Bachelorarbeit
im Fachgebiet Kryptographie und Computeralgebra

Sommersemester 2006

Fachgebiet Kryptographie und Computeralgebra

Fachbereich Informatik

TU Darmstadt

Algebraische Angriffe auf LFSR basierte Stromchiffren

Özgür Dagdelen

Betreuer: Kai Wirt

5. Oktober 2006

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Erklärung an Eides Statt

Hiermit versichere ich, die vorliegende Bachelorarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 5. Oktober 2006

Zusammenfassung

Aufgrund der kostengünstigen Implementierung in Hardware finden linear rückgekoppelte Schieberegister (LFSR) in vielen Applikationen Verwendung. Durch ihre Linearität sind sie jedoch anfällig für algebraische Angriffe. In dieser Bachelorarbeit werden LFSR-basierte Stromchiffren behandelt. Dabei werden einerseits LFSR vorgestellt, deren Ausgänge an eine nicht lineare Funktion weitergegeben werden. Diese Funktion gibt dann den Chiffretext aus. Andererseits werden auch unregelmäßig getaktete LFSR betrachtet. Hierbei dient die Ausgabe eines LFSR als Takt für die des zweiten LFSR. Die verschiedenen Designs für LFSRs werden verglichen mit Fokus auf die Sicherheit gegen mögliche Angriffe.

Durch die verschiedenen Implementierungsarten der LFSR werden verschiedene algebraische Angriffe angesetzt, um den Initialisierungsvektor der Verschlüsselung, den Schlüssel, aufzuspüren. Im Allgemeinen wird durch Kenntnis der Schlüsseltexte ein multivariates polynomiales Gleichungssystem gewonnen. Diese gilt es dann zu lösen. Es ergeben sich verschiedene Ansätze für Angriffe, je nach vorhandenem Wissen über die verwendeten Verschlüsslungsverfahren.

Es werden mehrere Algorithmen und Angriffe in dieser Arbeit vorgestellt. Unter diesen befindet sich die Linearisierungsmethode, in der diese erhaltenen nicht linearen Gleichungen in ein lineares Gleichungssystem umgewandelt werden. Aufgrund dessen, dass mit dieser Methode viele Schlüsseltexte nötig sind, werden andere Algorithmen bevorzugt, die wesentlich weniger Schlüsseltexte benötigen, aber möglicherweise auf Kosten von Rechenaufwand realisierbar sind. Zum einem gibt es die XL Methode, die durch Multiplikationen an den Gleichungen neue Gleichungen hinzugewinnt, welche linear unabhängig zu den vorherigen sind. Dann werden diese Gleichungen linearisiert und gelöst. Zum anderen existieren Gröbner basierende Algorithmen, die statt Multiplikationen an den Gleichungen vorzunehmen, die Gleichungen geeignet kombinieren. In dieser Arbeit werden diese algebraischen Angriffe auf momentan verwendete LFSR angesetzt und deren Komplexität verglichen.

Aufgrund der verschiedenen Angriffe und Sicherheitslücken vieler LFSR werden zuletzt Designansätze für LFSR beschrieben, die eine optimale Sicherheit gegen algebraische Angriffe und andere schon bekannte Angriffe garantieren soll.

1 Einleitung

Es existieren bereits etliche Angriffe auf Stromchiffren, welche linear rückgekoppelte Schieberegister verwenden. Doch seit neuerem gehören auch algebraische Angriffe zu diesen. Sie sind bisher die effizientesten unter diesen und es Bedarf an Analyse, in wie weit diese Angriffe in realen Systemen eine Gefahr ausüben können.

Die meisten verwendeten Stromchiffren wurden entworfen, als die algebraischen Angriffe noch nicht bekannt waren. Es stellt sich nun heraus, dass einige in Verwendung sich befindende Stromchiffren empfindlich gegen Angriffe solcher Art sind. Es treten neue Schwächen und Angriffspunkte auf. Das Ziel dieser Arbeit ist es diese Schwächen aufzuzeigen und die Vorgehensweise algebraischer Angriffe zu erläutern. Durch tiefere Analyse sollen auch weiter Designvorschläge präsentiert und anfällige existierende Stromchiffren benannt werden.

Diese Arbeit ist wie folgt aufgebaut: In Kapitel 2 werden zunächst die linear rückgekoppelten Schieberegister vorgestellt. Dabei werden die üblichen Designs solcher Stromchiffren beschrieben und verglichen. Weiter werden im Kapitel 3 Vorgehenstechniken zum Lösen von nichtlinearen Gleichungssystemen präsentiert. Darunter befindet sich die Linearisierungsmethode, der XL Algorithmus und die Algorithmen, welche mit Gröbner Basen arbeiten. Diese Techniken werden vor allem im Kapitel 4 benötigt, wenn algebraische Angriffe auf LFSR beschrieben werden. Hier wird erläutert, wie die Gleichungssysteme je nach vorliegendem Design der Stromchiffre gewonnen werden. Dabei liegt ein Teil des Fokus auf der Komplexität und dem benötigtem Wissen für einen erfolgreichen Angriff. Zuletzt werden Gegenmaßnahmen gegen algebraische Angriffe genannt. Dabei taucht der Begriff „Algebraische Immunität“ auf. Der Sinn und Zweck dieser Größe wird erläutert und es werden allgemeine Designvorschläge gebracht.

2 Linear rückgekoppelte Schieberegister (LFSR)

Wenn Bedarf besteht Nachrichten durch einen abhörbaren Kanal zu versenden, dann wird die Nachricht meist verschlüsselt, um die Vertraulichkeit der Nachricht zu gewährleisten. Stromchiffren bieten eine Möglichkeit Nachrichten zu verschlüsseln.

Eine Stromchiffre ist die symmetrische, kontinuierliche und verzögerungsfreie Ver- oder Entschlüsselung eines Datenstroms [Wik06]. Dabei ver- und entschlüsselt eine Stromchiffre Bit für Bit und braucht nicht wie eine Blockchiffre auf genügend viele Bits zu warten bis die gewünschte Blockgröße erreicht ist. Zum anderen haben Stromchiffren eine niedrigere Komplexität in Hardware. Somit sind Stromchiffren eine oft schnellere Variante gegenüber Blockchiffren vor allem bei der Implementierung in Hardware.

