 1$  gilt. Gibt es nun ein  $(a, b) \neq (0, 0)$  mit  $\lambda_{[f]_0}(a, b) = 1$ , so ist das lineare Potenzial von  $f$  gleich 1, wie behauptet. Andernfalls müssen im Maximum wegen  $\lambda_{[f]_0}(0, 0) = 1$  mindestens  $n - 1$  der  $n$  Faktoren gleich 1 sein, und der letzte Faktor muss dem maximalen nichttrivialen Eintrag in  $\lambda_{[f]_0}$  entsprechen, damit insgesamt  $(u, v) \neq (0, 0)$  gilt. Damit entspricht das obige Produkt in beiden Fällen dem nichttrivialen Maximaleintrag von  $\lambda_{[f]_0}$ , also

$$\Lambda_f = 1 \cdot \max_{\substack{u, v \in \mathbb{F}_2^k \\ (u, v) \neq (0, 0)}} \lambda_{[f]_0}(u, v) = \Lambda_{[f]_0},$$

was zu zeigen war. □

Wie das differentielle ist also auch das lineare Potenzial einer T-Funktion der Form (4.4) unabhängig von der Wortbreite  $n$  bereits vollständig durch die niederwertigste Bit-Slice

festgelegt. Hingegen beeinflusst in beiden Fällen die Anzahl der Variablen  $k$  die bestmöglichen Werte von  $\Omega_{[f]_0}$  und  $\Lambda_{[f]_0}$ , die ja denjenigen von  $\Omega_f$  und  $\Lambda_f$  entsprechen. Allgemein ergibt diese Beobachtung die folgenden unteren Schranken für die differentiellen und linearen Potenziale von Funktionen dieser Form:

**Proposition 4.12.** *Sei  $f : \mathbb{F}_2^{k \times n} \rightarrow \mathbb{F}_2^{k \times n}$  eine  $T$ -Funktion mit Bit-Slices der Form (4.4). Dann gilt für das differentielle beziehungsweise lineare Potenzial von  $f$ :*

$$\Omega_f \geq \frac{1}{2^{k-1}} \quad (4.11)$$

und

$$\Lambda_f \geq \frac{1}{2^{k-1}}. \quad (4.12)$$

*Beweis.* Zum Nachweis von (4.11): Da Einträge einer Differenzentabelle (siehe Satz 2.51 (a)) stets gerade sind, ist der kleinstmögliche von Null verschiedene Eintrag der Differenzentabelle einer beliebigen Funktion  $\phi : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^m$  gleich 2. Dass Einträge ungleich Null stets existieren, garantiert uns die Tatsache, dass die Zeilensummen stets  $2^m$  betragen müssen (Satz 2.51 (d)). Also gilt für beliebiges  $f$  der betrachteten Form

$$\begin{aligned} \Omega_f \stackrel{(\text{Prop. 4.8})}{=} \Omega_{[f]_0} &\geq \min_{g: \mathbb{F}_2^k \rightarrow \mathbb{F}_2^k} \max_{\substack{u, v \in \mathbb{F}_2^k \\ (u, v) \neq (0, 0)}} \frac{1}{2^k} \cdot \#D_{[g]_0}(u, v) \\ &= \frac{1}{2^k} \cdot 2 = \frac{1}{2^{k-1}}, \end{aligned}$$

wie behauptet.

(4.12) folgt mit Proposition 4.11 direkt aus der Schranke von Chabaud und Vaudenay (vgl. [6], Theorem 4): In unserer Schreibweise besagt diese, dass für eine Funktion  $F : \mathbb{F}_2^p \rightarrow \mathbb{F}_2^q$

$$\max_{\substack{u \in \mathbb{F}_2^p, v \in \mathbb{F}_2^q \\ (u, v) \neq (0, 0)}} |\text{LAT}_F(u, v)| \geq \frac{1}{2} \left( 3 \cdot 2^p - 2 - 2 \frac{(2^p - 1)(2^{p-1} - 1)}{2^q - 1} \right)^{1/2}$$

gilt, was für  $p = q$  gerade

$$\max_{\substack{u, v \in \mathbb{F}_2^p \\ (u, v) \neq (0, 0)}} |\text{LAT}_F(u, v)| \geq \frac{1}{2} \cdot 2^{(p+1)/2}$$

bedeutet. Im Falle der Funktion  $[f]_0$  ist  $p = q = k$ , und mit  $\lambda_{[f]_0} = \left(2 \cdot \frac{1}{2^k} |\text{LAT}_{[f]_0}|\right)^2$  (nach Korollar 2.48 und Gleichung (2.29)) ergibt sich umgerechnet auf das lineare Potenzial:

$$\begin{aligned} \Lambda_f = \Lambda_{[f]_0} &\geq \left(2 \cdot \frac{1}{2^k} \cdot \frac{1}{2} \cdot 2^{(k+1)/2}\right)^2 \\ &= 2^{k+1-2k} = \frac{1}{2^{k-1}}, \end{aligned}$$

was zu zeigen war. □

$k$	Untere Schranke		Gefunden		Untersuchte Anzahl
	$\Omega_f$	$\Lambda_f$	$\Omega_f$	$\Lambda_f$	
1	1	1	1	1	(alle)
2	$\frac{1}{2}$	$\frac{1}{2}$	1	1	(alle)
3	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	40.320 (alle)
4	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{4}$	500.000
8	$\frac{1}{128}$	$\frac{1}{128}$	$\frac{5}{128}$ $\frac{1}{32} = \frac{4}{128}$	$\frac{225}{4096} \approx \frac{7}{128}$ $\frac{1}{16} = \frac{8}{128}$	3.015.000

Tabelle 4.5: Untere Schranken für differentielles und lineares Potenzial einer T-Funktion mit nichtlinearen Bit-Slices in  $k$  Variablen.

Für eine Einschätzung dieser theoretischen unteren Schranken ist nun interessant, ob sich tatsächlich T-Funktionen der in diesem Abschnitt betrachteten Bauart mit  $k$  Variablen angeben lassen, für die diese Schranken scharf sind. Für  $k = 1$  etwa sind sie es: Nach Beispiel 3.24 ergibt das Konstruktionsverfahren für  $k \leq 2$  nur affine Bit-Slices, für die  $\Omega_f = \Lambda_f = 1$  gilt. Dies stimmt für  $k = 1$ , nicht jedoch für  $k = 2$  mit den unteren Schranken  $\frac{1}{2^{1-1}} = 1$  respektive  $\frac{1}{2^{2-1}} = \frac{1}{2}$  überein. Bei drei Variablen haben wir bereits gesehen, dass es Funktionen mit differentiellem und linearem Potenzial von  $\frac{1}{2^{3-1}} = \frac{1}{4}$  gibt (sogar 10752 viele, siehe Tabelle 4.3 auf Seite 106). Bis hierher sind die Schranken also für  $k = 1, 3$  scharf, für  $k = 2$  jedoch nicht.

Für  $k \geq 4$  wird der Suchraum aller möglichen Permutationen von  $\mathbb{F}_2^k$  zu groß:  $2^4! \approx 2^{44}$ ,  $2^5! \approx 2^{117}$ , und für 8 Variablen gibt es ungefähr  $2^{1684}$  mögliche Bit-Slice-Formen. Eine experimentelle Untersuchung des differentiellen und linearen Potenzials einer zufälligen Auswahl solcher Funktionen in 4 und 8 Variablen brachte keine Funktion hervor, deren Potenziale den unteren Schranken entsprechen. Eine Übersicht zu den für ihre Variablenzahl besten tatsächlich gefundenen Funktionen gibt Tabelle 4.5. Für  $k = 3$  wurde bereits die bestmögliche Funktion  $f_{12465}$  angegeben. Tabelle 4.6 auf der nächsten Seite gibt für  $k = 4, 8$  jeweils Beispiele für Permutationen an, welche die differentiellen und linearen Potenziale der aus ihnen konstruierten T-Funktionen die jeweils besten experimentell ermittelten Werte annehmen lassen.

Weitere Variationsmöglichkeiten bei der Erzeugung von invertierbaren T-Funktionen mit nichtlinearen Bit-Slices ergeben sich durch die Einführung von Parametern. Experimentell stellt sich zumindest für  $k = 3$  und  $n = 2, 3, 4$  heraus, dass additive Parameter (wie in (3.31)) an den differentiellen und linearen Potenzialen der Funktionen nichts ändern. Darunter ist sowohl das explizite additive Hinzufügen eines echten Parameters  $\alpha(x)$ , etwa  $(x_0 + x_2)^2 \vee 3$ , als auch die Verwendung von Addition oder Subtraktion statt XOR in den ANF-Termen erfasst. Im allgemeinen Fall können jedoch Parameter sehr wohl die Nichtlinearitätseigenschaften von Funktionen beeinflussen. So ergeben sich die

$k$	Permutation des $\mathbb{F}_2^k$	$\Omega_f$	$\Lambda_f$
4	$\pi_4 = [d, c, 4, a, 3, 8, b, 6, f, 0, 9, 1, 5, 2, e, 7]$	$\frac{1}{4}$	$\frac{1}{4}$
8	$\pi_8 =$ [86, c4, f6, 71, 8d, fb, b2, 5b, 12, ef, a8, 5f, 36, 68, 32, ec, 17, cd, 96, be, 46, 72, 0c, 5d, 49, cf, 2a, 27, 84, 11, 39, 16, 58, 33, d4, 13, df, b4, 1e, 1c, 62, b1, 3b, ac, 85, 20, 87, 43, bc, d2, 51, bd, 94, 55, a5, 35, 2c, 1a, c3, c5, 29, 4d, 81, 4e, eb, 76, e8, f1, a4, 26, 37, b5, e9, 4a, 90, 79, 6a, 53, 8e, 04, 42, 2b, 3a, 9f, d5, bb, aa, 00, ab, 83, 41, 6d, ad, 56, d3, 65, 9b, a6, 06, 1f, 7b, ca, e7, c1, 15, 24, 5e, 6e, dc, 5a, 7a, 98, 10, 48, 5c, 21, b0, 2e, 9d, 2f, 30, 69, ee, 38, 97, 99, db, 22, de, 3f, b3, 8a, f0, d6, e0, 34, 05, 77, ba, ce, cc, 4f, f7, ae, 08, e3, c7, a7, fd, bf, 23, 92, 80, 7e, 95, 78, fc, 91, 52, 19, 1b, fe, 8f, 4c, a3, 9e, 89, 93, 3c, 60, dd, a2, 9c, 03, 57, b9, 0a, 01, 64, 8c, f9, 75, 50, 88, f8, fa, d0, 45, d8, 7c, 4b, d9, 6f, 3d, 3e, 47, 25, ff, 2d, e4, 0b, 73, e6, 0e, f2, 28, af, 66, 02, c8, f3, 7d, e1, 82, 59, a0, 07, 6c, c9, c0, 8b, 61, da, 54, d7, f4, 14, 9a, a9, 7f, 67, 0d, d1, b8, f5, 09, a1, ea, b7, 74, e2, 31, 0f, c2, 1d, 6b, cb, 40, 70, c6, 44, ed, b6, e5, 63, 18]	$\frac{5}{128}$	$\frac{225}{4096}$
8	$\sigma_8 =$ [af, b4, 2c, b6, 17, 4f, 31, e0, a0, 89, 99, eb, 49, 52, 1b, 62, 57, c0, a3, 1f, 30, 0b, a4, 7d, 97, 21, a1, a6, 66, 9d, 6e, 80, 76, 5e, 93, 94, e1, 00, 0f, c9, a9, 91, 13, 8c, e4, ea, 9b, f5, 77, 4e, 20, 9f, b3, ef, f4, 3c, fa, 86, 71, 18, 2b, 26, c8, 98, e2, b7, 6b, 79, d1, c5, b1, 37, 38, 7c, bd, 09, aa, d0, 03, ce, 72, 5a, a7, e3, b5, ba, 6d, 23, 40, 08, 3a, 73, 85, 05, 78, 4a, f6, c3, be, 8d, d7, bc, e5, 92, 54, e8, ab, de, c1, 90, e6, 5c, e7, 81, ae, cd, 48, 58, 51, 1e, 29, db, 32, 67, 8e, 6c, 1d, dd, 5d, d9, 7e, 25, 34, 4b, 1a, 3f, 61, 47, 27, 02, c6, 15, a2, ff, 60, 9a, 22, f2, dc, 0e, 53, 5b, 87, 9c, 9e, 74, b9, 16, bf, 1c, d6, fc, ec, 7f, c7, bb, 56, 10, f9, 46, 0c, 28, b8, ed, cc, ee, d8, 33, 8f, 59, ad, df, 3e, 75, c2, 83, 50, fb, a8, 8b, 0d, ca, fe, 63, f3, 7b, 0a, 55, 64, 19, d4, 36, 41, 6a, 45, f1, 88, 6f, da, a5, e9, 96, 84, 4c, 11, 3d, 44, 04, 39, 70, fd, 2a, cb, f8, 7a, 07, 12, 06, 2f, d2, 95, b0, c4, cf, 24, f7, ac, 4d, 35, 42, 82, 5f, d3, 3b, 14, 69, d5, 8a, b2, 68, 43, 65, f0, 2e, 01, 2d]	$\frac{1}{32}$	$\frac{1}{16}$

Tabelle 4.6: Beispiele für beste experimentell ermittelte Permutationen bei  $k = 4, 8$ . Die Permutationen sind jeweils durch ihre hexadezimale Wertetabelle dargestellt.

$n$	$\Omega_f$	$n$	$\Omega_f$
1	1	7	$\frac{9}{16}$
2	1	8	$\frac{9}{16}$
3	$\frac{3}{4}$	9	$\frac{17}{32}$
4	$\frac{3}{4}$	10	$\frac{17}{32}$
5	$\frac{5}{8}$	11	$\frac{33}{64}$
6	$\frac{5}{8}$	12	$\frac{33}{64}$

Tabelle 4.7: Differentielle Potenziale von  $x \wedge (x - 1)^2$  für verschiedene  $n$ .

differentiellen Potenziale der (allerdings nicht invertierbaren) Funktion

$$f(x) = x \wedge (x - 1)^2$$

für verschiedene  $n$  wie in Tabelle 4.7 angegeben. Man sieht daran insbesondere, dass Funktionen mit nichtlinearen Bit-Slices durchaus mit  $n$  variierende differentielle Potenziale besitzen können. Dass die differentiellen und linearen Eigenschaften der Funktionen vom Typ (4.4) lediglich von denen einer ihrer Bit-Slices abhängen, ist demnach ein sehr spezielles Charakteristikum und keinesfalls allgemein übertragbar.

#### 4.1.5 Andere Konstruktionsprinzipien

Für den Entwurf von S-Boxen existieren im Wesentlichen zwei Ansätze: Entweder es werden zufällig generierte Abbildungen auf ihre Nichtlinearitätseigenschaften untersucht, bis ein vorab bestimmtes Qualitätskriterium erfüllt ist, oder es werden algebraische Konstruktionen verwendet, deren differentielle und lineare Potenziale sich durch mathematische Analyse beschränken lassen.

Nach der Methode der zufallsgesteuerten Suche erzeugte S-Boxen haben den Vorteil, dass die Existenz kryptoanalytisch ausnutzbarer algebraischer Struktur weniger wahrscheinlich ist. Eine solche Suche ist jedoch nur für kleine Dimensionen möglich, da der Aufwand zur Bestimmung der Potenziale exponentiell mit der Dimension wächst. Auf diese Weise erzeugte S-Boxen haben zumal in aller Regel keine optimalen Nichtlinearitätseigenschaften: Eine empirische Studie von Rijmen [31] für 8-Bit-S-Boxen brachte als beste Exemplare solche mit  $(\Omega_f, \Lambda_f) = (\frac{4}{128}, \frac{8}{128})$  bzw.  $(\frac{5}{128}, \frac{15}{256})$  hervor, was sich interessanterweise recht genau mit unseren Ergebnissen deckt, siehe Tabelle 4.5.

Explizite algebraische Konstruktionen haben den großen Vorteil, dass sich mit ihnen sehr niedrige Potenziale erhalten lassen, die insbesondere von der Dimension abhängig sind (was bei allen betrachteten T-Funktionen nicht der Fall war). Zumeist basieren sie auf Potenzierung oder Exponentiation in endlichen Körpern. Ein Beispiel dafür ist etwa die Erhebung in die dritte Potenz  $x \mapsto x^3$  in  $GF(2^n)$ : Für ungerades  $n$  ist dies eine Bijektion und es gilt  $\Omega_f = \Lambda_f = \frac{1}{2^{n-1}}$  [30]. Eine weitere Konstruktionsmethode

ist die Inversion  $x \mapsto x^{-1}$  in  $GF(2^n)$ , auf der die AES-S-Boxen basieren [9]; für sie gilt  $\Omega_f = \Lambda_f = \frac{1}{2^{n-2}}$  (jeweils für gerades  $n \geq 2$ , was für effiziente Implementierungen interessant ist). Die algebraische Struktur dieser S-Boxen lässt sich jedoch in auf Polynominterpolation basierenden Angriffen ausnutzen [20], was etwa die AES-Autoren dazu bewogen hat, ihre S-Boxen mit einer affinen Abbildung „nachzubehandeln“, um den Grad des Interpolationspolynoms zu erhöhen.

## 4.2 T-Funktionen als Diffusionselemente

Gute Diffusionsabbildungen bewirken, dass jedes Ausgabebit von möglichst vielen Eingabebits abhängt. Werden solche Abbildungen in der Diffusionsebene einer Blockchiffre eingesetzt, so erschwert dies differentielle und lineare Angriffe wesentlich, da es dem Angreifer nicht möglich ist, viele Runden umspannende differentielle Charakteristiken bzw. lineare Approximationen aufzustellen, in denen nur wenige nichtlineare Komponenten berücksichtigt werden müssen.