Stromchiffren sind typischer Weise in 2 Kategorien unterteilt. Zum einen gibt es die synchronen Stromchiffren, bei denen die Ausgabe der Stromchiffren, also der Schlüsselstrom, nicht von dem Klartext, der verschlüsselt werden soll, abhängt. Zum anderen haben wir asynchrone Stromchiffren, welche für ihre Ausgabe auf Klartextbits oder schon bereits ausgegebenen Bits aus dem Schlüsselstrom zurückgreifen. In dieser Bachelorarbeit gehen wir nicht weiter auf asynchrone Stromchiffren ein, sondern beschränken uns auf synchrone Stromchiffren.

Stromchiffren besitzen einen internen Zustand, durch welchen ihre Ausgabe bestimmt wird. Dabei wird die Ausgabe der Stromchiffre z_t mit dem t-ten Bit der Eingabe m_t verknüpft. Dies kann beispielsweise durch eine übliche Addition geschehen, aber in der Praxis werden vor allem binär additive Stromchiffren benutzt. In diesen Stromchiffren ist die Verknüpfung durch eine XOR-Verknüpfung gebeben. Das somit erhaltene Zeichen ist das t-te Bit der Chiffretextes c_t. Der interne Zustand wird nach jeder Ausgabe eines Bits verändert. Dabei ist das Geheimnis bei der Stromchiffre nicht die Zustandsänderungsfunktion, sondern der Initialisierungszustand. Der Initialisierungszustand wird als Schlüssel bezeichnet. Durch diesen Initialisierungszustand lassen sich alle folgenden Zustände durch Hintereinanderausführung der Zustandsänderungsfunktion nach jedem Bit berechnen. Auch kann der Klartext aus dem Wissen des verwendeten Schlüssels und des Schlüsselstroms gewonnen werden. Da die XOR-Verküpfung zu sich selbst invers ist, gilt: $c_t \oplus z_t = m_t$. Aus diesem Grund gilt es die Geheimhaltung des verwendeten Schlüssels in der Stromchiffre zu sichern.

Eine beliebte Komponente einer Stromchiffre sind die linear rückgekoppelten Schieberegister (LFSR), womit ein Schlüsselstrom erzeugt wird. Dabei finden sie vor allem Verwendung wegen der schon erwähnten kostengünstigen Implementierung in Hardware sowie ihrer einfachen mathematischen Analysierbarkeit [Wik06]. Eine andere Verwendung dieser LFSR findet man in der Generierung von Zufallszahlen.

LFSR-basierte Stromchiffren besitzen den internen Zustand $S_t = (s_t, s_{t+1}, \ldots, s_{t+n-1}) \in \{0,1\}^n$, worauf in jedem Schritt t eine lineare Funktion $L : \{0,1\}^n \to \{0,1\}$ ausgeführt wird. Diese Funktion nimmt sich meist $k \leq n$ Bits aus dem internen Zustand und führt eine XOR Verknüpfung mit diesen aus. Der somit erhaltene Funktionswert $s_{t+n} = L(S_t)$ wird in das Register eingeschoben. Am anderen Ende wird ein Ausgabebit s_t erhalten.

Um die Arbeitsweise eines LFSR deutlicher zu machen, folgt hier ein einfaches Beispiel.

Beispiel 1 *Gegeben sind folgende Werte:*

- *Initialisierungszustand* $S_0 = \{s_0, s_1, s_2, s_3\} = \{1, 0, 1, 1\}$
- *Feedback-Funktion* $L := s_0 \oplus s_3$

Um nun den Schlüsseltext zu berechnen, geht man wie folgt vor:
Man berechnet durch die Feedback-Funktion die neue Eingabe in das Register:
$s_4 = L(S_0) = 1 \oplus 1 = 0$
Als Ausgabe erhalten wir s_0*, also eine* 1*. Führt man nun weitere Schritte aus, bekommt man die folgenden Ausgaben.*

Schritt t	s_{t+3}	s_{t+2}	s_{t+1}	s_t	Ausgabe
0	1	1	0	1	-
1	0	1	1	0	1
2	0	0	1	1	0
3	1	0	0	1	1
4	0	1	0	0	1
5	0	0	1	0	0
6	0	0	0	1	0
7	1	0	0	0	1
8	1	1	0	0	0
9	1	1	1	0	0
10	1	1	1	1	0
11	0	1	1	1	1
12	1	0	1	1	1
13	0	1	0	1	1
14	1	0	1	0	1
15	1	1	0	1	0

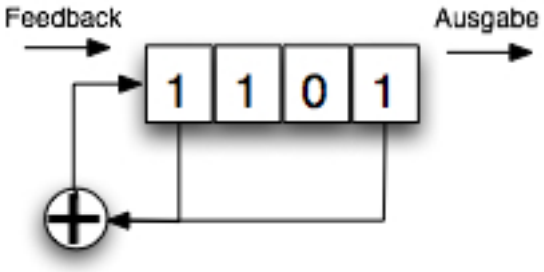

Abbildung 1: LFSR aus Beispiel 1

Wir sehen, dass $S_{15} = S_0$*. Somit erkennen wir, dass das LFSR aus diesem Beispiel eine Periodenlänge von 15 besitzt. Der Ausgabestrom lautet* 101100100011110*.*

Zu linear rückgekoppelte Schieberegister lassen sich einige statische Eigenschaften zeigen. Zum einem kann die maximale Periodenlänge leicht berechnet werden. Die maximale Periodenlänge eines LFSR beträgt $2^n - 1$, wobei n die Anzahl der Bits im Register ist, welche seinen Zustand angeben. Doch um die maximale Periodenlänge in einem LFSR zu erreichen, muss die Feedback-Funktion L gewisse Vorraussetzungen erfüllen.
Sei die Funktion $g : \{0,1\} \rightarrow \{0,1\}$ gegeben mit:

$$g := a_0 + a_1 * x + a_2 * x^2 + \ldots + a_{n-1} * x^{n-1} + x^n$$
$$\text{mit } a_i = \begin{cases} 1 & \text{falls } s_i \text{ ein Summand von } L \\ 0 & \text{sonst} \end{cases}$$
$$\text{für } i \in \mathbb{N}$$

Diese Funktion g wird auch oft als charakteristisches Polynom der Feedback-Funktion L bezeichnet.