**Definition 4.13** (Aktive Komponente, nach [8] Kapitel 5.6). In einer Blockchiffre heißt eine Komponente (**linear**) **aktiv** in einer linearen Approximation, wenn in dieser der Auswahlvektor ihrer Ausgabebits nicht Null ist; sie heißt (**differentiell**) **aktiv** in einer differentiellen Charakteristik, wenn ihre Eingabedifferenz in dieser von Null verschieden ist.  $\square$

Da in vielen Chiffren alleine die S-Boxen nichtlinear sind, kann in der Definition „Komponente“ häufig mit „S-Box“ gleichgesetzt werden. Zusammenfassend sprechen wir auch einfach von **aktiven S-Boxen**. Die Anzahl der aktiven Komponenten beeinflusst die Komplexität differentieller und linearer Angriffe erheblich; wird für die Diffusion von  $m$  S-Boxen eine MDS-Abbildung gewählt, so sind in je zwei aufeinanderfolgenden Runden  $m + 1$  S-Boxen aktiv: Sind es  $1 \leq k \leq m$  in der ersten, so sind es nach Definition 2.19 mindestens  $m + 1 - k$  in der zweiten. Da  $m + 1$  eine obere Schranke ist, bewirken MDS-Abbildungen optimale Diffusion. In den folgenden Abschnitten soll besprochen werden, wie man T-Funktionen konstruieren kann, welche MDS oder beinahe MDS sind.

### 4.2.1 MDS-Abbildungen mittels T-Funktionen

Der in Abschnitt 2.2.3 eingeführte Begriff der *Maximum Distance Separability* lässt sich unmittelbar von Bitvektoren auf Vektoren von Bitvektoren übertragen: Für  $x = (x_0, \dots, x_{m-1})^T, y = (y_0, \dots, y_{m-1})^T \in (\mathbb{F}_2^n)^m$  ist  $d_h(x, y) := \#\{i \mid x_i \neq y_i\}$ , und Definition 2.19 gilt nun analog für auf  $m$ -Tupel von  $n$ -Bit-Worten operierende Funktionen, d.h. eine Funktion  $f$  ist MDS- $(m + 1)$  oder kurz MDS, wenn  $\text{Dist}_f = m + 1$ .

Zunächst stellen wir fest, dass MDS-Abbildungen zwingend invertierbar sein müssen:

**Proposition 4.14.** *Ist die Funktion  $\phi : (\mathbb{F}_2^n)^m \rightarrow (\mathbb{F}_2^n)^m$  nicht invertierbar, so gilt*

$$\text{Dist}_\phi \leq m.$$

*Beweis.* Angenommen,  $\phi$  sei nicht invertierbar, dann existieren zwei Vektoren  $a, b \in (\mathbb{F}_2^n)^m$  mit  $a \neq b$  und  $\phi(a) = \phi(b)$ . Also haben wir

$$\begin{aligned} \text{Dist}_\phi &= \min_{\substack{x, x' \in (\mathbb{F}_2^n)^m \\ x \neq x'}} d_h(x, x') + d_h(\phi(x), \phi(x')) && \text{(nach Def. 2.19)} \\ &\leq d_h(a, b) + \underbrace{d_h(\phi(a), \phi(b))}_{=0} \\ &\leq m, \end{aligned}$$

da  $a$  und  $b$  aus  $m$  Komponenten bestehen und somit  $d_h(a, b) \leq m$  gilt.  $\square$

**Korollar 4.15.** *Sei  $\phi : (\mathbb{F}_2^n)^m \rightarrow (\mathbb{F}_2^n)^m$  eine MDS-Abbildung. Dann ist  $\phi$  invertierbar.  $\square$*

Für den Trivialfall  $m = 1$  lässt sich unmittelbar eine T-Funktion mit MDS-Eigenschaft angeben, nämlich die Identität auf  $\mathbb{F}_2^n$ :  $\text{id}(x) = x$ . Für  $x \neq x' \in \mathbb{F}_2^n$  gilt  $\text{id}(x) \neq \text{id}(x')$  und somit  $\text{Dist}_{\text{id}} = 1 + 1 = 2 = m + 1$ . Jedoch zeigen Klimov und Shamir in [18], dass es für  $m > 1$  keine MDS-T-Funktion in  $m$  Variablen à einem  $n$ -Bit-Wort geben kann. Wie wir sehen werden, sind MDS-T-Funktionen jedoch möglich, wenn  $m$  Komponenten mit je  $l > 1$  Variablen verwendet werden.

Sollen T-Funktionen mit einer bestimmten Eigenschaft (Invertierbarkeit oder hier MDS) konstruiert werden, so empfiehlt sich die Vorgehensweise, zunächst eine Bit-Slice-Form mit der gewünschten Eigenschaft zu konstruieren und diese dann zu einer Funktion auf  $n$ -Bit-Worten auszubauen (siehe [14] Kapitel 10). Diese Methodik setzt voraus, dass sich die gewünschte Eigenschaft von der Bit-Slice auf die gesamte Funktion vererbt. Bei der Invertierbarkeit war dieser Zusammenhang nach Satz 3.17 garantiert, für MDS gilt vergleichbarerweise

**Proposition 4.16** (nach [18]). *Sei  $\phi : (\mathbb{F}_2^{ln})^m \rightarrow (\mathbb{F}_2^{ln})^m$  eine T-Funktion in  $m$  Komponenten à  $l$  Variablen, wobei  $l > 1$ . Sind alle Bit-Slices von  $\phi$  MDS-Abbildungen auf  $(\mathbb{F}_2^l)^m$ , so ist  $\phi$  eine MDS-Abbildung auf  $(\mathbb{F}_2^{ln})^m$ .*

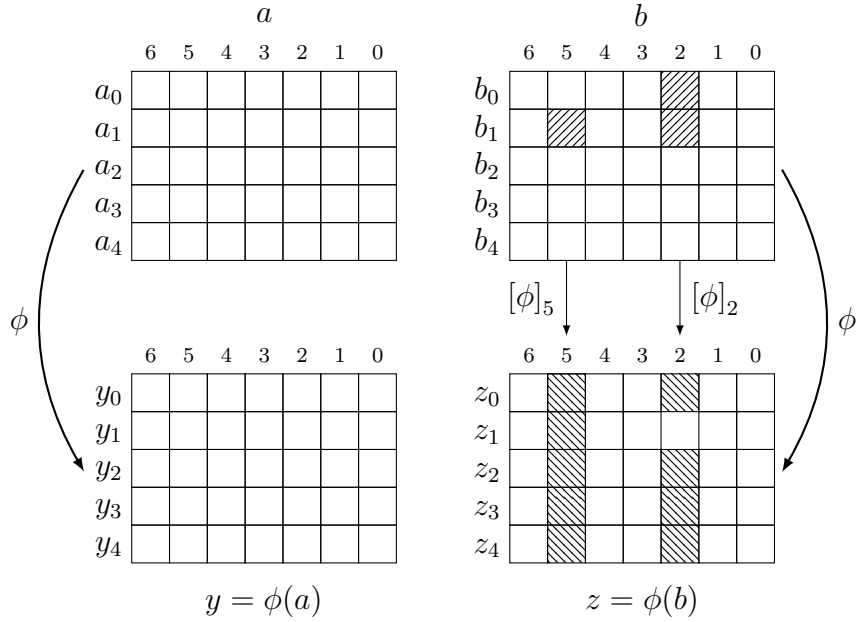
*Beweis.* Seien  $a = (a_0, \dots, a_{m-1})^T, b = (b_0, \dots, b_{m-1})^T \in (\mathbb{F}_2^{ln})^m$  beliebig, wobei  $a \neq b$ ; und sei  $\delta := d_h(a, b)$ , d.h.  $a$  und  $b$  unterscheiden sich in  $1 \leq \delta \leq m$  Komponenten. In jeder Bit-Slice sind folglich bis zu  $\delta$   $l$ -Bit-Werte verschieden, d.h.  $d_h([a]_i, [b]_i) \leq \delta$  für alle  $0 \leq i < n$ . Wegen  $a \neq b$  ist zumindest eine Bit-Slice von  $a$  und  $b$  verschieden, o.B.d.A. sei dies Bit-Slice Nummer  $j$ . Nach Voraussetzung sind alle Bit-Slice-Abbildungen  $[\phi(x)]_i$  MDS, folglich gilt  $d_h([a]_j, [b]_j) + d_h([\phi(a)]_j, [\phi(b)]_j) \geq m + 1$  und damit

$$d_h([\phi(a)]_j, [\phi(b)]_j) \geq m + 1 - \delta,$$

was insgesamt  $d_h(\phi(a), \phi(b)) \geq m + 1 - \delta$  bedeutet, da sich  $\phi(a)$  und  $\phi(b)$  in mindestens  $m + 1 - \delta$  Komponenten in ein- und derselben Bit-Slice unterscheiden. Folglich haben wir

$$\begin{aligned} d_h(a, b) + d_h(\phi(a), \phi(b)) &\geq \delta + (m + 1 - \delta) \\ &= m + 1, \end{aligned}$$

also ist  $\phi$  MDS, was zu zeigen war.  $\square$



Jede Zeile steht für eine Komponente à  $l$  Variablen, die schraffierten Quadrate signalisieren differierende Werte zwischen  $a$  und  $b$  bzw. zwischen  $\phi(a)$  und  $\phi(b)$ . In der Abbildung ist  $m = 5$  und  $\delta = 2$ . In den Bit-Slices 6, 4, 3, 1, 0 ist die Differenz von  $a$  und  $b$  0, in den Bit-Slices 5 und 2 ist sie  $\leq 2$ . Da alle Bit-Slices MDS sind, müssen in Bit-Slice 5 von  $\phi(a)$  und  $\phi(b)$  mindestens  $m + 1 - 1 = 5$  und in Bit-Slice 2 mindestens  $m + 1 - 2 = 4$  Komponenten verschieden sein. Damit  $\phi$  insgesamt MDS ist, müssen sich  $\phi(a)$  und  $\phi(b)$  in mindestens  $m + 1 - \delta = 4$  Komponenten unterscheiden (hier sind es sogar 5, der „Grenzfall“ 4 wird erreicht, wenn alle  $\delta$  Unterschiede genau eine Bit-Slice betreffen).

Abbildung 4.3: Illustration der Aussage von Proposition 4.16.

Die Aussage von Proposition 4.16 ist in Abbildung 4.3 illustriert. Es ist zu beachten, dass solche T-Funktionen (und ihre Bit-Slices)  $lm$  logische Variablen enthalten, jedoch MDS- $(m + 1)$  sind.

Mit Hilfe dieser Proposition reduziert sich die Suche nach MDS-T-Funktionen auf die Suche nach MDS-Bit-Slice-Abbildungen. Sollen in der Diffusionsebene einer Blockchiffre  $m$  S-Boxen mit einer MDS-T-Funktion vermischt werden, so benötigen wir eine Bit-Slice-Form, die MDS- $(m + 1)$  ist.

In [18] schlagen Klimov und Shamir eine Konstruktionsmethode für MDS-Bit-Slices vor, die auf einer linearen MDS-Abbildung über dem endlichen Körper  $GF(2^l)$  basiert. Für  $m \in \mathbb{N}$  und  $a_0, \dots, a_{m-1} \in GF(2^l)$  sei

$$\mathcal{A}(a_0, \dots, a_{m-1}) := \begin{pmatrix} 1 & a_0 & a_0^2 & \cdots & a_0^{m-1} \\ 1 & a_1 & a_1^2 & \cdots & a_1^{m-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_{m-1} & a_{m-1}^2 & \cdots & a_{m-1}^{m-1} \end{pmatrix}, \quad (4.13)$$

die Zeilen der dadurch beschriebenen  $m \times m$ -Matrix  $\mathcal{A}$  bestehen also aus den 0-ten bis  $m - 1$ -ten Potenzen der  $a_i$ .  $\mathcal{A}$  wird auch als *Vandermonde-Matrix* bezeichnet, und ihre Determinante ist

$$\det \mathcal{A} = \prod_{0 \leq i < j \leq m-1} (a_j - a_i)$$

(vgl. [10], Abschnitt 3.2.7). Daraus ist ersichtlich, dass  $\mathcal{A}$  genau dann regulär ist, wenn alle  $a_i$  verschieden sind. In diesem Fall existiert eine inverse Matrix  $\mathcal{A}^{-1}$ ; und für  $2m$  verschiedene Elemente  $a_0, \dots, a_{m-1}, a_m, \dots, a_{2m-1} \in GF(2^l)$  können wir die folgende lineare Abbildung definieren:

$$\psi : GF(2^l)^m \rightarrow GF(2^l)^m$$

$$\begin{pmatrix} x_0 \\ \vdots \\ x_{m-1} \end{pmatrix} \mapsto \mathcal{A}(a_0, \dots, a_{m-1}) \mathcal{A}^{-1}(a_m, \dots, a_{2m-1}) \begin{pmatrix} x_0 \\ \vdots \\ x_{m-1} \end{pmatrix}. \quad (4.14)$$

In [18] wird gezeigt, dass  $\psi$  eine MDS-Abbildung ist, also  $\text{Dist}_\psi = m + 1$  gilt.

Um aus (4.14) eine Bit-Slice-Form zu bilden, muss die bei der Matrix-Vektor-Multiplikation verwendete Arithmetik in  $GF(2^l)$  auf Arithmetik modulo 2 abgebildet werden. Dazu wird jede Variable  $x_i \in GF(2^l)$  als Bitvektor  $(x_{i,a}, x_{i,b}, \dots) \in \mathbb{F}_2^l$  gedeutet. Da die Elemente von  $GF(2^l)$  Polynome vom Grad kleiner  $l$  mit Koeffizienten aus  $\mathbb{F}_2$  sind (vgl. Definition 2.11), bedeutet Addition in  $GF(2^l)$  gerade Addition zweier Polynome  $p, q$  mit anschließender Reduktion der Koeffizienten modulo 2, in Hexadezimaldarstellung also das XOR von  $p$  und  $q$ . Dies überträgt sich direkt auf die Bitvektordarstellung, in der dann komponentenweise per XOR addiert werden kann.

Für die Multiplikation ist mehr Aufwand zu treiben. Sind  $p(X) = p_{l-1}X^{l-1} + \dots + p_0$ ,  $q(X) = q_{l-1}X^{l-1} + \dots + q_0 \in GF(2^l)$ , so ist

$$p(X) \cdot q(X) = p_{l-1}X^{l-1}q(X) + \dots + p_1Xq(X) + p_0q(X),$$

und das Produkt  $pq$  in  $\mathbb{F}_2[X]$  lässt sich berechnen, indem die Einzelprodukte von  $q$  und denjenigen Monomen  $1, X, \dots, X^{l-1}$  aufaddiert werden, wo  $p_i = 1$ . In Hexadezimalnotation entsprechen die Monome genau den Zweierpotenzen, d.h.  $\mathbf{b}_x \cdot q(X)$  lässt sich berechnen als  $\mathbf{8}_x \cdot q(X) + \mathbf{2}_x \cdot q(X) + \mathbf{1}_x \cdot q(X)$ . Auf den Bitvektoren müssen also lediglich die Produkte der Zweierpotenzen  $\theta = 1, \dots, 2^{l-1}$  mit  $v = (x_{i,a}, x_{i,b}, \dots)$  in allgemeiner Form berechnet werden. Anschließend wird das entstehende Polynom modulo eines irreduziblen Polynoms vom Grad  $l$  reduziert, um das Resultat in  $GF(2^l)$  zu erhalten.<sup>3</sup> Haben wir ein solches Reduktionspolynom – etwa  $f(X)$  – fixiert, so erhalten wir mittels Polynomdivision eine Formel für den Bitvektor  $\theta \cdot v \pmod{f}$ , die lediglich Koeffizientenadditionen, d.h. XORs von Bits, enthält.

Auf diese Weise lässt sich die Matrix-Vektor-Multiplikation in  $GF(2^l)^m$  auf bitweise Operationen in  $lm$  Variablen  $x_{0,a}, \dots, x_{m-1,l}$  abbilden. Da lediglich XOR als Verknüpfung vorkommt, sind diese Abbildungen auf  $(\mathbb{F}_2^l)^m$  nicht nur konstruktionsbedingt MDS,

<sup>3</sup>Irreduzible Polynome der benötigten Grade können etwa Tabelle C in [25] entnommen werden.

sondern auch T-Funktionen. Erweitern wir also die  $lm$  Variablen von 1 auf  $n$  Bits, so erhalten wir nach Proposition 4.16 eine T-Funktion in  $lm$  logischen Variablen, die MDS- $(m+1)$  ist.

Als letzte Frage verbleibt zu klären, wie der Parameter  $l$  zu wählen ist. Offensichtlich muss  $GF(2^l)$  mindestens  $2m$  verschiedene Elemente enthalten, damit diese Konstruktion anwendbar ist. Es muss also  $2m \leq 2^l$  gelten. Um die Anzahl von Variablen möglichst gering zu halten, wird man in der Regel  $l$  unter dieser Rahmenbedingung so klein wie möglich wählen.

Die bislang allgemein beschriebene Konstruktionsmethode soll nun in einigen Fällen angewandt werden.

Für  $m = l = 1$  ergibt sich die bereits angesprochene triviale MDS-Abbildung, nämlich die Identität auf  $GF(2)$ : Sowohl für  $a_0 = 0$  als auch für  $a_0 = 1$  ist  $\mathcal{A}(a_0)$  die  $1 \times 1$ -Matrix 1, also ist  $\psi(x_0) = 1 \cdot 1^{-1} \cdot x_0 = x_0$ , was auf  $(\mathbb{F}_2^n)^1$  die MDS-T-Funktion  $f(x_0) = x_0$  liefert.