Ein LFSR besitzt genau dann die maximale Periodenlänge, wenn die Polynomfunktion g, wie oben gebildet, primitiv ist[Nyb05]. Dabei gilt, dass ein Polynom primitiv ist, wenn es nicht in kleinere Polynome faktorisiert werden kann, also irreduzibel ist, und $deg(g) \geq 1$ ist [Ker02]. Das ist nicht die konkrete Definition von primitiven Polynomen, aber sie reicht für unsere Zwecke aus.

Im Beispiel 1 haben wir die Feedback-Funktion $L := s_0 \oplus s_3$ mit $n = 4$ genutzt. Mit diesen Werten erhalten wir für die Funktion $g := 1 + x^3 + x^4$. Tatsächlich ist dieses

Polynom nicht weiter faktorisierbar. Somit erreicht dieses LFSR aus unserem Beispiel die maximale Periodenlänge $2^n - 1 = 2^4 - 1 = 15$.

Es befinden sich noch weitere statistische Eigenschaften in maximalen LFSR:

- Der Wert 1 im Ausgabestrom taucht 2^{n-1} mal auf, der Wert 0 genau $2^{n-1} - 1$.
- Der Ausgabestrom beinhaltet ein String aus 1en der Länge n, einen String aus 0en der Länge $n - 1$.
- Es gibt in etwa gleich viele Paare, Trippel etc. aus 0en und 1en

Dass diese Eigenschaften für unser kleines Beispiel gelten, kann man leicht erkennen.

Die Einfachheit einer Implementierung der LFSR ist durch ihre Funktionsweise zwar gegeben, doch durch eben die linearen Abhängigkeiten der LFSR sind sie anfällig für algebraische Angriffe. So kann ein Angreifer durch beliebige n Schlüsselbits den Initialisierungszustand S_0 errechnen und wüsste somit den geheimen Schlüssel.

Um die Linearität der LFSR zu beseitigen, gibt es verschiedene Ansätze, wie man die LFSR geeignet designt und mit anderen Komponenten verbindet. Dadurch wird es schwieriger an den geheimen Schlüssel zu kommen. In den folgenden Unterkapiteln werden die verschiedenen Designmöglichkeiten vorgestellt, welche in späteren Kapitel auf ihre Sicherheit hinsichtlich algebraischer Angriffe geprüft werden.

2.1 LFSR mit nichtlinearem Filtergenerator

Bei diesem Ansatz versucht man an ein linear rückgekoppeltes Schieberegister einen nichtlinearen Filtergenerator anzuschließen, der mehrere Bits aus dem LFSR als Eingabe hat und durch etwaige Multiplikationen eine Nichtlinearität im Schlüsselstrom zu erreichen versucht.

Sei ein LFSR gegeben mit internem Zustand $S^{(t)} = (s_1^{(t)}, \ldots, s_l^{(t)})$ zum Zeitpunkt t. Das verwendete LFSR hat somit eine Länge von l Bit. Den Initialisierungszustand $S^{(0)}$ nennt man auch den Schlüssel dieser Stromchiffre. Er muss geheim bleiben.

Zudem sei eine boolesche Funktion $f(x_1, \ldots, x_n)$ gegeben, welche zur Berechnung des Schlüsselstrombits $z_{(t)}$ benötigt wird. Zum Zeitpunkt t erhalten wir die Ausgabe des Schlüsselstromgenerators $z^{(t)}$ durch folgendes Vorgehen:

$$z^{(t)} = f(s_{i_1}^{(t)}, \ldots, s_{i_n}^{(t)}) \text{ für alle } t \geq 0 \tag{1}$$

wobei $i_1, \ldots, i_n \in \{1, \ldots, l\}$ disjunkt sind. Es werden also $n \leq l$ Bits aus dem Zustand $S^{(t)}$ des LFSR zum Zeitpunkt t genommen und durch die nichtlineare boolesche Funktion f die Ausgabe $z^{(t)}$ berechnet.

Dadurch, dass ein LFSR nach jedem Schritt t seinen Zustand durch Shiften verändert, kann es vorkommen, dass ein Bit aus dem LFSR mehrmals abgefragt wird. Wenn man dies genau betrachtet, erkennt man schnell, dass jedes Zustandsbit s_i im LFSR ab dem Zeitpunkt n, also wenn das LFSR einmal komplett durchlaufen ist und kein Bit aus der Initialisierung mehr vorhanden ist, genau l mal in einer Zeitspanne n aufgerufen wird. Das mehrmalige Nutzen eines Zustandsbits wurde schnell als Gefahr erkannt. Diese bestehende Gefahr wurde bereits durch Korrelationsangriffe bestätigt, so dass diese Art von Stromchiffren in realen Systemen nicht genutzt werden sollten. Zumindest sollten sie das nur eingeschränkt.

Hier wird nun ein einfaches Beispiel für eine Stromchiffre mit einem nichtlinearen Filtergenerator vorgestellt. Es soll diese Art von Stromchiffre verdeutlichen:

Beispiel 2 *Der Schlüsselstromgenerator besteht aus einem 4-Bit LFSR , welche mit der folgenden booleschen Funktion verknüpft sind:*

$$f(s_0, s_2) = s_0 * s_2 \tag{2}$$

Dabei befindet sich der interne Zustand des LFSR $S^{(t)} = (s_1^{(t)}, s_2^{(t)}, s_3^{(t)}, s_4^{(t)})$ *im Schritt* t*. Er wird Initialisiert mit* $(s_1^{(0)}, s_2^{(0)}, s_3^{(0)}, s_4^{(0)}) = (1, 1, 1, 1)$*, welcher auch unser 4-Bit Schlüssel ist.*
Zum anderen haben wir die Zustandsüberganfunktion L *des LFSR gegeben mit:*

$$s^{(t+1)} = L(s^{(t)}) = s_1^{(t)} \oplus s_4^{(t)} \tag{3}$$

Wir bekommen somit folgende Ausgaben und Zustände im jeweiligen Schritt:

Schritt t	s_{t+3}	s_{t+2}	s_{t+1}	s_t	Ausgabe
0	1	1	1	1	1
1	0	1	1	1	1
2	1	0	1	1	0
3	0	1	0	1	1
4	1	0	1	0	0
5	1	1	0	1	0
6	0	1	1	0	1
7	0	0	1	1	0
8	1	0	0	1	0
9	0	1	0	0	1
10	0	0	1	0	0
11	0	0	0	1	0
12	1	0	0	0	0
13	1	1	0	0	0
14	1	1	1	0	0
15	1	1	1	1	-