Für  $m = 2$  ist  $l \geq 2$  erforderlich. Zur Konstruktion einer MDS-Abbildung mit  $m = l = 2$  können wir dann etwa so vorgehen: Für  $GF(2^2)$  wählen wir das irreduzible Polynom  $f(X) = X^2 + X + 1$  und  $(a_0, a_1, a_2, a_3) = (0_x, 1_x, 2_x, 3_x)$ . Die lineare MDS-Abbildung  $\psi : GF(4)^2 \rightarrow GF(4)^2$  ergibt sich dann zu

$$\begin{aligned} \psi\left(\begin{pmatrix} x_0 \\ x_1 \end{pmatrix}\right) &= \mathcal{A}(0_x, 1_x) \mathcal{A}^{-1}(2_x, 3_x) \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \\ &= \begin{pmatrix} 1_x & 0_x \\ 1_x & 1_x \end{pmatrix} \begin{pmatrix} 1_x & 2_x \\ 1_x & 3_x \end{pmatrix}^{-1} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \\ &= \begin{pmatrix} 1_x & 0_x \\ 1_x & 1_x \end{pmatrix} \begin{pmatrix} 3_x & 2_x \\ 1_x & 1_x \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \\ &= \begin{pmatrix} 3_x & 2_x \\ 2_x & 3_x \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}. \end{aligned} \tag{4.15}$$

Die Variablen  $x_0, x_1$  werden nun als Bitvektoren  $(x_{0,a}, x_{0,b})$  und  $(x_{1,a}, x_{1,b})$  dargestellt. Um Multiplikation in  $GF(4)$  mit  $f(X)$  als Reduktionspolynom auf diesen Bitvektoren darzustellen, benötigen wir Formeln für die Produkte mit den Monomen 1 und  $X$ , also  $1_x \cdot x_i$  und  $2_x \cdot x_i$ . Aus diesen kann dann mittels Addition das Produkt  $3_x \cdot x_i$  bestimmt werden. Multiplikation mit  $1_x$  entspricht der Identität, für  $2_x$  erhalten wir  $(aX + b) \cdot X = aX^2 + bX$ , und Polynomdivision durch  $f$  liefert  $aX^2 + bX = a \cdot f(X) + \underbrace{(b-a)X - a}_{\text{Divisionsrest}}$ ,

also haben wir insgesamt

$$\begin{aligned} 1_x \cdot (x_{i,a}, x_{i,b}) &= (x_{i,a}, x_{i,b}), \\ 2_x \cdot (x_{i,a}, x_{i,b}) &= (x_{i,b} \oplus x_{i,a}, x_{i,a}), \end{aligned}$$

und damit kann nun (4.15) wie folgt als Abbildung auf  $(\mathbb{F}_2^2)^2$  dargestellt werden:

$$\begin{pmatrix} x_{0,a} \\ x_{0,b} \\ x_{1,a} \\ x_{1,b} \end{pmatrix} \mapsto \begin{pmatrix} (x_{0,b}) \oplus (x_{1,b} \oplus x_{1,a}) \\ (x_{0,b} \oplus x_{0,a}) \oplus (x_{1,a}) \\ (x_{0,b} \oplus x_{0,a}) \oplus (x_{1,b}) \\ (x_{0,a}) \oplus (x_{1,b} \oplus x_{1,a}) \end{pmatrix}. \tag{4.16}$$

Wird nun jedes  $x_i$ , als  $n$ -Bit-Wort gesehen, so ist (4.16) eine MDS-T-Funktion auf  $(\mathbb{F}_2^{2n})^2$ , also in zwei Komponenten à zwei Worten.

Für  $m = l = 3$  geben bereits Klimov und Shamir ein Beispiel [18].  $m = 4$  ist ebenfalls interessant und bei  $l = 3$  möglich, da  $2 \cdot 4 = 2^3$ . Zur Konstruktion einer solchen Abbildung wählen wir für  $GF(2^3)$  das Reduktionspolynom  $f(X) = X^3 + X + 1$  und  $(a_0, \dots, a_7) = (0_x, 7_x, 1_x, 6_x, 2_x, 5_x, 3_x, 4_x)$ ; dies ergibt die lineare MDS-Abbildung  $\psi : GF(8)^4 \rightarrow GF(8)^4$  mit

$$\begin{aligned}
\psi \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} &= \mathcal{A}(0_x, 7_x, 1_x, 6_x) \mathcal{A}^{-1}(2_x, 5_x, 3_x, 4_x) \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \\
&= \begin{pmatrix} 1_x & 0_x & 0_x & 0_x \\ 1_x & 7_x & 3_x & 2_x \\ 1_x & 1_x & 1_x & 1_x \\ 1_x & 6_x & 2_x & 7_x \end{pmatrix} \begin{pmatrix} 1_x & 2_x & 4_x & 3_x \\ 1_x & 5_x & 7_x & 6_x \\ 1_x & 3_x & 5_x & 4_x \\ 1_x & 4_x & 6_x & 5_x \end{pmatrix}^{-1} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \\
&= \begin{pmatrix} 1_x & 0_x & 0_x & 0_x \\ 1_x & 7_x & 3_x & 2_x \\ 1_x & 1_x & 1_x & 1_x \\ 1_x & 6_x & 2_x & 7_x \end{pmatrix} \begin{pmatrix} 4_x & 6_x & 1_x & 2_x \\ 7_x & 5_x & 0_x & 2_x \\ 5_x & 6_x & 2_x & 1_x \\ 7_x & 7_x & 7_x & 7_x \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \\
&= \begin{pmatrix} 4_x & 6_x & 1_x & 2_x \\ 6_x & 4_x & 2_x & 1_x \\ 1_x & 2_x & 4_x & 6_x \\ 2_x & 1_x & 6_x & 4_x \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}. \tag{4.17}
\end{aligned}$$

Es fällt auf, dass diese Matrix (wie bereits diejenige bei  $m = l = 2$ ) symmetrisch ist, was jedoch nicht zwangsläufig der Fall sein muss, wie das bereits angesprochene Beispiel für  $m = l = 3$  aus [18] zeigt.

Jede Variable  $x_i$  entspricht einem Bitvektor  $(x_{i,a}, x_{i,b}, x_{i,c})$ , für dessen Produkt mit den Monomen  $1, X, X^2$  (hexadezimal  $1_x, 2_x$  und  $4_x$ ) wir nun Formeln benötigen. Die Multiplikationsformel für  $6_x$  erhalten wir dann durch Addition derer für  $2_x$  und  $4_x$ . Für  $2_x \cdot x_i$  ergibt sich  $(aX^2 + bX + c) \cdot X = (aX^3 + bX^2 + cX)$  und nach Division durch  $f$  der Rest  $bX^2 + (c - a)X - a$ , für  $4_x \cdot x_i$  analog  $(aX^2 + bX + c) \cdot X^2 = aX^4 + bX^3 + cX^2$  und damit der Divisionsrest  $(c - a)X^2 + (a - b)X - b$ . Zusammenfassend haben wir

$$\begin{aligned}
1_x \cdot (x_{i,a}, x_{i,b}, x_{i,c}) &= (x_{i,a}, x_{i,b}, x_{i,c}), \\
2_x \cdot (x_{i,a}, x_{i,b}, x_{i,c}) &= (x_{i,b}, x_{i,c} \oplus x_{i,a}, x_{i,a}), \\
4_x \cdot (x_{i,a}, x_{i,b}, x_{i,c}) &= (x_{i,c} \oplus x_{i,a}, x_{i,a} \oplus x_{i,b}, x_{i,b}),
\end{aligned}$$

und damit

$$\begin{pmatrix} x_{0,a} \\ x_{0,b} \\ x_{0,c} \\ x_{1,a} \\ x_{1,b} \\ x_{1,c} \\ x_{2,a} \\ x_{2,b} \\ x_{2,c} \\ x_{3,a} \\ x_{3,b} \\ x_{3,c} \end{pmatrix} \mapsto \begin{pmatrix} (x_{0,c} \oplus x_{0,a}) \oplus (x_{1,b} \oplus x_{1,c} \oplus x_{1,a}) \oplus (x_{2,a}) \oplus (x_{3,b}) \\ (x_{0,a} \oplus x_{0,b}) \oplus (x_{1,c} \oplus x_{1,b}) \oplus (x_{2,b}) \oplus (x_{3,c} \oplus x_{3,a}) \\ (x_{0,b}) \oplus (x_{1,a} \oplus x_{1,b}) \oplus (x_{2,c}) \oplus (x_{3,a}) \\ (x_{0,b} \oplus x_{0,c} \oplus x_{0,a}) \oplus (x_{1,c} \oplus x_{1,a}) \oplus (x_{2,b}) \oplus (x_{3,a}) \\ (x_{0,c} \oplus x_{0,b}) \oplus (x_{1,a} \oplus x_{1,b}) \oplus (x_{2,c} \oplus x_{2,a}) \oplus (x_{3,b}) \\ (x_{0,a} \oplus x_{0,b}) \oplus (x_{1,b}) \oplus (x_{2,a}) \oplus (x_{3,c}) \\ (x_{0,a}) \oplus (x_{1,b}) \oplus (x_{2,c} \oplus x_{2,a}) \oplus (x_{3,b} \oplus x_{3,c} \oplus x_{3,a}) \\ (x_{0,b}) \oplus (x_{1,c} \oplus x_{1,a}) \oplus (x_{2,a} \oplus x_{2,b}) \oplus (x_{3,c} \oplus x_{3,b}) \\ (x_{0,c}) \oplus (x_{1,a}) \oplus (x_{2,b}) \oplus (x_{3,a} \oplus x_{3,b}) \\ (x_{0,b}) \oplus (x_{1,a}) \oplus (x_{2,b} \oplus x_{2,c} \oplus x_{2,a}) \oplus (x_{3,c} \oplus x_{3,a}) \\ (x_{0,c} \oplus x_{0,a}) \oplus (x_{1,b}) \oplus (x_{2,c} \oplus x_{2,b}) \oplus (x_{3,a} \oplus x_{3,b}) \\ (x_{0,a}) \oplus (x_{1,c}) \oplus (x_{2,a} \oplus x_{2,b}) \oplus (x_{3,b}) \end{pmatrix} \quad (4.18)$$

für die Darstellung von (4.17) als Abbildung auf  $(\mathbb{F}_2^3)^4$  bzw.  $(\mathbb{F}_2^{3n})^4$ , wenn die 12 Variablen jeweils als ganzes Wort interpretiert werden. Für größere  $m$  und  $l$  verläuft die Konstruktion in analoger Weise, lediglich die dabei entstehenden Funktionen tendieren zu gewisser Unübersichtlichkeit.

Wie bei T-Funktionen üblich sind die bislang konstruierten Abbildungen als Rohmaterial zu verstehen, welches mittels Parametern ausgestaltet werden kann. Parameter sind insbesondere vonnöten, wenn auf diesem Weg nichtlineare MDS-Abbildungen konstruiert werden sollen, denn die bisherigen Abbildungen sind allesamt linear. Ist  $(x_0, \dots, x_{m-1})^T \mapsto (y_0, \dots, y_{m-1})^T$  eine MDS-Abbildung, so ist  $(x_0, \dots, x_{m-1})^T \mapsto (\phi_0(y_0), \dots, \phi_{m-1}(y_{m-1}))^T$ , wobei die  $\phi_i$  invertierbare Funktionen sind, ebenfalls MDS, da alle  $\phi_i$  nach Voraussetzung injektiv sein müssen. Da  $\phi(x; \alpha) = x \oplus \alpha$  invertierbar ist, können also in den Bit-Slices beliebige additive Parameter eingeführt werden, ohne dass sich die MDS-Eigenschaft ändert. Additive Parameter können dabei sowohl durch die Verwendung von  $\pm$  an Stelle von XOR als auch durch explizite Addition eines  $\alpha(x)$  entstehen.

Die in (4.16) gegebene MDS-T-Funktion kann auf diese Weise in etwa die folgende nichtlineare MDS-Abbildung verwandelt werden:

$$\begin{pmatrix} x_{0,a} \\ x_{0,b} \\ x_{1,a} \\ x_{1,b} \end{pmatrix} \mapsto \begin{pmatrix} x_{0,b} \oplus (x_{1,b} + x_{1,a}) \oplus 4(x_{0,a} \wedge x_{1,b}) \\ (x_{0,b} \oplus x_{0,a}) - x_{1,a} \oplus (x_{1,b}x_{0,a} \vee 1)^2 \\ (x_{0,b} - x_{0,a}) \oplus x_{1,b} + \overline{6x_{0,b}x_{1,a}} \\ x_{0,a} + (x_{1,b} \oplus x_{1,a}) \oplus 2x_{0,b}x_{1,a} \end{pmatrix}. \quad (4.19)$$

Allgemein kann eine nach diesem Verfahren konstruierte Diffusionsebene  $m$  S-Boxen à  $ln$  Bits vermischen. Bei der Wahl der Parameter muss, wie bereits erwähnt, die Ungleichung  $2m \leq 2^l$  beachtet werden. Soll  $n$  nach der Wortbreite des Prozessors gewählt werden und  $mln$  der Blockgröße der Chiffre entsprechen, schränkt dies den Spielraum stark ein. Mit Funktion (4.19) können 2 S-Boxen à  $2n$  Bits vermischt werden, was etwa für ein 128-Bit-Feistelnetz mit zwei 32-Bit-S-Boxen in der auf 64 Bits operierenden  $f$ -Funktion geeignet sein kann, da dann 16-Bit-Arithmetik verwendet wird. Entsprechend

$b$	S-Box-Anzahl $m$	Galoiskörper $GF(2^l)$	Variablengröße $n$ [Bits]	S-Box-Größe $p$ [Bits]
32	8	$GF(2^4)$	1	4
	4	$GF(2^4)$	2	8
	2	$GF(2^2)$	8	16
48	8	$GF(2^6)$	1	6
	6	$GF(2^4)$	2	8
	4	$GF(2^3)$	4	12
	3	$GF(2^4)$	4	16
	2	$GF(2^3)$	8	24
	2	$GF(2^2)$	12	24
64	8	$GF(2^4)$	2	8
	4	$GF(2^4)$	4	16
	2	$GF(2^4)$	8	32
	2	$GF(2^2)$	16	32

Tabelle 4.8: Mögliche Parameterkombinationen für die Konstruktion von MDS-Diffusionsebenen in einem  $2b$ -Bit-Feistelnetz nach dem auf  $GF(2^l)$  basierenden Verfahren.

käme Funktion (4.18) für die Diffusionsebene eines 128-Bit-SP-Netzwerks mit 4 S-Boxen à 32 Bits in Frage, wobei dies 8-Bit-Arithmetik bedeuten würde.

Eine Aufstellung von denkbaren Kombinationsmöglichkeiten für die Parameter  $m, l$  und  $n$  geben Tabelle 4.8 für Feistel-Netze sowie Tabelle 4.9 auf der nächsten Seite für SP-Netzwerke. In ihnen werden generell nur bijektive  $p \times p$ -S-Boxen berücksichtigt; für die Blocklänge  $b$  (bei Feistel-Netzen: deren Hälfte) muss also  $b = mp$  gelten. Die möglichen Werte für  $l$  werden zusätzlich durch die Bedingungen  $l \mid b$  sowie  $2m \leq 2^l$  eingeschränkt. Entstehen dabei Variablengrößen und damit Wortbreiten von  $n \neq 8k$  für  $k \geq 1$ , so ist die Abbildung in Software nicht direkt effizient implementierbar; dies gilt insbesondere für die winzigen „Wortbreiten“  $n = 1, 2$ . Des Weiteren ist ersichtlich, dass für großes  $m$  auch ein hoher Erweiterungsgrad  $l$  benötigt wird, so dass sich diese Konstruktion wegen der erforderlichen hohen Variablenanzahl für viele parallele S-Boxen – die dann für eine gegebene Blockgröße zwangsläufig recht klein sind – nur sehr bedingt eignet. Für wenige, dafür aber große S-Boxen bietet sie hingegen eine Möglichkeit, effiziente nichtlineare Diffusionsebenen zu erhalten.

### 4.2.2 MDS- $m$ -Abbildungen mittels T-Funktionen

Wie wir gesehen haben, ist die bislang betrachtete, auf linearen MDS-Abbildungen in  $GF(2^l)$  basierende Konstruktionsmethode nicht für jeden Anwendungsfall geeignet: die

$b$	S-Box-Anzahl $m$	Galoiskörper $GF(2^l)$	Variablengröße $n$ [Bits]	S-Box-Größe $p$ [Bits]
64	8	$GF(2^4)$	2	8
	4	$GF(2^4)$	4	16
	2	$GF(2^4)$	8	32
	2	$GF(2^2)$	16	32
96	12	$GF(2^8)$	1	8
	8	$GF(2^6)$	2	12
	6	$GF(2^4)$	4	16
	4	$GF(2^3)$	8	24
	3	$GF(2^4)$	8	32
	2	$GF(2^3)$	16	48
	2	$GF(2^2)$	24	48
128	16	$GF(2^8)$	1	8
	8	$GF(2^4)$	4	16
	4	$GF(2^4)$	8	32
	2	$GF(2^8)$	8	64
	2	$GF(2^4)$	16	64
	2	$GF(2^2)$	32	64

Tabelle 4.9: Mögliche Parameterkombinationen für die Konstruktion von MDS-Diffusionsebenen in einem  $b$ -Bit-SP-Netzwerk nach dem auf  $GF(2^l)$  basierenden Verfahren.

Diffusionsebene der Blockchiffre *Rijndael*<sup>4</sup> muss etwa 16 S-Boxen à 8 Bits verarbeiten, was nach Tabelle 4.9 1-Bit-Arithmetik nach sich ziehen würde. Idealerweise würde eine MDS-Abbildung in  $m$  Komponenten lediglich  $m$  Variablen à  $n$  Bits benötigen, was mit T-Funktionen jedoch nicht realisierbar ist (vgl. Seite 116).

Dennoch müssen die Diffusionseigenschaften solcher T-Funktionen nicht zwingend unbrauchbar sein. Eine MDS- $m$ -Abbildung  $\phi$ , für welche also  $\text{Dist}_\phi = m$  gilt anstatt wie bei MDS-Abbildungen  $\text{Dist}_\phi = m + 1$ , würde in je zwei aufeinanderfolgenden Runden einer Blockchiffre  $m$  (statt  $m + 1$ ) aktive S-Boxen garantieren.

Da für  $l = 1$  Arithmetik in  $GF(2^l)$  mit der gewohnten Arithmetik in  $\mathbb{F}_2$  zusammenfällt, liefert der bisherige Ansatz die folgende Bit-Slice-Form:

$$\phi : \begin{pmatrix} x_0 \\ \vdots \\ x_{m-1} \end{pmatrix} \mapsto A \begin{pmatrix} x_0 \\ \vdots \\ x_{m-1} \end{pmatrix} \oplus \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{m-1} \end{pmatrix}, \quad (4.20)$$

wobei  $m > 1$  gilt,  $A \in \mathbb{F}_2^{m \times m}$  invertierbar sein muss und die  $\alpha_i$  für Parameter stehen. Daraus lassen sich dann auf dem üblichen Weg T-Funktionen auf  $(\mathbb{F}_2^n)^m$  konstruieren.

<sup>4</sup>Siehe [9]. Rijndael wurde 2001 vom US-amerikanischen National Institute of Standards and Technology als neuer Verschlüsselungsstandard AES ausgewählt.

Für die Beurteilung einer solchen Konstruktion entscheidend ist der Wert  $\text{Dist}_\phi$ . Da jener nicht von den additiven Parametern abhängt, können wir diese vernachlässigen, womit sich die Untersuchung auf Abbildungen der Form  $x \mapsto Ax$  mit regulärem  $A \in \mathbb{F}_2^{m \times m}$  beschränken lässt. Zur Abkürzung notieren wir dabei die durch  $A$  beschriebene lineare Abbildung mit  $\varphi_A$ . Solche Abbildungen können schon auf Grund der Ergebnisse in [18] maximal MDS- $m$  sein, die obere Schranke lässt sich jedoch auch folgendermaßen einsehen:

**Proposition 4.17.** *Sei  $m > 1$  und  $\varphi_A : \mathbb{F}_2^m \rightarrow \mathbb{F}_2^m$  die durch die reguläre Matrix  $A$  beschriebene lineare Abbildung. Dann gilt*

$$\text{Dist}_{\varphi_A} \leq m.$$

*Beweis.* Da  $A$  regulär ist, gilt  $\text{rank}(A) = m$  und somit muss es eine Spalte von  $A$  mit Hamming-Gewicht kleiner  $m$  geben (ansonsten bestünden zwei Spalten nur aus Einsen und wären somit im Widerspruch zur Regularitätsannahme linear abhängig). O.B.d.A. sei dies die Spalte  $A_{\cdot,j}$ . Bezeichnet  $\mathbf{e}_i$  den  $i$ -ten Einheitsvektor des  $\mathbb{F}_2^m$ , so gilt mit den Vektoren  $a := \mathbf{e}_j$  sowie  $b := 0$

$$\begin{aligned} \text{Dist}_{\varphi_A} &= \min_{\substack{x, x' \in \mathbb{F}_2^m \\ x \neq x'}} d_h(x, x') + d_h(\varphi_A(x), \varphi_A(x')) \\ &\leq d_h(a, b) + d_h(\varphi_A(a), \varphi_A(b)) \\ &= 1 + w_h(\varphi_A(\mathbf{e}_j) \oplus \varphi_A(0)) \\ &= 1 + w_h(\varphi_A(\mathbf{e}_j)) \\ &= 1 + w_h(A_{\cdot,j}) \\ &\leq 1 + (m - 1) \\ &= m, \end{aligned}$$

wie behauptet. □

Inspiziert durch ein Beispiel in [18] versuchen wir, MDS- $m$ -Matrizen auf die folgende Weise zu konstruieren:

$$\mathcal{C}_m := \begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix} - I_m, \quad (4.21)$$

wobei  $I_m$  die  $m \times m$ -Einheitsmatrix bezeichnet. Die Behauptung ist nun, dass auf diese Weise eine für gerades  $m$  reguläre  $m \times m$ -Matrix entsteht, die für  $m = 2, 3, 4$  MDS- $m$  ist. Wir bezeichnen mit  $\mathbf{e}_i$  den  $i$ -ten Einheitsvektor des  $\mathbb{F}_2^m$  und mit  $\mathbf{c}^{(i)}$  den Vektor  $(1, \dots, 1)^T - \mathbf{e}_i$ . Für die angesprochene Konstruktion gilt also

$$\mathcal{C}_m = (\mathbf{c}^{(0)} \quad \mathbf{c}^{(1)} \quad \dots \quad \mathbf{c}^{(m-1)}),$$

und zum Nachweis der Regularität für gerades  $m$  beobachten wir, dass für  $a_0, \dots, a_{m-1} \in \mathbb{F}_2$  aus

$$\sum_{i=0}^{m-1} a_i \mathbf{c}^{(i)} = \sum_{i=0}^{m-1} a_i ((1, \dots, 1)^T - \mathbf{e}_i) = 0$$

die Beziehung

$$\sum_{i=0}^{m-1} a_i (1, \dots, 1)^T = \sum_{i=0}^{m-1} a_i \mathbf{e}_i$$

folgt, was bei geradem  $m$  nur für  $a_0 = a_1 = \dots = a_{m-1} = 0$  erfüllt sein kann,<sup>5</sup> womit alle Spalten von  $\mathcal{C}_m$  linear unabhängig sind. Also ist der Rang von  $\mathcal{C}_m$  gleich  $m$  und  $\mathcal{C}_m$  damit regulär, falls  $m$  gerade ist.

Es bleibt zu zeigen, dass  $\varphi_{\mathcal{C}_m}$  für die angegebenen Werte von  $m$  eine MDS- $m$ -Abbildung auf  $\mathbb{F}_2^m$  ist. Nach Proposition 2.21 stimmt für lineare Abbildungen  $\text{Dist}_f$  mit der Branch Number  $\mathcal{B}(f)$  überein, also genügt es zu zeigen, dass  $\mathcal{B}(\varphi_{\mathcal{C}_m}) = m$ . Dazu stellen wir  $0 \neq v \in \mathbb{F}_2^m$  als Linearkombination  $v = \sum_{i=0}^{m-1} v_i \mathbf{e}_i$  der kanonischen Basisvektoren dar. Nach Abbildung durch  $\varphi_{\mathcal{C}_m}$  gilt

$$\begin{aligned} \varphi_{\mathcal{C}_m}(v) &= \mathcal{C}_m \left( \sum_{i=0}^{m-1} v_i \mathbf{e}_i \right) \\ &= \sum_{i=0}^{m-1} v_i (\mathcal{C}_m \mathbf{e}_i) \\ &= \sum_{i=0}^{m-1} v_i \mathbf{c}^{(i)} \\ &= \sum_{v_i=1} v_i \mathbf{c}^{(i)}. \end{aligned} \tag{4.22}$$

Aus der Konstruktion folgt, dass alle  $\mathbf{c}^{(i)}$  ein Hamming-Gewicht von  $m - 1$  haben, d.h.  $w_h(\mathbf{c}^{(i)}) = m - 1$ . Die Summe in (4.22) wird über  $w_h(v)$  viele verschiedene solche  $\mathbf{c}^{(i)}$  gebildet. Es ist leicht einzusehen, dass die (als komponentenweises XOR berechnete) Summe von  $k$  verschiedenen  $\mathbf{c}^{(i)}$  für  $m = 2, 3, 4$  einen Vektor  $d$  mit  $w_h(d) = m - k$  ergibt. Damit haben wir

$$\begin{aligned} \mathcal{B}(\varphi_{\mathcal{C}_m}) &= \min_{0 \neq v \in \mathbb{F}_2^m} w_h(v) + w_h(\varphi_{\mathcal{C}_m}(v)) \\ &= \min_{0 \neq v \in \mathbb{F}_2^m} w_h(v) + (m - w_h(v)) \\ &= m, \end{aligned}$$

wie gewünscht.

Leider lässt sich diese Methode nicht für allgemeines  $m$  verwenden, denn bereits für  $m = 5$  ist die Branch Number von

$$\mathcal{C}_5 = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

---

<sup>5</sup>Um rechts  $(1, \dots, 1)^T$  zu erhalten, müssten alle  $a_i = 1$  sein. Für gerades  $m$  entsteht dann aber auf der linken Seite eine 0. Sind einige  $a_i = 1$  und einige gleich 0, so kann die Gleichung ebenfalls nicht erfüllt sein, da dann links 0 oder  $(1, \dots, 1)^T$  steht und rechts keins von beidem.

lediglich gleich 4, da der Vektor  $v = (0, 0, 0, 1, 1)^T$  ein Fixpunkt von  $\varphi_{\mathcal{C}_5}$  ist und somit  $w_h(v) + w_h(\mathcal{C}_5 \cdot v) = 2 + 2 = 4$  gilt. Betrachtet man die Herleitung für  $m = 2, 3, 4$  genauer, so stellt sich der Schluss von  $w_h(\mathbf{c}^{(i)}) = m - 1$  auf das Hamming-Gewicht der Summe von  $k$  verschiedenen solchen Vektoren als diejenige Voraussetzung heraus, welche offenbar nur für  $n \leq 4$  erfüllt ist. Experimentell ergibt sich, dass die Matrizen  $\mathcal{C}_m$  für  $m = 6, 7, \dots, 16$  ebenfalls eine Branch Number von 4 besitzen, also der Fehlschlag dieser Methode bei  $m = 5$  kein singulärer Defekt ist. Die auf diese Weise konstruierbaren Matrizen sind also lediglich

$$\mathcal{C}_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \mathcal{C}_3 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}, \quad \mathcal{C}_4 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix},$$

welche für ihre Dimension allesamt MDS- $m$  sind (wobei  $\mathcal{C}_3$  nicht regulär ist). Ist jedoch eine solche Matrix gefunden, so lassen sich mittels Permutationsmatrizen  $P$  in  $A' = PA$  Zeilenpermutationen und in  $A'' = AP$  Spaltenpermutationen von  $A$  erhalten, welche dann ebenfalls MDS- $m$ -Matrizen sind, da diese Eigenschaft dabei nicht verändert wird:

**Proposition 4.18.** *Sei  $A \in \mathbb{F}_2^{m \times m}$  eine MDS- $m$ -Matrix und  $P$  eine beliebige Spaltenpermutation der Einheitsmatrix  $I_m$ . Dann sind die linearen Abbildungen  $\varphi_{PA}$  und  $\varphi_{AP}$  ebenfalls MDS- $m$ .*

*Beweis.* Nach Proposition 2.21 ist  $\varphi_A$  genau dann MDS- $m$ , wenn  $\mathcal{B}(\varphi_A) = m$ . Dies ist gleichbedeutend zu  $w_h(v) + w_h(Av) \geq m$  für alle  $v \in \mathbb{F}_2^m \setminus \{0\}$ , und so haben wir für alle  $x \in \mathbb{F}_2^m$  mit  $x \neq 0$

$$\begin{aligned} w_h(x) + w_h(\varphi_{AP}(x)) &= w_h(x) + w_h((AP)x) \\ &= w_h(x) + w_h(A(Px)) \\ &\geq w_h(x) + (m - w_h(Px)) \\ &= m, \end{aligned}$$

da Multiplikation mit  $P$  das Hamming-Gewicht nicht verändert. Also gilt  $\mathcal{B}(\varphi_{AP}) = m$ , was zu zeigen war. Für  $\varphi_{PA}$  kann der Nachweis in ähnlicher Weise geführt werden.  $\square$

Die Matrizen  $\mathcal{C}_m$  und ihre Permutationen sind für  $m = 2, 3, 4$  jedoch nicht die einzigen MDS- $m$ -Matrizen. In Tabelle 4.10 auf der nächsten Seite werden sämtliche MDS- $m$ -Matrizen für diese Dimensionen aufgeführt.

In Ermangelung einer allgemeinen Konstruktionsmethode ist für die Dimensionen  $m \geq 5$  eine computerbasierte Suche erforderlich, welche zufällig gewählte  $m \times m$ -Matrizen  $A$  auf  $\mathcal{B}(\varphi_A) = m$  überprüft, bis eine solche gefunden wurde. Für  $m = 5$  ergab eine vollständige Überprüfung aller  $2^{25}$  binären Matrizen keine Matrix  $A$  mit  $\mathcal{B}(\varphi_A) = 5$ . Damit ist  $\mathcal{C}_5$  mit einer Branch Number von 4 eine bestmögliche binäre  $5 \times 5$ -Matrix, was die Frage aufwirft, für welche  $m > 5$  überhaupt MDS- $m$ -Matrizen existieren. Eine experimentelle Untersuchung dieser Frage gestaltet sich jedoch sehr aufwändig, da die Anzahl der zu testenden Matrizen exponentiell wächst. Für  $m = 4$  wird bei einer zufallsgesteuerten Suche übrigens fast instantan eine der MDS-4-Matrizen aus Tabelle 4.10 gefunden.



### 4.2.3 Andere Konstruktionsprinzipien

Für die Konstruktion von Diffusionsebenen sind im Wesentlichen zwei andere Ansätze üblich: Bitpermutationen und lineare Abbildungen über endlichen Körpern.

Bitpermutationen werden vor allem in klassischen Blockchiffren wie etwa dem DES [5] eingesetzt. Sie sind stets linear, denn eine Transposition von  $n$  Bits lässt sich als Multiplikation mit einer permutierten Einheitsmatrix des  $\mathbb{F}_2^{n \times n}$  darstellen. Ihr größter Nachteil ist, dass sie die Hamming-Distanz eines Eingabepaares nicht verändern, d.h. für die Vektoren  $a = 0, b = (1, 0, \dots, 0) \in \mathbb{F}_2^n$  mit  $d_h(a, b) = 1$  gilt nach der Anwendung der Bitpermutation  $\sigma$  ebenfalls  $d_h(\sigma(a), \sigma(b)) = 1$ . Für die Branch Number einer Bitpermutation  $\sigma$  gilt also stets  $\mathcal{B}(\sigma) = 2$ , was nicht optimal ist.

Insbesondere seit der Einführung der **Wide-Trail-Strategie** durch Daemen und Rijmen ([8, 31]) werden  $m$ -dimensionale lineare Abbildungen über dem endlichen Körper  $GF(2^n)$  zur Diffusion eingesetzt. Mit Hilfe der Kodierungstheorie können lineare Abbildungen des  $GF(2^n)^m$  konstruiert werden, welche MDS sind und somit optimale Diffusion bewirken [25, 31]. Auch die in Abschnitt 4.2.1 besprochene Konstruktionsmethode für MDS-T-Funktionen benutzt solche lineare Abbildungen für die Bit-Slice-Formen. Jedoch können diese mittels additiven Parametern zu *nichtlinearen* MDS-Abbildungen ausgestaltet werden, was die Resistenz gegen differentielle und lineare Angriffe über das durch die S-Boxen gegebene Niveau hinaus erhöhen kann. Nachteilig im Vergleich zu den gewöhnlichen linearen MDS-Abbildungen sind allerdings die Einschränkungen bei der Parameterwahl, welche für nicht wenige Kombinationen für Mikroprozessorarithmetik ungünstige Wortbreiten erzwingen (vgl. Tabellen 4.8 auf Seite 122 und 4.9 auf Seite 123).

## 4.3 Effizienz

T-Funktionen, und damit alle bislang für Konfusions- und Diffusionsebene vorgestellten Abbildungen, lassen sich in der Regel in relativ wenigen Maschineninstruktionen ausdrücken und damit sehr effizient in Software implementieren. Dies setzt jedoch voraus, dass  $n$  nach der Wortbreite des Prozessors oder zumindest als Zweierpotenz zwischen 8 und  $n$  gewählt werden kann, was sich mitunter nur schwer realisieren lässt. Soll etwa eine T-Funktion als 8-Bit-S-Box verwendet werden, so sind nach den Ergebnissen aus Abschnitt 4.1.4 für gute Nichtlinearitätseigenschaften zumindest 4 Variablen erforderlich, welche dann nur noch 2 Bit breit sind.

Für so kleine T-Funktionen bietet sich daher eine auf ihrer Wertetabelle basierende Implementierung an; dabei wird die Funktion  $f$  so in einem Array  $A$  abgespeichert, dass  $A[v] = f(v)$  gilt. Eine Funktionsauswertung besteht folglich noch aus einem Arrayzugriff. Die Größe der Tabelle für eine Funktion  $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$  beträgt dabei  $n \cdot 2^n$  Bits. Während dies für die angesprochene 8-Bit-S-Box unproblematisch ist ( $8 \cdot 2^8$  Bit entsprechen 256 Byte), sind für  $n = 16$  bereits 128 KByte nötig, was insbesondere für eingebettete Systeme zuviel sein kann; für noch größeres  $n$  wird die Speicherung schnell unmöglich. Ein weiterer Vorteil dieses Vorgehens ist, dass die inverse Funktion  $f^{-1}$  dann ebenfalls

sehr leicht als Wertetabelle implementiert werden kann, zu deren Erzeugung lediglich die Wertetabelle von  $f$  vorliegen muss und somit keine algebraische Beschreibung für  $f^{-1}$  benötigt wird.

Für großes  $n$ , etwa 32-Bit-S-Boxen und insbesondere die Diffusionsfunktionen aus Abschnitt 4.2, ist der Tabellenansatz untauglich, die Funktion also normal über ihre Berechnungsvorschrift zu implementieren. Besonders problematisch ist bei solchen T-Funktionen die Implementierung der Inversen. Bei nichtlinearen Funktionen ist aus der algebraischen Beschreibung von  $f$  nicht direkt eine solche für  $f^{-1}$  ablesbar. In diesen alles andere als seltenen Fällen muss die Inversion auf der Basis der Bit-Slices bei der LSB beginnend Bit für Bit erfolgen, was wesentlich ineffizienter ist als die Vorwärtsberechnung. Für den Spezialfall, dass die T-Funktion modulo  $2^n$  als Polynom in einer Veränderlichen dargestellt werden kann, geben Klimov und Shamir in [16] ein auf vorausberechneten Tabellen basierendes schnelleres Inversionsverfahren an. Auf (womöglich sogar breite) T-Funktionen in mehreren Variablen oder nicht als Polynom modulo  $2^n$  darstellbare Funktionen ist dieses Verfahren nicht anwendbar.