Wir erhalten wie im Beispiel 1 *eine Periodenlänge des LFSR von 15. Als Ausgabestrom gibt uns dieser Schlüsselstromgenerator* 110100100100000*.*

Beispiel 3 *Ein anderes Beispiel aus der Praxis stammt aus Japan. Die Stromchiffre wird Toyocrypt[CM03] genannt. Sie besitzt ein 128-Bit LFSR. Somit gilt* $n = 128$*. Die nichtlineare boolesche Funktion* f *in Toyocrypt ist wie folgt definiert:*

$$f(s_0, \ldots, s_{127}) = s_{127} + \sum_{i=1}^{n} s_i s_{\alpha_i} + s_{10}s_{23}s_{32}s_{42} + \\ + s_1 s_2 s_9 s_{12} s_{18} s_{20} s_{25} s_{26} s_{28} s_{33} s_{38} s_{41} s_{42} s_{51} s_{53} s_{59} + \prod_{i=0}^{62} s_i \tag{4}$$

wobei $\{\alpha_0, \ldots, \alpha_{62}\}$ *eine Permutation aus der Menge* $\{63, \ldots, 125\}$ *ist.*

2.2 Mehrere LFSR mit nichtlinearem Filtergenerator und Combinergenerator

Hier existiert wieder eine Unterteilung zwischen speicherlosen und mit Speicher behafteten Stromchiffren. In den folgenden Unterkapiteln sind beide Arten vorgestellt und beschrieben.

2.2.1 Nutzen eines Combiners ohne zusätzlichen Speicher

Eine Variante die Problematik der Linearität in den Schlüsselstrombits zu meistern, hat den Ansatz, dass die Ausgänge mehrerer LFSR als Eingänge einer nicht-linearen booleschen Funktion $f : \{0,1\}^n \rightarrow \{0,1\}$ dienen [SC0X, Sar02]. Das Modell ist in Abbildung 2 dargestellt. Die internen Zustände der n LFSR $S_1, \ldots, S_n$ bestehen jeweils aus $l_1, \ldots, l_n$ Bits. Dabei besitzt das i-te LFSR im Zeitpunkt t den internen Zustand $S_i^{(t)} = (s_{i,1}^{(t)}, \ldots s_{i,l_i}^{(t)})$ für $i \in 1, \ldots, n$. Der Schlüsselstrom $Z = z^{(0)}z^{(1)} \ldots z^{(k)}$ wird mit dem Klartext $M = m_0 m_1 \ldots m_k$ XOR-verknüpft und man erhält den Chiffretext $C = c_0 c_1 \ldots c_k$. Für das Schlüsselstrombit z gilt daher:

$$z^{(t)} = f(s_{1,1}^{(t)}, s_{2,1}^{(t)}, \ldots, s_{n,1}^{(t)}) \text{ für } t \geq 0. \tag{5}$$

Die Entschlüsselung funktioniert analog, in dem man $M = C \oplus Z$ rechnet.
Der Schlüssel dieser Stromchiffre sind die internen Zustände der einzelnen LFSR $S_1^{(0)}, S_2^{(0)}, \ldots S_n^{(0)}$. Das bedeutet, dass der Schlüssel eine Länge $l = l_1 + l_2 + \ldots + l_n$ besitzt.

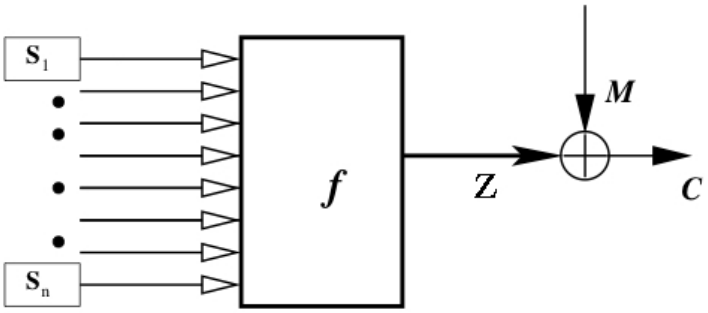

Abbildung 2: Modell: n LFSR mit nichtlinearer booleschen Funktion

Die verwendete boolesche Funktion $f(x_1, \ldots, x_n)$ kann dabei als ein multivariates Polynom gesehen werden. Dieses Polynom kann durch eine Summe von Produkten, in welche jede disjunkte Auswahl von den n Funktionsvariablen dargestellt sind, ausgedrückt werden. Mathematisch ausgedrückt bildet sich die Funktion f wie folgt:

$$f(x_1, \ldots, x_n) = a_0 \oplus (\oplus_{i=1}^{i=n} a_i * x_i) \oplus (\oplus_{1 \leq i=j \leq n} a_{ij} * x_i * x_j \oplus \ldots \oplus a_{12\ldots n} * x_1 \ldots x_n)$$
$$\text{mit } a_0, a_i, a_{ij}, \ldots, a_{12\ldots n} \in \{0,1\} \tag{6}$$

Betrachtet man die Summanden der Funktion als Terme und zählt die Anzahl der Multiplikationen in allen Termen, so erhält man den Grad der Funktion durch die maximale Anzahl von Multiplikationen in allen Termen. Als Beispiel sei die Funktion $f := x_o + x_1 x_3 x_6$ gegeben. Hier besitzt die Funktion f den Grad 3 ($deg(f) = 3$).

Ein praktisches Beispiel für ein System, welches mehrere lineare rückgekoppelte Schieberegister mit einer nichtlinearen booleschen Funtion verknüpft nutzt, ist der Geffe Generator [Luc05, Arm].

Beispiel 4 *Der Geffe Generator besteht aus 3 LFSR, welche mit der folgenden booleschen Funktion verknüpft sind:*

$$f(a_t, b_t, c_t) = \begin{cases} a_t & falls\ c_t = 0 \\ b_t & falls\ c_t = 1 \end{cases} \tag{7}$$

Dabei sind a_t, b_t, c_t die Ausgänge der linear rückgekoppelten Schieberegister zum Zeitpunkt t. Die nichtlineare boolesche Funktion des Geffe Generators, wie sie in (7) dargestellt ist, lässt sich auch umschreiben als:

$$f(a_t, b_t, c_t) = a_t \oplus a_t c_t \oplus b_t c_t \tag{8}$$

Hier können wir leicht herauslesen, dass die verwendete Funktion f des Geffe Generator den Grad 2 besitzt.