Es existieren jedoch Fälle, in denen die Berechnung von  $f^{-1}$  komplett vermieden werden kann. In Feistel-Netzen muss die  $f$ -Funktion lediglich vorwärts berechnet werden (vgl. Abschnitt 2.2.4), so dass eine langsame Invertierung keine Rolle spielt. Gleiches gilt für den Counter-Betriebsmodus (und die meisten eine Stromchiffre emulierenden Modi), welcher nur die Verschlüsselungsfunktion der Blockchiffre nutzt, siehe Abschnitt 2.1.2. Im Counter-Modus müssen also insbesondere auch die Ebenen eines SP-Netzwerks nicht invertiert werden.

## 4.4 Sicherheit

Im Sinne der in Abschnitt 2.2 beschriebenen praktischen Sicherheit betrachten wir eine Blockchiffre als sicher, wenn jeder bekannte kryptoanalytische Angriff ineffizienter als Brute Force ist. Bei konkreten Entwürfen kann deren Sicherheitsniveau ausgelotet werden, indem möglichst viele gute Kryptoanalytiker deren Resistenz gegen alle bekannten (und darüber hinaus an der speziellen Struktur der Chiffre orientierten) Angriffstechniken untersuchen. Für Blockchiffren, deren Komponenten aus den in den letzten Abschnitten ermittelten T-Funktionen bestehen, beschränken wir uns im Rahmen dieser Arbeit darauf, deren Resistenz gegen differentielle und lineare Kryptoanalyse abzuschätzen. Insbesondere wollen wir für eine gegebene Auswahl an Komponentenfunktionen eine geeignete Rundenanzahl  $r$  bestimmen, nach der differentielle und lineare Angriffe sehr wahrscheinlich ineffektiv werden.

Um harte Schranken für ein solches  $r$  zu erhalten, müssen so genannte **Differentiale** bzw. **lineare Hüllen** betrachtet werden, in denen alle Charakteristiken bzw. Approximationen zusammengefasst werden, die sich lediglich in für die Kryptoanalyse irrelevanten Zwischenwerten unterscheiden [20]. Da dies die Analyse wesentlich erschwert, beschränken wir uns auf einzelne Charakteristiken und Approximationen; für viele Chiffren dominiert darüber hinaus eine Charakteristik das Differential bzw. eine Approximation die Hülle. Weiterhin setzen wir wie üblich Unabhängigkeit der Runden voraus.

Die in Gleichung (2.9) angegebene Wahrscheinlichkeit einer differentiellen Charakteristik hängt offensichtlich lediglich von den Einzelwahrscheinlichkeiten der *aktiven* nichtlinearen Komponenten der Rundenfunktion ab (für lineare Komponenten sind diese 1). Besteht die Rundenfunktion  $g$  aus affiner Schlüsseladdition, einer Konfusions- und einer Diffusionsebene, so ist die Wahrscheinlichkeit einer differentiellen  $\rho$ -Runden-Charakteristik  $\Xi$  beschränkt durch

$$\Pr(\Xi) \leq (\Omega_s)^A \cdot (\Omega_p)^\rho, \quad (4.23)$$

wobei  $\Omega_s$  das maximale differentielle Potenzial einer S-Box,  $\Omega_p$  dasjenige der Diffusionsschicht und  $A$  die Gesamtanzahl der aktiven S-Boxen ist (vgl. [31], Kapitel 7.1.3). Bei linearen Diffusionsebenen ist  $\Omega_p = 1$ , was den zweiten Faktor verschwinden lässt. Da auch die nichtlinearen Diffusionsabbildungen in Abschnitt 4.2 aus linearen Bit-Slices entstehen, ist nach den analytischen und experimentellen Resultaten aus Abschnitt 4.1.4 zu erwarten, dass auch für sie  $\Omega_p = 1$  gelten wird. Auch wenn sie nicht wie affine Abbildungen in jeder Zeile ihres Differenzenprofils einen Maximaleintrag aufweisen werden, gebietet es dennoch die Vorsicht, ihre Nichtlinearität nicht mit „einzuplanen“, sondern als Bonus zu verstehen. Damit reduziert sich (4.23) zu

$$\Pr(\Xi) \leq (\Omega_s)^A. \quad (4.24)$$

Da der Bias linearer Approximationen mit Hilfe des Piling-Up-Lemmas aus den Biasen der Ein-Runden-Approximationen bestimmt wird (Korollar 2.29), lässt sich der Bias  $\epsilon$  einer gesamten Approximation analog nach oben beschränken:

$$|\epsilon| \leq 2^{A-1} \cdot (|\epsilon_s|)^A, \quad (4.25)$$

wobei  $\epsilon_s$  den maximalen Bias einer S-Box und  $A$  die Anzahl der aktiven S-Boxen bezeichnet. (Dieser Ausdruck lässt sich alternativ via  $|\epsilon_s| = \frac{1}{2}\sqrt{\Lambda_s}$  mittels des linearen Potenzials formulieren.)

Besteht die Konfusionssebene aus  $m$  parallelen S-Boxen und die Diffusionsebene aus einer MDS-Abbildung oder MDS- $m$ -Abbildung, so können untere Schranken für die Anzahl aktiver S-Boxen in einer bestimmten (von der Struktur der Blockchiffre abhängigen) Zahl aufeinanderfolgender Runden formuliert werden. In Verbindung mit (4.24) sowie (4.25) lässt sich dadurch ein Richtwert für die maximale Wahrscheinlichkeit einer  $r$ -Runden-Charakteristik und den maximalen Bias einer linearen  $r$ -Runden-Approximation ableiten. Es sei ausdrücklich betont, dass dieser Richtwert für jeden konkreten Entwurf erneut zu überprüfen ist und keinesfalls unbedacht übernommen werden sollte.

## 4.5 T-Funktionsbasierte Feistel-Netze

Nachdem in den vorigen Abschnitten die Eignung von T-Funktionen für den Einsatz in Konfusions- und Diffusionsebenen von Blockchiffren untersucht wurde, wollen wir solche nun zur Konstruktion von Feistel-Netzen verwenden. Dabei konzentrieren wir uns auf  $f$ -Funktionen, die aus einer affinen Schlüsseladdition, parallel angewandten bijektiven S-Boxen  $s$  und einer abschließenden Diffusionsfunktion  $p$  bestehen.

### 4.5.1 Einsatzparameter

Zunächst sind die Blocklänge  $n = 2b$  und die Anzahl  $m = b/q$  der parallelen  $q$ -Bit-S-Boxen festzulegen, dann kann für diese Parameter eine MDS- oder MDS- $m$ -Abbildung zur Diffusion gewählt werden. Auch wenn bei auf T-Funktionen basierenden S-Boxen das Vorhandensein einer algebraischen Beschreibung die Wahl sehr breiter S-Boxen ermöglicht, so sind doch aus Abschnitt 4.1.4 keine invertierbaren T-Funktionen bekannt, deren differentielle und lineare Potenziale mit steigender Wortbreite sinken, was stark für kleinere S-Boxen spricht.

Auf Basis der differentiellen und linearen Potenziale der S-Boxen kann für eine solche Konfiguration dann mit Hilfe der Ungleichungen (4.24) und (4.25) eine empfehlenswerte Rundenanzahl veranschlagt werden, ab der differentielle und lineare Angriffe wirkungslos zu werden versprechen. Eine solche Anzahl ist erreicht, wenn die maximale Wahrscheinlichkeit einer differentiellen Charakteristik geringer ist als  $2^{-(n-1)}$ , denn für einen Angriff wären dann mehr als  $2^{n-1}$  Paare bzw.  $2^n$  Klartexte erforderlich. Analog ist die Schranke für den maximalen Bias linearer Approximationen  $2^{-(n/2)}$ , da dann für einen linearen Angriff mehr als  $(2^{n/2})^2 = 2^n$  Klartexte benötigt würden. Ist die Schlüssellänge  $k$  geringer als die Blocklänge, so kann in diesen Ausdrücken  $n$  durch  $k$  ersetzt werden, da in diesem Fall analytische Angriffe ab da uninteressant werden, wo ihr Aufwand denjenigen der exhaustiven Schlüsselsuche übersteigt.

Bei Feistel-Netzen muss davon ausgegangen werden, dass eine  $r$ -Runden-Chiffre mit einer  $r - 3$ -Runden-Charakteristik bzw.  $r - 2$ -Runden-Approximation gebrochen werden kann [1, 27], so dass wir zu der aus den Schranken ermittelten Rundenanzahl stets noch drei addieren. Darüber hinaus können durch geschickte Parameterwahl Charakteristiken und Approximationen mit „Nullrunden“ konstruiert werden. Können solche mit sich selbst konkateniert werden, reduziert sich die Anzahl der aktiven S-Boxen auch bei hohen Rundenzahlen trotz MDS-Diffusionsebene erheblich. Da die Existenz solcher Charakteristiken bzw. Approximationen über zwei Runden eine Differenzenpropagierung der Form  $\Delta_a \rightarrow 0$  bzw. eine S-Box-Approximation der Form  $0 \bullet X \oplus \beta \bullet f(X)$  voraussetzt, sind solche für Feistel-Netze mit bijektiver  $f$ -Funktion (wie in unserem Fall) nicht möglich. Für  $\rho \geq 3$  lassen sich jedoch iterierbare  $\rho$ -Runden-Charakteristiken formulieren, in denen eine Runde mit Wahrscheinlichkeit 1 bzw. Bias  $1/2$  passiert wird [19]. Folglich gehen wir davon aus, anstatt in *zwei* in jeweils *drei* aufeinanderfolgenden Runden  $A$  aktive S-Boxen zu haben, wobei  $A \geq m + 1$  für MDS- und  $A \geq m$  für MDS- $m$ -Abbildungen.

Auf diesen Überlegungen basierende sinnvolle Parameterkombinationen sind in Tabelle 4.11 auf der nächsten Seite zusammengefasst. Es ist sehr deutlich zu sehen, dass angesichts der von der S-Box-Breite unabhängigen differentiellen und linearen Potenziale viele kleinere parallele S-Boxen zu bevorzugen sind. Für die Schlüssellänge werden keine konkreten Vorschläge gemacht, da diese relativ unabhängig von den anderen Parametern wählbar ist.

Blocklänge $N=2b$	$m$ S-Boxen $s$	MDS-Abbildung $p: (\mathbb{F}_2^{l \cdot n})^m \rightarrow (\mathbb{F}_2^{l \cdot n})^m$	Empfohlene Rundenzahl $r$	Schranke für $\Pr(\Xi)$	Schranke für $ \epsilon $
64	$2 \times s : \mathbb{F}_2^{16} \rightarrow \mathbb{F}_2^{16}$	$p : (\mathbb{F}_2^{2 \cdot 8})^2 \rightarrow (\mathbb{F}_2^{2 \cdot 8})^2$	18	$2^{-70}$	$2^{-32.5}$
64	$4 \times s : \mathbb{F}_2^8 \rightarrow \mathbb{F}_2^8$	$p^* : (\mathbb{F}_2^{1 \cdot 8})^4 \rightarrow (\mathbb{F}_2^{1 \cdot 8})^4$	15	$2^{-75}$	$2^{-34.5}$
128	$2 \times s : \mathbb{F}_2^{32} \rightarrow \mathbb{F}_2^{32}$	$p : (\mathbb{F}_2^{4 \cdot 8})^2 \rightarrow (\mathbb{F}_2^{4 \cdot 8})^2$	33	$2^{-140}$	$2^{-64}$
128	$4 \times s : \mathbb{F}_2^{16} \rightarrow \mathbb{F}_2^{16}$	$p^* : (\mathbb{F}_2^{1 \cdot 16})^4 \rightarrow (\mathbb{F}_2^{1 \cdot 16})^4$	24	$2^{-149.5}$	$2^{-68}$
128	$8 \times s : \mathbb{F}_2^8 \rightarrow \mathbb{F}_2^8$	$p : (\mathbb{F}_2^{4 \cdot 2})^8 \rightarrow (\mathbb{F}_2^{4 \cdot 2})^8$	15	$2^{-168.5}$	$2^{-76}$

Die S-Box  $s$  basiert auf  $\pi_8$  aus Tabelle 4.6, d.h. besteht aus deren Erweiterung auf die angegebene Wortbreite gemäß (4.4). Für  $p$  wird die auf  $GF(2^l)$  basierende MDS-Konstruktion aus Abschnitt 4.2.1 verwendet, was in der Diffusionsebene insbesondere  $n$ -Bit-Arithmetik bedeutet. Sinkt dabei  $n$  unter 8, wird wenn möglich eine MDS- $m$ -Funktion  $p^*$  verwendet (es liegen nur für  $m = 2, 3, 4$  MDS- $m$ -Konstruktionen vor, siehe Abschnitt 4.2.2). Die angegebenen Schranken gelten für  $r - 3$  Runden.

Tabelle 4.11: Parameterkombinationen und Rundenzahlen für T-Funktionsbasierte Feistel-Netze.

## 4.5.2 Exemplarische Konstruktion

Aus für Konfusion und Diffusion geeigneten T-Funktionen soll nun als Beispiel ein kleines Feistel-Netz erstellt werden. Die Parameter werden dabei absichtlich künstlich klein gewählt, um erstens mit den dem Verfasser zur Verfügung stehenden Ressourcen eine praktische Implementierung von Angriffen zu erlauben, und um zweitens den Beispielcharakter zu unterstreichen.<sup>6</sup>

Das konstruierte Feistel-Netz  $\mathcal{FN}$  ist eine 16-Bit-Blockchiffre mit einer demnach auf 8 Bit operierenden  $f$ -Funktion. Jene besteht aus zwei identischen bijektiven 4-Bit-S-Boxen  $s$  und einer *nichtlinearen* MDS-Diffusionsabbildung  $p$ . Schlüsseladdition erfolgt per XOR, die Rundenzahl beträgt 4. Die Schlüssellänge beträgt 32 Bits, die Runden Schlüssel  $K^1, \dots, K^4$  entsprechen den vier Bytes des Schlüssels, beim MSB beginnend. Algorithmus 4.7 auf der nächsten Seite enthält eine Beschreibung des Verfahrens.

Die S-Box ist dabei eine T-Funktion in 4 Variablen bei einer Wortbreite von  $n = 1$ . Sie entspricht der Permutation  $\pi_4$  aus Tabelle 4.6 auf Seite 113, welcher auch zu entnehmen ist, dass  $\Omega_s = \Lambda_s = \frac{1}{4}$ . Die Diffusionsfunktion  $p$  entspricht der auf  $GF(2^2)$  basierenden MDS-Konstruktion für  $m = 2$  mit additiven Parametern, welche in (4.19) beschrieben ist; die Faktoren 4 und 6 wurden durch 2 ersetzt. Wie um die in Abschnitt 4.4 formulierten Bedenken zu bestätigen, gilt  $\Omega_p = \Lambda_p = 1$ , obwohl  $p$  nichtlinear ist. Im Gegensatz zu einer linearen Abbildung sind  $\delta_p$  und  $\lambda_p$  allerdings nur in einigen wenigen Fällen gleich 1. Sowohl  $s$  als auch  $p$  werden am besten mit Tabellen implementiert.

Welche Rundenzahl wäre empfehlenswert? Nach 9 Runden erhalten wir bei mindestens

<sup>6</sup>Wie bei der Diskussion praktischer Sicherheit angedeutet, wäre für die Rechtfertigung einer realistisch dimensionierten Chiffre ein den Rahmen dieser Arbeit übersteigender Analyseaufwand zu treiben.

---

**Algorithmus 4.7** T-Funktionsbasiertes 16-Bit-Feistelnetz  $\mathcal{FN}$ .

---

S-Box  $s : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4 : s = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 13 & 12 & 4 & 10 & 3 & 8 & 11 & 6 & 16 & 0 & 9 & 1 & 5 & 2 & 14 & 7 \end{pmatrix}$ .

Diffusionsfunktion  $p : (\mathbb{F}_2^{2 \cdot 2})^2 \rightarrow (\mathbb{F}_2^{2 \cdot 2})^2$ :

$$\begin{pmatrix} x_{0,a} \\ x_{0,b} \\ x_{1,a} \\ x_{1,b} \end{pmatrix} \mapsto \begin{pmatrix} x_{0,b} \oplus (x_{1,b} + x_{1,a}) \oplus 2(x_{0,a} \wedge x_{1,b}) \\ (x_{0,b} \oplus x_{0,a}) - x_{1,a} \oplus (x_{1,b}x_{0,a} \vee 1)^2 \\ (x_{0,b} - x_{0,a}) \oplus x_{1,b} + \overline{2x_{0,b}x_{1,a}} \\ x_{0,a} + (x_{1,b} \oplus x_{1,a}) \oplus 2x_{0,b}x_{1,a} \end{pmatrix}.$$

Innere Rundenfunktion  $f : (\mathbb{F}_2^8 \times \mathbb{F}_2^8) \rightarrow \mathbb{F}_2^8$ :

**Input:** Rechte Hälfte  $r$ , Rundenschlüssel  $K^i$

- 1:  $y \leftarrow r \oplus K^i$
- 2:  $[y]_{7,\dots,4} \leftarrow s([y]_{7,\dots,4})$
- 3:  $[y]_{3,\dots,0} \leftarrow s([y]_{3,\dots,0})$
- 4: **return**  $p(y)$ .

Die Verschlüsselung erfolgt über 4 Runden wie durch (2.3) beschrieben, wobei nach der letzten Runde die Vertauschung von  $L$  und  $R$  unterbleibt. Für die Entschlüsselung werden die Rundenschlüssel in umgekehrter Reihenfolge angewandt.

---

Testvektoren:

Klartext = 0000<sub>x</sub>,  $k = 01234567_x \Rightarrow$  Chiffretext = c842<sub>x</sub>

Klartext = 0001<sub>x</sub>,  $k = 01234567_x \Rightarrow$  Chiffretext = afb6<sub>x</sub>

---

3 aktiven S-Boxen je drei Runden  $\Pr(\Xi) \leq (\frac{1}{4})^{3 \cdot 3} = 2^{-18}$  für differentielle Charakteristiken und  $|\epsilon| \leq 2^{9-1} \cdot (\frac{1}{4})^9 = 2^{-10}$  für lineare Approximationen. Die Anzahl der benötigten Klartexte liegt dann mindestens bei  $2^{19}$  bzw.  $(2^{10})^2 = 2^{20}$ , was bei  $n = 16$  unmöglich ist. Plus drei weitere Runden erhalten wir  $r = 12$  als Richtwert.<sup>7</sup> Für das im Beispiel gewählte  $r = 4$  sind also effiziente Angriffe zu erwarten.

**Kryptoanalyse** Bei  $|\mathcal{K}| = 2^{32}$  ist eine exhaustive Schlüsselsuche möglich, mittels differentieller oder linearer Kryptoanalyse kann der Aufwand jedoch erheblich reduziert werden.

Betrachten wir zunächst **differentielle Angriffe**. Mit einer 3-Runden-Charakteristik würde ein normaler 1R-Angriff 8 Schlüsselbits liefern; da die Rundenschlüssel jedoch so klein sind, können wir problemlos gleichzeitig auf  $K^4$  und  $K^3$  zählen, indem wir einen 2R-Angriff durchführen, der auf einer 2-Runden-Charakteristik mit höherer Wahrscheinlichkeit basiert und direkt 16 Schlüsselbits liefert. Als Charakteristik verwenden wir eine Konkatenierung der trivialen 1-Runden-Charakteristik  $((l', 0), (0, l'))$  mit  $((0, l'), (l', B' \oplus 0))$ , wobei  $B'$  die wahrscheinliche Ausgabedifferenz der  $f$ -Funktion bei Eingabedifferenz  $l'$  bezeichnet. Dabei wählen wir  $l'$  derart, dass

---

<sup>7</sup>Selbstverständlich sind die inhärenten Probleme der sehr kleinen Block- und Schlüssellänge auch durch hohes  $r$  nicht zu kompensieren.

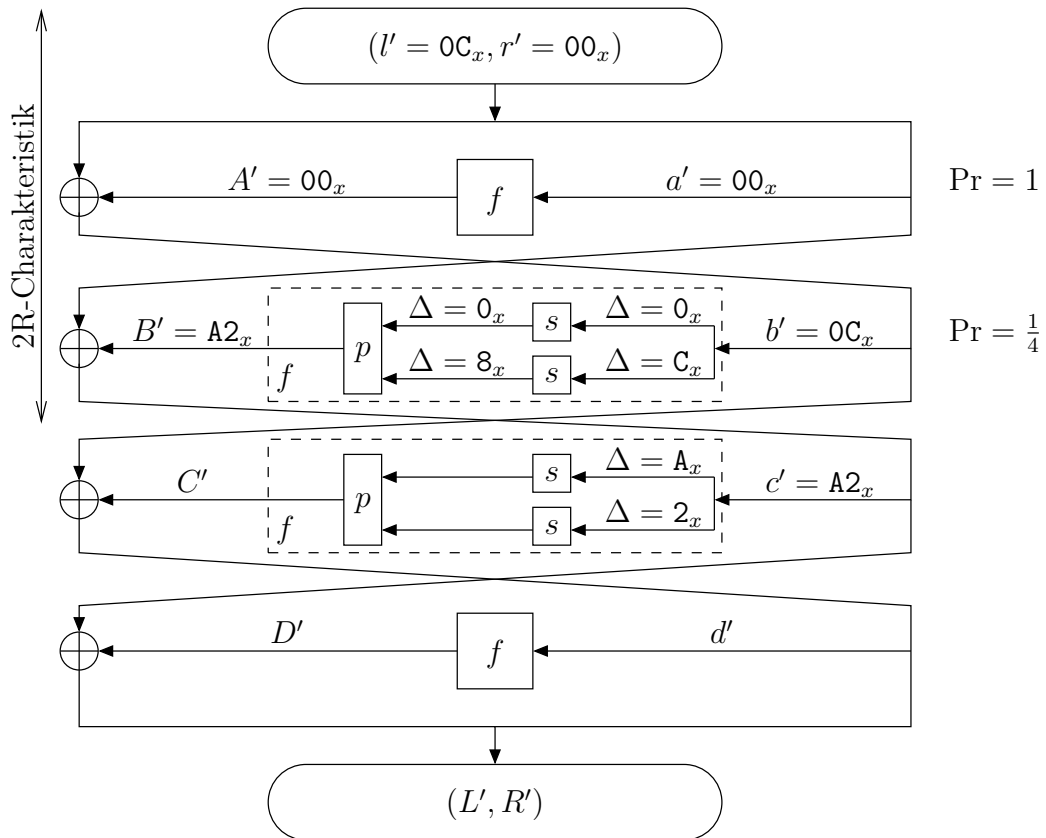


Abbildung 4.4: Für einen 2R-Angriff nutzbare differentielle 2-Runden-Charakteristik des Feistel-Netzes  $\mathcal{FN}$ .

- (a) in der zweiten Runde nur eine S-Box aktiv ist und
- (b) die nichtlineare  $p$ -Funktion in der zweiten Runde mit Wahrscheinlichkeit 1 passiert werden kann.

Dies gilt etwa für  $l' = 0C_x$ , womit wir die in Abbildung 4.4 dargestellte Charakteristik mit Wahrscheinlichkeit  $p = 1/4$  erhalten. Für die Anzahl der benötigten gewählten Klartext-Chiffretext-Paare erwarten wir also ein kleines Vielfaches von  $p^{-1} = 4$ . Eine direkte Implementierung dieses Angriffs ergab bei 20 bzw. 30 Paaren eine Erfolgswahrscheinlichkeit von etwa 80% bzw. 96%; fünf bis sieben gute Paare genügen also zur Identifizierung der korrekten Schlüsselbits von  $K^3$  und  $K^4$ . Die verbleibenden 16 Schlüsselbits können anschließend mit einer 1-Runden-Charakteristik für  $K^2$  und Brute Force für  $K^1$  gefunden werden.

In Abbildung 4.4 lässt sich übrigens auch die Wirkung der MDS-Abbildung  $p$  erkennen: Jenseits der Feistel-spezifischen „Nullrunde“ war in Runde 2 eine S-Box aktiv, was in Runde 3 gemäß MDS zu  $2+1-1 = 2$  aktiven S-Boxen führt (die Eingabedifferenzen sind  $A_x$  und  $2_x$ ). Hätten wir für  $p$  eine Bitpermutation gewählt, hätte die Eingabedifferenz  $08_x$  in Runde 2 eine Ausgabedifferenz mit Hamming-Gewicht 1 zur Folge, wonach in Runde 3 lediglich eine S-Box aktiv sein könnte.

Für einen **linearen Angriff** gehen wir ähnlich zum differentiellen vor und verwenden eine 2-Runden-Approximation, hier jedoch für die *mittleren* zwei Runden, so dass die Rundenschlüssel  $K^1$  und  $K^4$  gesucht werden. Einen Auswahlvektor  $\alpha \in \mathbb{F}_2^8$  für einen halben Block notieren wir dabei als Hexadezimalzahl; mit  $P_i^L$  bzw.  $C_i^R$  bezeichnen wir die linke bzw. rechte Hälfte der Eingabe bzw. Ausgabe der  $i$ -ten Runde, wobei  $C_i^L = P_i^L \oplus f(P_i^R, K^i)$  und  $C_i^R = P_i^R$ .

Die beste lineare 1-Runden-Approximation ist selbstverständlich diejenige, welche auf der Nullapproximation  $0 \bullet X \oplus 0 \bullet f(X)$  der  $f$ -Funktion basiert:

$$\mathcal{L}_1 : (0 \bullet P_i^L \oplus \alpha \bullet P_i^R) \oplus (0 \bullet C_i^L \oplus \alpha \bullet C_i^R) = 0, \quad (4.26)$$

oder kurz:  $(0, \alpha) \rightarrow (0, \alpha)$ . Sie gilt für beliebiges  $\alpha$  mit Wahrscheinlichkeit 1, hat also einen Bias von  $\epsilon_1 = \frac{1}{2}$ . Die besten nichttrivialen Approximationen der  $f$ -Funktion haben einen Bias von  $\pm \frac{1}{4}$ ; eine solche ist etwa  $01_x \bullet X \oplus 45_x \bullet f(X)$ , welche wir zur 1-Runden-Approximation  $(45_x, 01_x) \rightarrow (45_x, 0)$  ausbauen können:

$$\mathcal{L}_2 : (45_x \bullet P_i^L \oplus 01_x \bullet P_i^R) \oplus (45_x \bullet C_i^L \oplus 0 \bullet C_i^R) = 0, \quad (4.27)$$

denn der Term  $01_x \bullet P_i^R$  taucht zwei Mal auf und kürzt sich daher weg. Es gilt  $|\epsilon_2| = \frac{1}{4}$ . Unter Berücksichtigung der  $(L, R)$ -Vertauschung zwischen zwei Runden lassen sich  $\mathcal{L}_2$  und  $\mathcal{L}_1$  zu einer 2-Runden-Approximation  $(45_x, 01_x) \rightarrow (0, 45_x)$  kombinieren, wobei  $\alpha = 45_x$  gewählt wird:

$$\begin{aligned} \mathcal{L}_2 &: [(45_x \bullet P_2^L \oplus 01_x \bullet P_2^R) \oplus \cancel{(45_x \bullet C_2^L)}] = 0 \\ \oplus \mathcal{L}_1 &: [\cancel{(45_x \bullet P_3^R)} \oplus (45_x \bullet C_3^R)] = 0 \\ \rightsquigarrow \mathcal{L} &: [(45_x \bullet P_2^L \oplus 01_x \bullet P_2^R) \oplus (45_x \bullet C_3^R)] = 0. \end{aligned} \quad (4.28)$$

Die durchgestrichen Terme können gekürzt werden, da in einem Feistel-Netz  $P_3^R = C_2^L$  gilt. Für den Bias der 2-Runden-Approximation  $\mathcal{L}$  gilt nach dem Piling-Up-Lemma  $|\epsilon| = 2 \cdot |\epsilon_1| |\epsilon_2| = \frac{1}{4}$ , wie erwartet.

Ein linearer Angriff würde nun also  $N = c \cdot |\epsilon|^{-2} = c \cdot 16$  bekannte Klartexte wählen, für alle möglichen Werte von  $K^1$  und  $K^4$  die Anzahl der Paare zählen, für die  $\mathcal{L}$  gilt, und schließlich denjenigen Schlüssel  $(K^1, K^4)$  mit maximal von  $N/2$  abweichendem Zählwert ausgeben. Allerdings stellt sich heraus, dass die Linearkombination  $45_x \bullet f(X)$  nicht von den höherwertigen 4 Bits von  $X$  abhängig ist. Damit hängt beim Test auf  $\mathcal{L}$  nach Probeentschlüsselung mit dem Rundenschlüssel  $K^4$  der Term  $45_x \bullet C_3^R$  nur von den niederwertigen 4 Bits von  $K^4$  ab; bezüglich der höherwertigen 4 Bits verhält sich der Wert wie zufällig, so dass im Zählverfahren für diese Bits keine Entscheidung möglich ist. Statt auf allen 16 Bits der Rundenschlüssel  $K^1, K^4$  zählen zu können, müssen wir uns also auf insgesamt 12 Bits beschränken.

Diese 12 Schlüsselbits wurden in einer Implementierung dieses Angriffs bei 64 bekannten Klartexten mit etwa 75% Erfolgswahrscheinlichkeit identifiziert, bei 128 Klartexten steigt die Erfolgsrate auf 98%. Diese Werte entsprechen  $c = 4$  bzw.  $c = 8$ , womit sie denjenigen für Matsuis Angriff auf DES auffallend ähnlich sind (vgl. [27], Tabelle 3).

## 4.6 T-Funktionsbasierte SP-Netzwerke

In diesem Abschnitt sollen – wie für Feistel-Netze bereits geschehen – Einsatzparameter für SP-Netzwerke betrachtet werden, welche in Konfusions- und Diffusionsebene aus T-Funktionen bestehen.

### 4.6.1 Einsatzparameter

Die Auswahl der S-Boxen und Diffusionsfunktionen sowie die Abschätzung der besten differentiellen Charakteristiken bzw. linearen Approximationen kann vollkommen analog zur Situation bei den Feistel-Netzen geschehen (siehe Abschnitt 4.5.1). Der hauptsächliche Unterschied ist, dass auf Grund der Struktur von SP-Netzwerken keine „Nullrunden“ möglich sind und somit in je zwei aufeinanderfolgenden Runden tatsächlich von  $m + 1$  bzw.  $m$  aktiven S-Boxen ausgegangen werden kann. Des Weiteren sind  $2R$ - oder  $3R$ -Angriffe im Allgemeinen nicht durchführbar. Nichtsdestotrotz fügen wir den von den Schranken gelieferten Rundenzahlen jeweils noch zwei Runden als Sicherheitsabstand hinzu.

Eine auf Basis dieser Überlegungen erstellte Liste denkbarer Parameter für die zwei gängigsten Blocklängen  $n = 64$  und  $n = 128$  findet sich in Tabelle 4.12 auf Seite 138. Auffällig ist, dass erneut die Diffusionseigenschaften der T-Funktionen wesentlich stärker wiegen als die durch sie bewirkte Konfusion. Da in einem SP-Netzwerk in jeder Runde der gesamte Block verarbeitet wird, sind durchgängig weniger Runden zu veranschlagen als in vergleichbaren Feistel-Netzen.

### 4.6.2 Exemplarische Konstruktion

Nachdem bereits ein kleines T-Funktionsbasiertes Feistel-Netz vorgestellt wurde, soll nun auch ein SP-Netzwerk aus T-Funktionen konstruiert werden, wobei erneut alle Parameter künstlich klein gewählt sind: Die Blocklänge beträgt  $n = 24$  Bit, die Rundenfunktion besteht aus drei parallelen 8-Bit-S-Boxen  $s$  und einer *linearen* MDS-3-Abbildung  $p$  (als solche ist übrigens eine effiziente Inversion möglich). Da auch nach der letzten Runde eine Schlüsseladdition erfolgen sollte, gibt es fünf 24-Bit-Rundenschlüssel, welche aus einem 80-Bit-Hauptschlüssel abgeleitet werden.<sup>8</sup> Die Details sind in Algorithmus 4.8 auf der nächsten Seite dargestellt.

Für die S-Box gilt  $\Omega_s = \frac{5}{128}$  und  $\Lambda_s = \frac{225}{4096}$ , also haben wir für den maximalen Bias  $|\epsilon_s| = \frac{1}{2}\sqrt{\Lambda_s} = \frac{15}{128}$ . Da die Diffusionsfunktion  $p$  MDS-3 ist, sind in je zwei aufeinanderfolgenden Runden 3 S-Boxen aktiv. Damit können wir eine empfehlenswerte Rundenzahl wie folgt ermitteln: Die Wahrscheinlichkeit einer differentiellen 4-Runden-Charakteristik  $\Xi$  ist beschränkt durch  $\Pr(\Xi) \leq \left(\frac{5}{128}\right)^{2 \cdot 3} \approx 2^{-28}$ , für den maximalen Bias linearer Approximationen haben wir  $|\epsilon| \leq 2^{6-1} \cdot \left(\frac{15}{128}\right)^6 \approx 2^{-13,5}$ . Die Anzahl der benötigten Klartexte beträgt dann mindestens  $2^{29}$  bzw.  $(2^{13,5})^2 = 2^{27}$ , was bei  $n = 24$  unmöglich ist. Mit zwei

<sup>8</sup>Fünf unabhängige Rundenschlüssel (also  $5 \cdot 24 = 120$  Bit Schlüssellänge) wären für eine 24-Bit-Chiffre exzessiv,  $5 \cdot 8 = 40$  Bit sehr wenig; daher die Wahl von  $5 \cdot 16 = 80$  Bit, welche dann zu fünf 24-Bit-Rundenschlüsseln expandiert werden.

---

**Algorithmus 4.8** T-Funktionsbasiertes 24-Bit-SP-Netzwerk  $\mathcal{SPN}$ .

---

S-Box  $s : \mathbb{F}_2^8 \rightarrow \mathbb{F}_2^8$ , gegeben durch die Wertetabelle von  $\pi_8$  in Tabelle 4.6.

Diffusionsfunktion  $p : (\mathbb{F}_2^8)^3 \rightarrow (\mathbb{F}_2^8)^3$ :

$$\begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} \mapsto \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_0 \oplus x_1 \oplus x_2 \\ x_0 \oplus x_2 \\ x_0 \oplus x_1 \end{pmatrix}.$$

Rundenfunktion  $g : (\mathbb{F}_2^{24} \times \mathbb{F}_2^{24}) \rightarrow \mathbb{F}_2^{24}$ :

**Input:** Zustand  $s^i = (y_0, y_1, y_2)$ , Rundenschlüssel  $K^i$

- 1:  $s^i \leftarrow s^i \oplus K^i$
- 2:  $(y_0, y_1, y_2) \leftarrow (s(y_0), s(y_1), s(y_2))$
- 3: **return**  $p(y_0, y_1, y_2)$ .