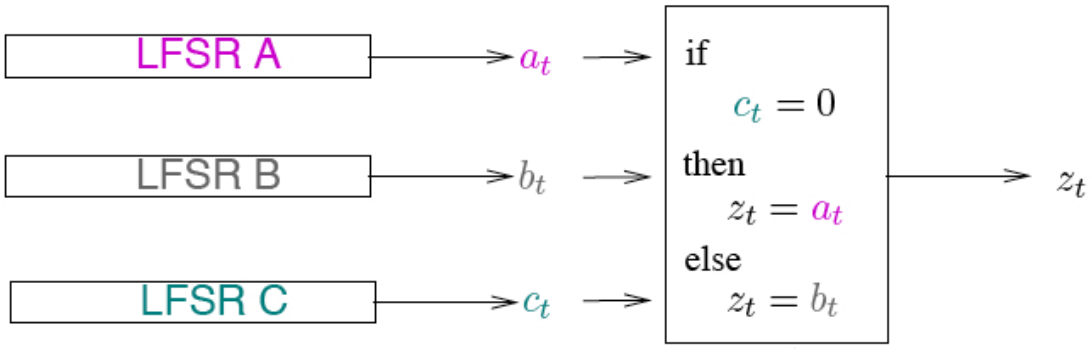

Abbildung 3: Geffe - Generator

2.2.2 Nutzen eines Combiners mit zusätzlichen Speicher

Ein anderer Ansatz eine Stromchiffre sicher zu machen ist, noch zusätzlich zu einer nichtlinearen booleschen Funktion F ein Speicher mit einem Combinergenerator C zu nutzen [AK03, Arm04a]. Das bedeutet, dass noch zusätzlich ein Speicher genutzt wird, welcher ein Folgezustand durch den Combinergenerator erhält und deren Ausgänge die Ausgabe des Schlüsselstromgenerators beeinflussen.

Seien $n \geq 1$ LFSR gegeben mit den internen Zuständen $S_i^{(t)} \in \{0,1\}^{l_i}$ zur Zeit t, wobei l_i die Anzahl der Bits ist, welche den internen Zustand von $S_i^{(t)}$ besitzen. Jedes dieser LFSR besitzt eine lineare boolesche Funktion L_i, welche das Eingangsbit beim Shiften bestimmt.

Das komplette System besitzt den Zustand $S^{(t)} = (S_1^{(t)}, \ldots . S_n^{(t)}) \in \{0,1\}^l$ mit $l = l_1 + \ldots + l_n$. Die zusätzlich verwendeten Bits aus dem Speicher $c_t \in \{0,1\}^k$ dienen zusammen mit den Ausgängen $s_t := (s_1^{(t)}, \ldots, s_n^{(t)})$ der LFSR als Eingang für die Funktion f. Dabei gilt $c_{t+1} := C(s_t, c_t)$. Der Initialisierungszustand $S^{(0)}$ ist dabei das Geheimnis und ist zusammen mit der Initialisierungsbelegung des Speichers c_0 der Schlüssel des Systems.

So ein Schlüsselstromgenerator wird (n,k)-Combiner genannt, wobei n die Anzahl der genutzten LFSR ist und der Wert k angibt, wieviele Bits aus dem Speicher zusätzlich genutzt werden. Wenn keine zusätzlichen Bits aus einem Speicher genutzt werden, kann auch von einem $(n,0)$-Combiner gesprochen werden. Anschaulich ist diese Art der Stromchiffre in der Abbildung 4 zu finden.

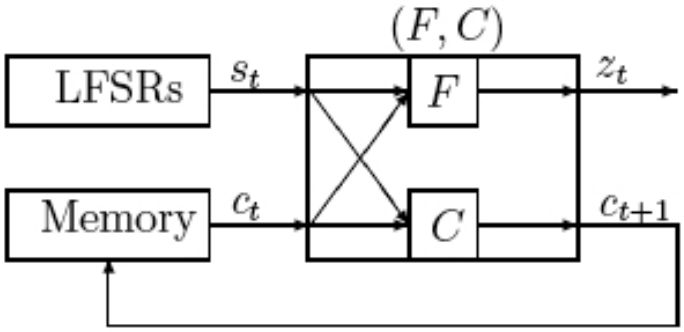

Abbildung 4: Ein (n,k)-Combiner

Die Funktionweise dieses Schlüsselstromgenerators kann man wie folgt festlegen:
Das System führt im Schritt t folgende Aktionen aus:

1. Ausgabe $z_t = f(s_t, c_t)$
2. Aktualisierung des internen Zustandes $S^{(t)}$ zu
   $S^{(t+1)}$ mit $S^{(t+1)} := L(S^{(t)}) = (L_1(S_1^{(t)}), \ldots, L_n(S_n^{(t)}))$
3. Aktualisierung der verwendeten Speichers: $c_{t+1} = C(s_t, c_t)$

Eine bekannte Stromchiffre, welche nach diesem Typ aufgebaut ist, nennt sich E_0 [FL01]. Der E_0 Schlüsselstromgenerator ist Teil des Bluetooth Verschlüsselungssystems, welches für eine wireless Verbindung genutzt wird.

Beispiel 5 *Schlüsselstromgenerator E_0:*
Der Schlüsselstromgenerator E_0 besteht aus 4 LFSR mit einer Gesamtlänge von 128 Bits. Die einzelnen LFSR haben jeweils ein Bitlänge von $l_1 = 25, l_2 = 31, l_3 = 33$ und $l_4 = 39$. Die Feedback-Funktionen der LFSR lauten wie folgt [Gmb02]:

- $f_1(x) = x^{25} + x^{20} + x^{12} + x^8 + 1$
- $f_2(x) = x^{31} + x^{24} + x^{16} + x^{12} + 1$
- $f_3(x) = x^{33} + x^{28} + x^{24} + x^4 + 1$
- $f_4(x) = x^{39} + x^{36} + x^{28} + x^4 + 1$

Zusätzlich wird ein Speicher aus 4 Bits mit $c_t \in \{0, 1\}$ genutzt. Dieser Generator besitzt somit eine Schlüssellänge von $l_1 + l_2 + l_3 + l_4 = 128$. Der Generator ist in der Abbildung 5 abgebildet.