KEY-SCHEDULE:

**Input:** 80-Bit-Hauptschlüssel  $k = (k_1, \dots, k_{10}) \in (\mathbb{F}_2^8)^{10}$

**Output:** Rundenschlüssel  $(K^1, \dots, K^5) \in (\mathbb{F}_2^{24})^5$

- 1: **for**  $i = 1 \dots 5$  **do**
- 2:    $K^i \leftarrow (k_{2i-1}, k_{2i}, k_{2i-1} \times k_{2i} \bmod 2^8)$
- 3: **end for**
- 4: **return**  $(K^1, \dots, K^5)$ .

Verschlüsselungsfunktion  $e_k$ :

**Input:** Klartext  $P$ , Schlüssel  $k$

**Output:** Chiffretext  $C$

- 1:  $(K^1, \dots, K^5) \leftarrow \text{KEY-SCHEDULE}(k)$
- 2: **for**  $i = 1 \dots 4$  **do**
- 3:    $P \leftarrow g(P, K^i)$
- 4: **end for**
- 5: **return**  $P \oplus K^5$ .

Zur Entschlüsselung wird das SPN rückwärts abgearbeitet, wobei an Stelle von  $s$  und  $p$  ihre Inversen  $s^{-1}, p^{-1}$  zum Einsatz kommen.

---

Testvektoren:

Klartext = 000000<sub>x</sub>,  $k = 0123456789abcdef0123_x \Rightarrow$  Chiffretext = 36f9f5<sub>x</sub>

Klartext = 000001<sub>x</sub>,  $k = 0123456789abcdef0123_x \Rightarrow$  Chiffretext = 17009d<sub>x</sub>

---

Blocklänge $N$	$m$ S-Boxen $s$	MDS-Abbildung $p: (\mathbb{F}_2^{l \cdot n})^m \rightarrow (\mathbb{F}_2^{l \cdot n})^m$	Empfohlene Rundenzahl $r$	Schranke für $\Pr(\Xi)$	Schranke für $ \epsilon $
64	$2 \times s : \mathbb{F}_2^{32} \rightarrow \mathbb{F}_2^{32}$	$p : (\mathbb{F}_2^{2 \cdot 16})^2 \rightarrow (\mathbb{F}_2^{2 \cdot 16})^2$	12	$2^{-70}$	$2^{-32.5}$
64	$4 \times s : \mathbb{F}_2^{16} \rightarrow \mathbb{F}_2^{16}$	$p^* : (\mathbb{F}_2^{1 \cdot 16})^4 \rightarrow (\mathbb{F}_2^{1 \cdot 16})^4$	10	$2^{-75}$	$2^{-34.5}$
64	$8 \times s : \mathbb{F}_2^8 \rightarrow \mathbb{F}_2^8$	$p : (\mathbb{F}_2^{4 \cdot 2})^8 \rightarrow (\mathbb{F}_2^{4 \cdot 2})^8$	6	$2^{-84}$	$2^{-38.5}$
128	$2 \times s : \mathbb{F}_2^{64} \rightarrow \mathbb{F}_2^{64}$	$p : (\mathbb{F}_2^{2 \cdot 32})^2 \rightarrow (\mathbb{F}_2^{2 \cdot 32})^2$	22	$2^{-140}$	$2^{-64}$
128	$4 \times s : \mathbb{F}_2^{32} \rightarrow \mathbb{F}_2^{32}$	$p : (\mathbb{F}_2^{4 \cdot 8})^4 \rightarrow (\mathbb{F}_2^{4 \cdot 8})^4$	14	$2^{-140}$	$2^{-64}$
128	$8 \times s : \mathbb{F}_2^{16} \rightarrow \mathbb{F}_2^{16}$	$p : (\mathbb{F}_2^{4 \cdot 4})^8 \rightarrow (\mathbb{F}_2^{4 \cdot 4})^8$	10	$2^{-168.5}$	$2^{-76}$
128	$16 \times s : \mathbb{F}_2^8 \rightarrow \mathbb{F}_2^8$	$p : (\mathbb{F}_2^{8 \cdot 1})^{16} \rightarrow (\mathbb{F}_2^{8 \cdot 1})^{16}$	6	$2^{-159}$	$2^{-72}$

Die S-Box  $s$  basiert auf  $\pi_8$  aus Tabelle 4.6, d.h. besteht aus deren Erweiterung auf die angegebene Wortbreite gemäß (4.4). Für  $p$  wird die auf  $GF(2^l)$  basierende MDS-Konstruktion aus Abschnitt 4.2.1 verwendet, was in der Diffusionsebene insbesondere  $n$ -Bit-Arithmetik bedeutet. Sinkt dabei  $n$  unter 8, wird wenn möglich eine MDS- $m$ -Funktion  $p^*$  verwendet (es liegen nur für  $m = 2, 3, 4$  MDS- $m$ -Konstruktionen vor, siehe Abschnitt 4.2.2). Die angegebenen Schranken gelten für  $r - 2$  Runden.

Tabelle 4.12: Parameterkombinationen und Rundenzahlen für T-Funktionsbasierte SP-Netzwerke.

weiteren Runden als Sicherheitsabstand erhalten wir  $r = 6$  Runden als Empfehlung. Unser Beispiel hat vier Runden, was dem Richtwert bereits recht nahe kommt.

**Kryptoanalyse** Exhaustive Schlüsselsuche ist angesichts einer Schlüssellänge von 80 Bit unrealistisch.

Für einen **differentiellen 1R-Angriff** benötigen wir eine 3-Runden-Charakteristik. Um deren Gesamtwahrscheinlichkeit zu maximieren, sollten in ihr so wenig S-Boxen wie möglich aktiv sein. Da  $p$  eine MDS-3-Abbildung ist, beträgt die minimale Anzahl aktiver S-Boxen über 3 Runden vier: je eine aktive in der ersten und dritten, zwei in der zweiten Runde. Nach Gleichung (2.7) lassen sich dabei die Differenzen nach Anwendung von  $p$  mittels der Funktionsgleichung aus den vorherigen Differenzen errechnen. Auf diese Weise können wir etwa die folgende 3-Runden-Charakteristik  $\Xi_3$  mit vier aktiven S-Boxen ermitteln (dabei steht ein Tripel für die Differenzen der drei Zustandsbytes):

$$\begin{aligned}
\text{Klartext:} & \quad (0, \Delta_a, 0) \\
R1 : & \quad (0, \Delta_a, 0) \rightarrow (\Delta_b, 0, \Delta_b) \\
R2 : & \quad (\Delta_b, 0, \Delta_b) \rightarrow (0, 0, \Delta_c) \\
R3 : & \quad (0, 0, \Delta_c) \rightarrow (\Delta_d, \Delta_d, 0).
\end{aligned}$$

Es verbleibt, die Werte der  $\Delta_i$  so zu wählen, dass  $\Pr(\Xi_3)$  maximal wird. Aus der Differenzentabelle von  $s$  ergeben sich die folgenden sechs Differenzenpaare mit Wahrschein-

lichkeit  $\frac{5}{128}$ :

$$\begin{array}{ll} 1F_x \rightarrow 88_x, & 72_x \rightarrow 75_x, \\ 59_x \rightarrow 46_x, & CF_x \rightarrow 83_x, \\ 6F_x \rightarrow 10_x, & D8_x \rightarrow 80_x. \end{array}$$

Eines dieser Paare wird als  $\Delta_b \rightarrow \Delta_c$  gewählt, um die Wahrscheinlichkeit der beiden parallel aktiven S-Boxen zu maximieren. Leider gibt es keine zwei Paare der Form  $(a \rightarrow b, b \rightarrow c)$ , so dass für  $\Delta_a \rightarrow \Delta_b$  und  $\Delta_c \rightarrow \Delta_d$  schlechtere Paare gewählt werden müssen. Es ergibt sich, dass diese „Anschlusspaare“ beim ersten Paar aus obiger Liste am günstigsten gewählt werden können, nämlich als Differenzenpaare mit Wahrscheinlichkeit  $\frac{4}{128}$ . Insgesamt erhalten wir die in Abbildung 4.5 auf der nächsten Seite dargestellte Charakteristik. Ihre Wahrscheinlichkeit beträgt  $\Pr(\Xi_3) = (\frac{4}{128})^2 \cdot (\frac{5}{128})^2 = \frac{25}{2^{24}} \approx 2^{-19,4}$ . Damit werden für einen Angriff mindestens  $2^{20,4}$  gewählte Klartexte benötigt, was bereits nahe an der Gesamtanzahl der 24-Bit-Blöcke liegt.

Diese beste ermittelte Charakteristik kommt der sich nach (4.24) ergebenden oberen Schranke  $\Pr(\Xi) \leq (\frac{5}{128})^4 \approx 2^{-18,7}$  für Drei-Runden-Charakteristiken sehr nahe. Da sich nach nochmaliger Anwendung der  $p$ -Funktion in Runde 4 die Chiffretextdifferenz  $(0, \Delta_e, 0)$  ergibt, kann in Algorithmus 2.1 darüber hinaus eine effektive Filterung falscher Paare erfolgen, indem getestet wird, ob die ersten und letzten Bytes der Chiffretexte identisch sind.

Analog suchen wir für einen **linearen Angriff** eine möglichst gute lineare Approximation der ersten drei Runden des SPN. Erneut sind bestenfalls Approximationen mit vier aktiven S-Boxen möglich. Deren Auswahl muss nun derart erfolgen, dass sich die Zwischenterme gegenseitig aufheben; dabei ist der Einfluss der linearen Diffusionsfunktion  $p$  zu beachten. Als Muster für eine solche Approximation wurde das Folgende gefunden:

$$\begin{array}{l} R1 : (0, 0, \alpha) \bullet (P_0^1, P_1^1, P_2^1) \oplus (\beta, 0, \beta) \bullet (C_0^1, C_1^1, C_2^1) \\ R2 : (\beta, 0, \beta) \bullet (P_0^2, P_1^2, P_2^2) \oplus (0, \gamma, 0) \bullet (C_0^2, C_1^2, C_2^2) \\ R3 : (0, \gamma, 0) \bullet (P_0^3, P_1^3, P_2^3) \oplus (\delta, \delta, 0) \bullet (C_0^3, C_1^3, C_2^3). \end{array}$$

Dabei bezeichnen die Tripel  $(P_0^i, P_1^i, P_2^i)$  bzw.  $(C_0^i, C_1^i, C_2^i)$  die drei Zustandsbytes vor bzw. nach der  $i$ -ten Runde (unter Vernachlässigung der Schlüsseladdition). Insgesamt erhalten wir also die die ersten drei Runden umspannende Approximation

$$\mathcal{L} : \alpha \bullet P_2^1 \oplus \delta \bullet (C_0^3 \oplus C_1^3) = 0. \quad (4.29)$$

Zur Begründung ihrer Korrektheit bezeichnen wir mit  $s_j^i$  bzw.  $S_j^i$  die Ein- bzw. Ausgabe der  $j$ -ten S-Box in Runde  $i$  und erhalten mit Hilfe der Funktionsgleichung von  $p$

$$\begin{aligned} & \alpha \bullet s_2^1 \oplus \beta \bullet (S_2^1 \oplus s_0^2 \oplus s_2^2) \oplus \gamma \bullet (S_0^2 \oplus S_2^2 \oplus s_1^3) \oplus \delta \bullet S_1^3 \\ = & \alpha \bullet s_2^1 \oplus \beta \bullet (S_2^1 \oplus (S_0^1 \oplus S_1^1 \oplus S_2^1) \oplus (S_0^1 \oplus S_1^1)) \\ & \oplus \gamma \bullet (S_0^2 \oplus S_2^2 \oplus (S_0^2 \oplus S_2^2)) \oplus \delta \bullet S_1^3 \\ = & \alpha \bullet s_2^1 \oplus \delta \bullet S_1^3; \end{aligned}$$

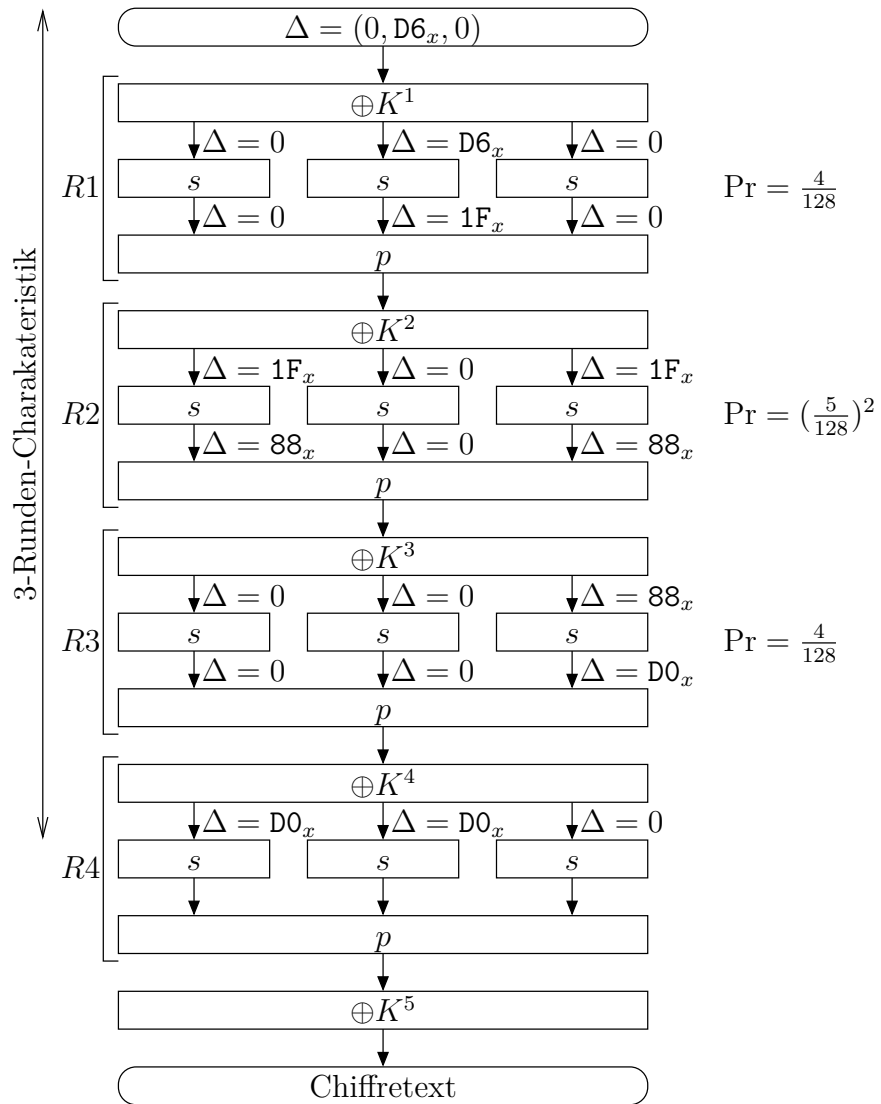


Abbildung 4.5: Beste ermittelte differentielle 3-Runden-Charakteristik von  $SPN$ .

und da  $s_0^4 \oplus s_1^4 = (S_0^3 \oplus S_1^3 \oplus S_2^3) \oplus (S_0^3 \oplus S_2^3) = S_1^3$ , ergibt dies gerade (4.29).

Zur konkreten Wahl der Vektoren betrachten wir die sich aus der linearen Approximationstabelle von  $s$  ergebende Liste der S-Box-Approximationen mit einem bestmöglichen Bias von  $\pm \frac{15}{128}$ :

$(05_x, 6B_x),$	$(65_x, ED_x),$	$(A5_x, 4D_x),$
$(13_x, 05_x),$	$(66_x, AA_x),$	$(B0_x, F8_x),$
$(1C_x, 38_x),$	$(7A_x, 8C_x),$	$(B1_x, 43_x),$
$(37_x, 5A_x),$	$(A4_x, AE_x),$	$(E7_x, C3_x),$
$(48_x, 43_x);$		

wobei das als „Tandem“ einsetzbare Paar  $(13_x, 05_x), (05_x, 6B_x)$  sofort ins Auge sticht. Darüber, ob diese Kombination für die Runden 1+2 oder 2+3 eingesetzt wird, entscheidet die Tatsache, dass für  $6B_x$  bessere Fortsetzungen existieren als Vorgänger für  $13_x$ . Zusammenfassend ergibt sich die in Abbildung 4.6 auf der nächsten Seite illustrierte lineare Approximation. Für ihren Bias erhalten wir  $|\epsilon_{\mathcal{L}}| = 2^3 \cdot (\frac{15}{128})^3 \cdot \frac{11}{128} \approx 2^{-9,8}$ , für einen linearen Angriff werden also mindestens  $(2^{9,8})^2 = 2^{19,6}$  bekannte Klartexte benötigt. Ein Vergleich mit der durch (4.25) beschriebenen oberen Schranke  $|\epsilon| \leq 2^3 \cdot (\frac{15}{128})^4 \approx 2^{-9,4}$  verdeutlicht, wie nahe  $\mathcal{L}$  dem Optimum kommt.

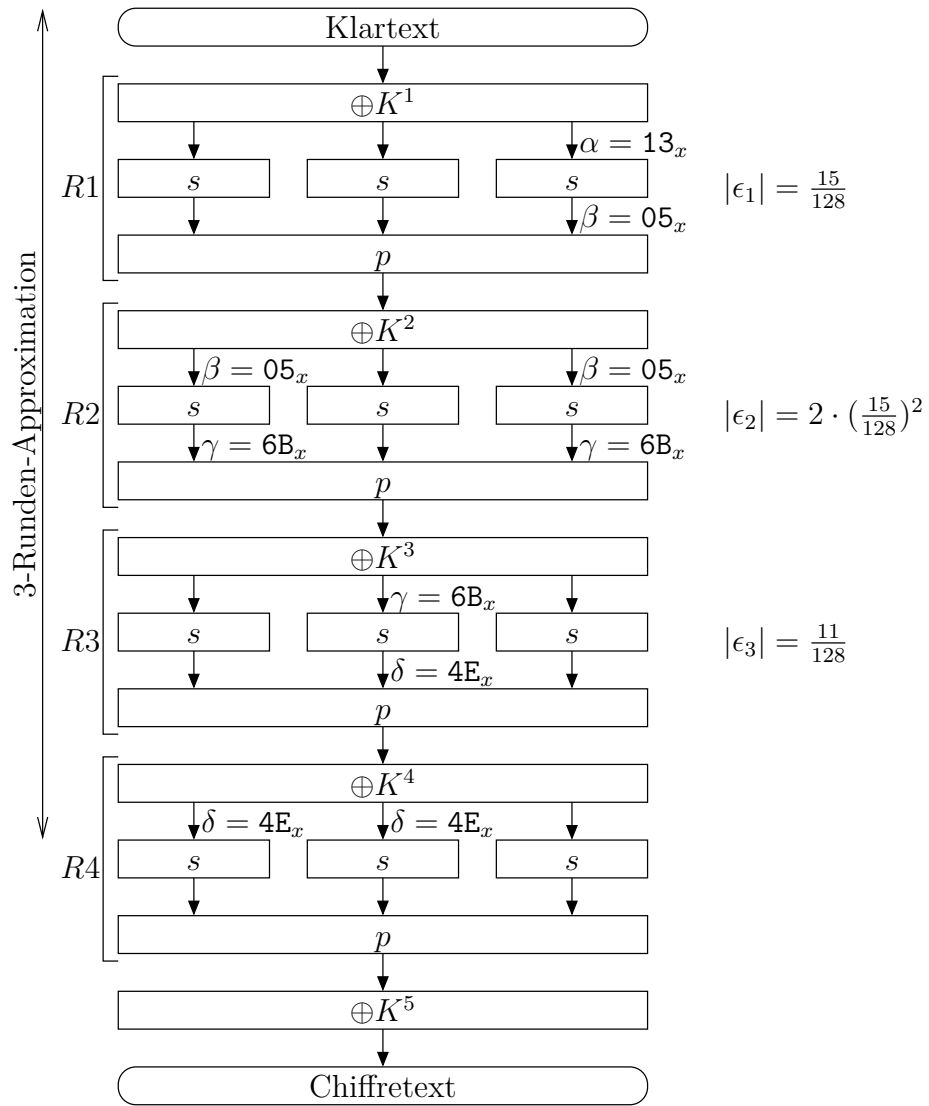


Abbildung 4.6: Beste ermittelte lineare Approximation der ersten 3 Runden von  $SPN$ .

# Kapitel 5

## Zusammenfassung

### 5.1 Ergebnis

Wir wollen die untersuchten Möglichkeiten zum Einsatz von T-Funktionen in den Konfusions- und Diffusionsebenen iterierter Blockchiffren nun abschließend zusammenstellen und bewerten.

In der Konfusionebene können T-Funktionen als algorithmische S-Boxen eingesetzt werden, was insbesondere effiziente Konstruktionsmethoden für große S-Boxen verspricht, welche nicht mehr als Tabelle realisiert werden können. Zunächst wurden nach den von Klimov und Shamir formulierten Konstruktionsprinzipien erzeugte invertierbare T-Funktionen für kleine Wortbreiten experimentell auf ihre differentiellen und linearen Potenziale untersucht. Dabei haben sich auf linearen Bit-Slices basierende (selbst jedoch durchaus nichtlineare) T-Funktionen als vollkommen ungeeignet erwiesen, da ihre differentiellen und linearen Potenziale jeweils schlechtestmöglich ausfielen. Für bis zu zwei Variablen sowie korrespondierende breite T-Funktionen ( $1 \times 2$  bzw.  $2 \times 2$  Variablen) konnte diese Beobachtung bezüglich des linearen Potenzials auch für allgemeines  $n$  gezeigt werden.

Als für S-Boxen wesentlich geeigneter haben sich dagegen auf der algebraischen Normalform der Bit-Slices von Permutationen basierende T-Funktionen herausgestellt. Für diese Konstruktionsform konnten wir allgemein zeigen, wie sich Differenzentabelle und Linearitätsprofil mittels Produktformeln aus denjenigen ihrer Bit-Slices bestimmen lassen. Dabei hat sich herausgestellt, dass die differentiellen und linearen Potenziale dieser Funktionen bereits vollständig durch diejenigen ihrer niederwertigsten Bit-Slice (LSB) festgelegt sind. Sie ändern sich somit bei steigender Variablenlänge nicht, sondern sind nur von der *Anzahl* der Variablen abhängig. Aus diesen Beobachtungen ließen sich für jedes  $k \geq 1$  untere Schranken für das differentielle und lineare Potenzial solcher Funktionen in  $k$  Variablen ableiten, welche sich für einige Werte von  $k$  als scharf erwiesen haben.

Insgesamt gesehen haben sich auch die ANF-basierten T-Funktionen für den Einsatz in S-Boxen nicht sonderlich nachhaltig empfohlen. Für kleine S-Boxen sind die differentiellen und linearen Potenziale im Vergleich zu gleichdimensionalen anderen Konstruktionen gut, jedoch sind für kleine S-Boxen keine algebraischen Funktionsbeschreibungen nötig, da solche ohnehin via Table-Lookup implementiert werden und sich mittels zufallsgesteuerter Suche vergleichbar gute S-Boxen ermitteln lassen. Bei großen S-Boxen, also

gerade dort, wo T-Funktionen mit ihrer Funktionsvorschrift punkten könnten, sind zwar die Potenziale nach den in dieser Arbeit entwickelten Produktformeln effizient berechenbar; da diese jedoch nicht mit der Wortbreite  $n$  fallen, werden sie mit steigendem  $n$  sogar immer ungeeigneter: Während etwa für eine 8-Bit-S-Box eine als Tabelle implementierte ANF-basierte T-Funktion in 8 Variablen à einem(!) Bit mit  $\Omega_f = \frac{5}{128}$  und  $\Lambda_f = \frac{225}{4096}$  durchaus konkurrenzfähig ist, würden sich bei einer Verbreiterung der Variablen auf 4 Bit in einer 32-Bit-S-Box exakt die gleichen Potenziale ergeben, was für diese S-Box-Größe nicht mehr akzeptabel ist. Von der Idealvorstellung,  $n$  nach der Wortbreite des Prozessors zu wählen, muss zumeist ohnehin Abstand genommen werden, da gezeigt werden konnte, dass alle invertierbaren T-Funktionen in einer Variablen ein lineares Potenzial von 1 besitzen.

Für Diffusionszwecke lassen sich T-Funktionen hingegen effektiv einsetzen, da nach dem von Klimov und Shamir entwickelten Verfahren MDS-Abbildungen für eine beliebige Anzahl  $m$  von parallelen S-Boxen konstruiert werden können. Dadurch wird definitionsgemäß bestmögliche Diffusion erreicht. Ein Nachteil dieser Konstruktion sind die mitunter unflexiblen Anforderungen an die Anzahl logischer Variablen, welche für großes  $m$  bei vorgegebener Blocklänge Arithmetik auf zunehmend kleiner Wortbreite nach sich ziehen. Dafür lassen sich auf diese Weise nichtlineare MDS-Abbildungen konstruieren, was für die Resistenz der Chiffre gegen kryptoanalytische Angriffe vorteilhaft sein kann.

Zur Konstruktion einer T-Funktion für die Diffusion von  $m$  S-Boxen, welche lediglich aus  $m$  Variablen bestehen muss, wurden ausgehend von einem Beispiel von Klimov und Shamir MDS- $m$ -Abbildungen betrachtet, welche sich aus einer binären  $m \times m$ -Matrix konstruieren lassen, deren korrespondierende lineare Abbildung eine Branch Number von  $m$  besitzt. Bei der Untersuchung solcher Abbildungen hat sich herausgestellt, dass das angesprochene, für  $m = 4$  gegebene Beispiel nicht ohne Weiteres verallgemeinerbar ist: Während eine analoge Konstruktion für  $m = 2, 3$  das erwünschte Resultat hervor gebracht hat, versagt eine solche für höhere  $m$ , was für  $m = 5, \dots, 16$  überprüft wurde. Für  $m = 5$  konnte sogar trotz exhaustiver Suche *überhaupt* keine binäre MDS- $m$ -Matrix gefunden werden, was die Nutzbarkeit dieser Konstruktion für allgemeines  $m$  ernsthaft in Frage stellt.

Zusammenfassend ist zum Einsatz von T-Funktionen in Blockchiffren also anzumerken, dass sich eine vollständig auf T-Funktionen basierende Blockchiffre erst dann anbieten wird, wenn aus ihnen große bijektive S-Boxen mit kleinen dimensionsabhängigen differentiellen und linearen Potenzialen konstruiert werden können, was für keine der betrachteten Konstruktionsmethoden der Fall war. Der Verfasser bekennt aufrichtig, sich anfangs von den „wild“ und „nichtlinear“ aussehenden T-Funktionen wesentlich bessere Konfusionsseigenschaften erhofft zu haben.

In der Diffusionsebene machen die MDS-T-Funktionen und – wo anwendbar – insbesondere die MDS- $m$ -T-Funktionen jedoch eine gute Figur. Da diese allerdings besonders gut mit großen S-Boxen harmonieren, wäre es denkbar, jene mittels anderen Entwurfsprinzipien zu konstruieren und mit einer T-Funktionsbasierten Diffusionsebene zu kombinieren.

## 5.2 Ausblick

Die Untersuchung von T-Funktionen und deren Eignung für Konfusion und Diffusion hat eine Reihe interessanter Fragestellungen und weiterführender Probleme aufgeworfen, deren Behandlung den Rahmen dieser Diplomarbeit überstiegen hätte. Diese sollen im Folgenden kurz angesprochen werden.

**Invertierbare T-Funktionen mit linearen Bit-Slices** Das lineare Potenzial solcher T-Funktionen konnte nicht allgemein, sondern nur für bis zu zwei Variablen für allgemeines  $n$  als  $\Lambda_f = 1$  bestimmt werden. Während die experimentellen Resultate bei kleinem  $n$  auch für  $k > 2$   $\Lambda_f = 1$  nahelegen, wäre zum allgemeinen Nachweis eine Verallgemeinerung der auf linearen Approximationen der LSBs basierenden Beweistechnik erforderlich.

Darüber hinaus wurde das differentielle Potenzial solcher Funktionen nur experimentell als  $\Omega_f = 1$  berechnet, was die Vermutung nahelegt, dass dies allgemein gilt. Zwar verhindert das bewiesenermaßen schlechte lineare Potenzial unabhängig davon deren Einsatz als S-Box, es wäre jedoch interessant, diese Vermutung für allgemeines  $k$  und  $n$  zu verifizieren oder zu widerlegen.

**ANF-basierte T-Funktionen mit nichtlinearen Bit-Slices** Diese Funktionsklasse wurde zwar für allgemeines  $n$  und  $k$ , jedoch nur in ihrer „Reinform“, d.h. ohne additive Parameter analysiert. Die Experimente legen nahe, dass jene die differentiellen und linearen Potenziale nicht beeinflussen, ein mathematischer Nachweis dieser Beobachtung steht jedoch aus.

Darüber hinaus stellt sich die Frage, ob zu Proposition 4.7 und Lemma 4.9 vergleichbare Aussagen, welche das Differenzen- und Linearitätsprofil gesamter T-Funktionen auf dasjenige ihrer Bit-Slices herunterbrechen, für weitere Klassen von Funktionen als hier betrachtet möglich sind.

**Bessere differentielle und lineare Potenziale für große  $n$**  Alle betrachteten invertierbaren T-Funktionen haben entweder enttäuschende differentielle und lineare Potenziale oder solche, welche nicht von der Wortbreite  $n$ , sondern nur von der Variablenanzahl  $k$  abhängen. Es ist eine interessante offene Frage, eine solche invertierbare T-Funktion zu finden, welche qualitativ in die Nähe der Potenz- oder Inversionsabbildungen endlicher Körper kommt.

**MDS- $m$ -Abbildungen** Die Konstruktion von MDS- $m$ -Abbildungen aus binären  $m \times m$ -Matrizen mit Branch Number  $m$  hat nur für  $m \leq 4$  Erfolg gezeitigt und ist für  $m = 5$  als unmöglich nachgewiesen worden. Dies wirft die Fragestellung auf, für welche  $m > 5$  solche Matrizen existieren, und ob sich bei Existenz eine Konstruktionsvorschrift für solche Abbildungen entwickeln lässt.

**Kryptoanalyse realistisch dimensionierter Blockchiffren** Wie bereits in den Abschnitten 4.5.2 und 4.6.2 erwähnt, mussten die exemplarischen Entwürfe je eines T-Funktionsbasierten Feistel-Netzes und eines SP-Netzwerks zwecks analytischer Handhabbarkeit mit künstlich kleinen Parametern arbeiten. Die ausführliche Kryptoanalyse realistisch dimensionierter T-Funktionsbasierter Blockchiffren (vgl. Tabellen 4.11 und 4.12) bleibt damit offen.

# Literaturverzeichnis

- [1] E. Biham, A. Shamir: Differential Cryptanalysis of DES-like Cryptosystems. *Journal of Cryptology* Vol. 4(1), 1991, S. 3–72.
- [2] E. Biham: On Matsui’s Linear Cryptanalysis. *Proc. EUROCRYPT ’94*, LNCS 950, S. 341–355.
- [3] E. Biham: New types of cryptanalytic attacks using related keys. *Proc. EUROCRYPT ’93*, LNCS 765, S. 398–409.
- [4] E. Biham: Comment on Selecting the Ciphers for the AES Second Round. *Public Comments on AES Candidate Algorithms - Round 1*, 1999.
- [5] J. Buchmann: *Einführung in die Kryptografie. 2., erweiterte Auflage*, Springer 2001.
- [6] F. Chabaud, S. Vaudenay: Links between differential and linear cryptanalysis. *Proc. EUROCRYPT ’94*, LNCS 950, S. 356–365.
- [7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: *Introduction to Algorithms. 2nd Ed.*, MIT Press 2001.
- [8] J. Daemen: Cipher and hash function design strategies based on linear and differential cryptanalysis. *Dissertation*, Katholieke Universiteit Leuven 1995.
- [9] J. Daemen, V. Rijmen: *The block cipher Rijndael. Smart Card Research and Applications*, LNCS 1820, Springer 2000.
- [10] G. Fischer: *Lineare Algebra. 12., verbesserte Auflage*, Vieweg 2000.
- [11] S. Goldwasser, M. Bellare: *Lecture Notes on Cryptography. Skriptum*, Massachusetts Institute of Technology und University of California San Diego, 2001.
- [12] K. C. Gupta, P. Sarkar: Computing Partial Walsh Transform from the Algebraic Normal Form of a Boolean Function. *Technical Report CRG/2002/08*, Indian Statistical Institute 2002.
- [13] J. Kelsey, B. Schneier, D. Wagner: Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and triple-DES. *CRYPTO ’96*, LNCS 1109, S. 237–251.
- [14] A. Klimov: *Applications of T-functions in Cryptography. Dissertation*, Weizmann Institute of Science 2005.

- [15] A. Klimov, A. Shamir: A New Class of Invertible Mappings. Workshop on Cryptographic Hardware and Embedded Systems (CHES) 2002.
- [16] A. Klimov, A. Shamir: Cryptographic Applications of T-functions. Selected Areas in Cryptography (SAC) 2003.
- [17] A. Klimov, A. Shamir: New Cryptographic Primitives Based on Multiword T-functions. Fast Software Encryption (FSE) 2004.
- [18] A. Klimov, A. Shamir: New Cryptographic Applications of T-functions in Block Ciphers and Hash Functions. Fast Software Encryption (FSE) 2005.
- [19] L. R. Knudsen: Block Ciphers – Analysis, Design and Applications. Dissertation, Universität Aarhus 1994.
- [20] L. R. Knudsen: Block Ciphers — A Survey. COSIC'97 Course, LNCS 1529, S. 18-48, Springer 1998.
- [21] D. E. Knuth: The Art of Computer Programming, Vol. 1, Fundamental Algorithms. 3rd Ed., Addison-Wesley 1997.
- [22] D. E. Knuth: The Art of Computer Programming, Vol. 2, Seminumerical Algorithms. 3rd Ed., Addison-Wesley 1998.
- [23] X. Lai, J. Massey: A Proposal for a New Block Encryption Standard. Proc. EUROCRYPT '90, LNCS 473, Springer 1990.
- [24] X. Lai, J. Massey: Markov Ciphers and Differential Cryptanalysis. Proc. EUROCRYPT '91, LNCS 547, Springer 1991.
- [25] R. Lidl, H. Niederreiter: Introduction to finite fields and their applications. Revised Edition, Cambridge University Press, Cambridge (UK) 1994.
- [26] H. Lipmaa, P. Rogaway, D. Wagner: CTR-mode encryption. First NIST Workshop on Modes of Operation, 2000.
- [27] M. Matsui: Linear Cryptanalysis Method for DES Cipher. Proc. EUROCRYPT '93, LNCS 765, S. 386–397.
- [28] M. Matsui: On a structure of block ciphers with provable security against differential and linear cryptanalysis. IEICE Trans. Fundamentals E82-A (1), 1999, S. 117–122.
- [29] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone: Handbook of Applied Cryptography. CRC Press 1997.
- [30] K. Pommerening: Linearitätsmaße für Boolesche Abbildungen. Skriptum, Universität Mainz 2005.

- [31] V. Rijmen: Cryptanalysis and design of iterated block ciphers. Dissertation, Katholieke Universiteit Leuven 1997.
- [32] V. Rijmen, B. Preneel: On Weaknesses of non-surjective round functions. Proc. Workshop on Selected Areas in Cryptography (SAC), Ottawa 1995, S. 100–106.
- [33] R. L. Rivest, M. J. B. Robshaw, R. Sidney, Y. L. Yin: The RC6 Block Cipher. v1.1, August 20, 1998. Available at <http://www.rsalabs.com/rc6/>.
- [34] R. L. Rivest: Permutation Polynomials modulo  $2^w$ . In: Finite Fields and their Applications, Vol. 7, 2001.