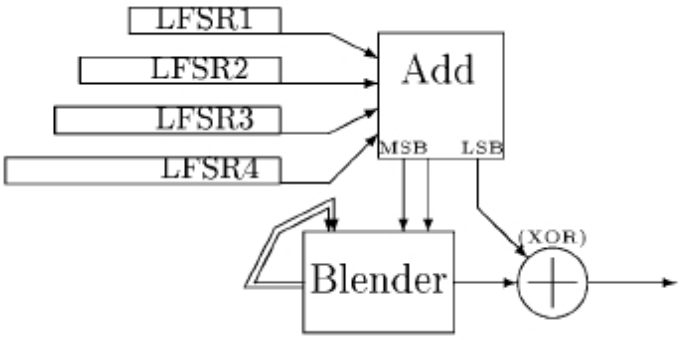

Abbildung 5: Das Modell des Schlüsselstromgenerators E_0

2.3 Unregelmäßig getaktete LFSR

Eine ganz andere Idee Nichtlinearität in den Schlüsselstrombits zu erlangen, sind unregelmäßig getaktete LFSR. Die Idee ist es hier einen Takt an die Register zu setzen, wodurch Unregelmäßigkeiten bei den Ausgaben erreicht werden sollen, um damit komplexere und hohe nichtlineare Ausgaben der Schlüsselstromgeneratoren zu erzeugen. Dabei ist die eigentliche Idee durch LFSRs anderen LFSRs den Takt angeben zu lassen, womit bestimmt wird, ob normal geshiftet werden soll, oder anders als üblich reagiert werden soll.

Diese Art von Schlüsselstromgeneratoren ist nennenswert, denn sie kommt häufig in der Praxis zum Einsatz. Beispielweise finden sie am Ende dieses Kapitels den Schlüsselstromgenerator A5/1 [Can03], welcher für die Verschlüsselung im GSM Standard gebraucht wird. Andere nennenswerte Generatoren sind shrinking generator, self-shrinking generator, LILI-128 und LILI-II.

Der einfachste Fall bei unregelmäßig getakteten LFSR ist, wenn die Ausgabe eines LFSR als Takt für ein zweites dient. Das zweite LFSR nimmt dann einen neuen Zustand ein, wenn die Ausgabe des ersten eine 1 war. Andernfalls wird die vorherige Ausgabe wiederholt ausgegeben. Diese Variante ist der einfachste Fall der "Stop and Go" Variante [Rob95]. Eine andere Art, welche auch zur "Stop and Go" Variante gehört, is es, das LFSR unregelmäßig zu takten, so dass das zweite LFSR doppelt schaltet, falls eine 1 ausgegeben wird. Andernfalls schaltet es einmal. Diese Variante hat statistisch gesehen bessere Ausgaben im Schlüsselstrom und ist daher der vorherigen vorzuziehen. Doch dies setzt natürlich voraus, dass das zweite LFSR eine ausreichend große Periodenlänge besitzt.

Eine andere Variante, die als eine Erweiterung des "Stop and Go" angesehen werden kann, ist die Kaskade Variante [Rob95]. Die Idee in diesem Modell ist die, dass ein LFSR den Takt für ein zweites LFSR gibt, welches wiederum den Takt für ein drittes LFSR gibt, usw. Diese Methode verspricht lange Perioden und gute statistische Eigenschaften.

Allgemein lassen sich unregelmäßig getaktete LFSR in zwei Teile aufteilen. Zum einen besitzen solche Schlüsselstromgeneratoren eine Komponente, welche den Takt angibt für etwaige Aktionen. Nennen wir diese Komponente "Clock Control"[EJ05]. Dieser Teil, wie auch immer er aufgebaut ist, wird eine endgültige Ausgabe c_t der anderen Komponente des Systems überreichen. Nennen wir diese Komponente "Data generation". Diese Komponente heißt aus diesem Grund so, weil sie die endgültige Ausgabe des Schlüsselstrom-

generators liefert, nachdem sie das Taktsignal c_t bekommen hat. Siehe auch Abbildung 6 zur Verdeutlichung.

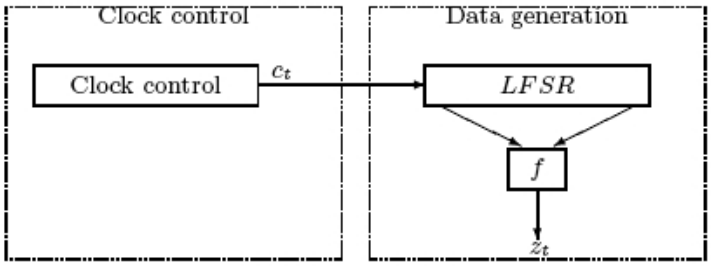

Abbildung 6: Ein allgemeines Modell eines irregulär getakteter Filtergenerators

Der Daten generierende Teil der Stromchiffre ist meist ein Filtergenerator, so wie in Kapitel 2.1 vorgestellt.

Beispiel 6 *A5/1 ist eine symmetrische Stromchiffre, welche bei der Verschlüsselung in GSM verwendet wird. Es existiert eine leichte Abänderung dieser Stromchiffre mit der Bezeichnung A5/2. Diese ist jedoch ein wenig anfälliger gegenüber Angriffen. A5/1 wird in den meisten Ländern Europas genutzt, wobei A5/2 in den Restlichen Verwendet findet.*

Die Stromchiffre A5/1 verwendet 3 LFSR der Länge $l_1 = 19, l_2 = 22, l_3 = 23$. *Die charakteristischen Polynome für die verwendeten LFSR sind die folgende:*

$$g_1(x) = x^{19} + x^5 + x^2 + x + 1 \quad (9)$$
$$g_2(x) = x^{22} + x + 1 \quad (10)$$
$$g_3(x) = x^{23} + x^{15} + x^2 + x + 1 \quad (11)$$

Der Initialisierungszustand des Generators besteht aus einen 64-Bit geheimen Schlüssel K und aus einer öffentlichen zugänglichen 22-Bit Framenummer F.

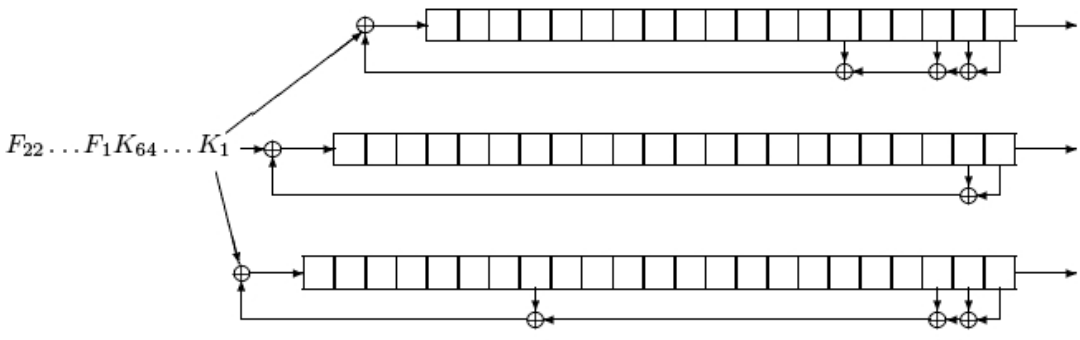

Abbildung 7: Die Initialisierung des A5/1 Schlüsselstromgenerators

Die Funktionsweise von A5/1 ist dabei wie folgt:

1. *Zum Zeitpunkt 0 sind alle LFSR mit 0 initialisiert.*
2. *Zum Zeitpunkt $t \in \{1, \ldots, 64\}$ wird das t-te Bit des Schlüssel K_t mit den Feedback-Funktionen der LFSR XOR-verknüpft.*
3. *Zum Zeitpunkt $t \in \{65, \ldots, 86\}$ wird das $(t-64)$-te Bit der Framenummer F_{t-64} mit der Feedback-Funktionen der LFSR XOR-verknüpft.*

Das weitere Vorgehen ab dem Zeitpunkt $t = 87$ ist dann unregelmäßig getaktet. Jeweils 1 Bit aus dem internen Zustand des LFSR, welches "clocking tap" genannt wird, wird gelesen und wie beim Wählen werden diese Bits in einer Urne "b" gesammelt. Nun wird überprüft, welches Bit die Mehrheit besitzt. Beispielsweise stünde im Speicher $b = (1, 0, 0)$. Dann haben wir eine Mehrheit für 0. Alle LFSR, die im clocking tap den Bitwert der Mehrheit besitzen schalten weiter. Die übrigen behalten ihren internen Zustand. Danach wird wie üblich die Ausgabe berechnet.

Die "clocking taps" der in A5/1 verwendeten LFSR sind das 8. Bit im ersten LFSR und das 10. Bit in den anderen beiden.

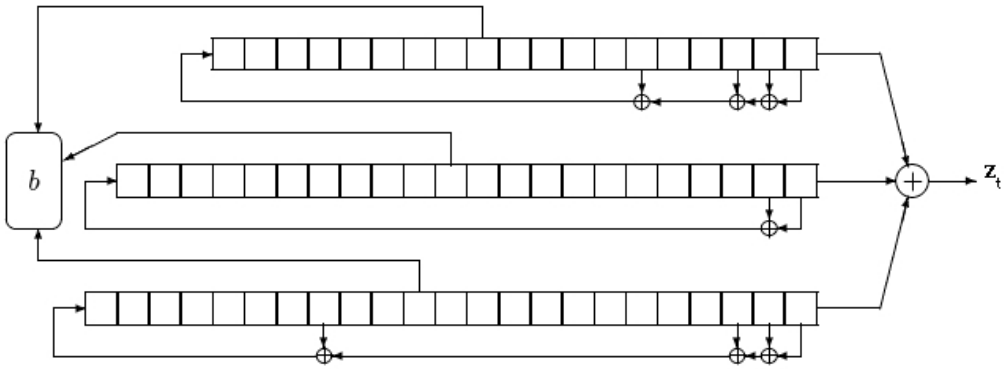

Abbildung 8: A5/1 Schlüsselstromgenerator nach der Initialisierung

2.4 Vergleich der vorgestellten Designs

Wenn man Designs vergleichen möchte, so muss man dieses nach gewissen Kriterien tun. Einige Kriterien werden in den Kapitel 4 und 5 dargestellt. Vor allem sicherheitsbezogene Unterschiede und Gemeinsamkeiten können in diesen Kapitel gefunden werden.

Zu den Gemeinsamkeiten und Unterschieden können folgende Punkte zusammengefasst werden:
Alle 3 vorgestellten Modelle besitzen eine boolesche Funktion f, damit sie mit Hilfe dieser die Nichtlinearität im Schlüsselstrom erhalten. In Filtergeneratoren dienen aber nur Bits aus dem internen Zustand eines LFSR als Eingang der booleschen Funktion. In Combiner nutzt man mehrere LFSR und auch nur die Ausgabe dieser LFSR. Man liest nicht direkt aus dem internen Zustand des LFSR. Irregulär getaktete Stromchiffren bieten bereits durch die asynchrone Taktung die gewünschte Unregelmäßigkeit in den Ausgaben.

Aber da dies noch nicht ausreichend Sicherheit gibt, sollte eine nichtlineare Filterfunktion verwendet werden. Es können auch mehrere LFSR, wie man es in der Stromchiffre A5/1 findet, zum Einsatz kommen.

Es ist möglich Filtergeneratoren durch äquivalente Combiner zu ersetzen, also den selben Schlüsselstrom zu erzeugen. Wenn die boolesche Funktion f des Filtergenerators $l \leq n$ Bits aus dem internen Zustand des LFSR als Eingang nimmt, so wird man für den äquivalenten Combiner l LFSR mit der jeweiligen Länge des LFSR aus dem Filtergenerator verwenden. Die Zustandsänderungsfunktionen werden beim Combiner übernommen. Es existiert dann eine Initialisierung der l LFSR , so dass die Ausgaben beider Generatoren die gleichen sind. Das soll nochmal an einem kleinen Beispiel verdeutlicht werden.

Beispiel 7 *Sei ein Filtergenerator gegeben mit folgenden Werten:*

- *Das LFSR hat den internen Zustand* $S^{(t)} = (s_0^{(t)}, s_1^{(t)}, s_2^{(t)})$ *zum Zeitpunkt* t*.*
- *Zustandsänderungsfunktion* $L(s_0, s_1, s_2) := s_2 \oplus s_0$
- *Die boolesche Funktion* $f(s_0, s_1) = s_0 * s_1$*.*
- *Initialisierungszustand* $S^{(0)} = (1, 1, 0)$

Sei ein Combiner gegeben mit folgenden Werten:

- *2 LFSR mit internen Zuständen* $S_i^{(t)} = (s_{i,0}^{(t)}, s_{i,1}^{(t)}, s_{i,2}^{(t)})$ *zum Zeitpunkt* t *mit* $i \in \{1, 2\}$
- *Zustandsänderungsfunktion* $L_i(s_0, s_1, s_2) := s_2 \oplus s_0$ *für* $i \in \{0, 1\}$
- *Die boolesche Funktion f ist gegeben mit* $f(s_0, s_1) = s_0 * s_1$*.*
- *Initialisierungszustand* $S_1^{(0)} = (1, 1, 0)$ *und* $S_2^{(0)} = (0, 1, 0)$

Dann erhalten wir für die Ausgabe des Filtergeneratoren: 0100001
Die Ausgabe des Combiners lautet: 0100001

Nach dieser Theorie kann man nun sagen, dass die Combiner eine Art Erweiterung der Filtergeneratoren sind.

Alle 3 Typen der vorgestellten Stromchiffren sind schnell in Hardware zu implementieren und unterscheiden sich kaum bezüglich ihrer Komplexität im Aufbau.

Irregulär getaktete Stromchiffren passen nicht ganz in das Schema der anderen beiden, aber sie können wie bereits erwähnt einen Filtergenerator in der Datenerzeugungskomponente nutzen.

3 Algorithmen zum Lösen nichtlinearer Gleichungssysteme

Das effiziente Lösen mutivariater polynomieller Gleichungen ist in der Kryptographie von großem Interesse. Wir interessieren uns für das Lösen solcher Gleichungen, weil es einen Angreifer ermöglicht an den geheimen Schlüssel der Stromchiffre heranzukommen. Wie in Kapitel 2 bereits erwähnt werden Funktionen höheren Grades genutzt, um Nichtlinearität in den Schlüsselstrombits zu erreichen. Wenn wir nun mehrere Schlüsselstrombits haben und die dazugehörige Funktion f kennen, können wir durch Lösen der nichtlinearen Gleichungen die Werte der Variablen und somit den internen Zustand der LFSR ermitteln. Das bedeutet, dass der geheime Schlüssel somit bestimmt werden kann.

Aus der Kryptographie ist bekannt, dass ein System allgemein als sicher eingestuft wird, wenn ein Angreifer exponentiell viele Operationen bezüglich der Problemgröße (hier: Anzahl der Variablen) benötigt, um an das Geheimnis zu kommen. Dieses sollte durch multivariate polynomielle Gleichungen gegeben sein, denn das Lösen dieser Gleichungen ist NP-hart. Selbst wenn alle Polynome höchstens 2. Grades sind und der Definitions- und Wertebereich in {0,1} liegt, ist es dennoch NP-hart. So ist hier die Motivation gegeben Algorithmen mit möglichst geringer Komplexität zu finden, welche dieses Problem zu lösen versuchen.

In der Praxis tritt häufiger der Fall auf, dass man mehr Variablen als Gleichungen hat. Das erschwert das Lösen dieser Gleichungen und natürlich ist die Eindeutigkeit der Lösung somit nicht mehr garantiert. In wie weit die vorgestellten Algorithmen dieses Problem zu meistern versuchen, wird in den folgenden Unterkapiteln vorgestellt. Zum Schluss dieses Kapitels finden sie ein Vergleich dieser Algorithmen, wobei der Fokus auf der Komplexität und den Voraussetzungen der Algorithmen liegt.

3.1 Linearisierung

3.1.1 Idee

Die Idee bei der Linearisierungsmethode ist es, das nichtlineare Gleichungssystem in ein lineares Gleichungssystem umzuwandeln [Arm]. Danach können die bekannten linearen Techniken genutzt werden, um das entstehende lineare Gleichungssystem zu lösen [CL05]. Dabei muss beachtet werden, dass bei der Linearisierung nur linear unabhängige Gleichungen entstehen. Da keine eindeutige Lösung des Gleichungssystem erwartet werden kann, wird gehofft, dass die Anzahl möglicher Lösungen niedrig bleibt.

Mit der Lösung des linearen Gleichungssystem wird dann die Lösung des nichtlinearen Gleichungssytems gewonnen.

3.1.2 Funktionsweise

In einem nichtlinearen Gleichungssystem sind in den Gleichungen Produkte von Variablen enthalten wie beispielsweise bei der Gleichung $x * y = 0$. In linearen Gleichungssystemen haben wir Kombinationen von Variablen, aber sie sind nicht miteinander multipliziert, sondern additiv verknüpft. So könnte eine lineare Gleichung die Form $x + y + z = 1$ besitzen. Das bedeutet also, dass die Terme in linearen Gleichungen alle den Grad 1 besitzen.

Um also ein nichtlineares Gleichungssystem in ein lineares Gleichungssystem umzuwandeln, geht die Linearisierung wie folgt vor: Alle Produkte in einem Term werden umbenannt in eine neue unabhängige Variable. Diese Einführung neuer Variablen wird in allen Termen und Gleichungen ausgeführt. Somit werden alle Produkte zu einer Variablen und die dadurch entstandenen neuen Gleichungen sind alle linear.

Nachdem nun ein lineares Gleichungsystem entstanden ist, können die uns bekannten Methoden aus der linearen Algebra benutzt werden, wie beispielsweise der Gauss-Algorithmus. Doch zuvor muss beachtet werden, dass linear abhängige Gleichungen entfernt werden, da sonst nicht eindeutige Lösungen entstehen können.

Es können beim Lösen des linearen Gleichungssystems mehrere mögliche Lösungen gefunden werden. Nun muss jede Lösung überprüft werden, ob sie auch das nichtlineare Gleichungssystem, welches eigentlich zu lösen war, lösen kann.

Um die Funktionsweise der Linearisierung zu verdeutlichen, folgt nun ein kleines Beispiel.

Beispiel 8 *Das zu lösende nichtlineare Gleichungssystem hat folgende Gestalt:*

$$\begin{aligned} x_a \oplus x_b \oplus x_a * x_b &= 1 \\ x_b \oplus x_a * x_b &= 1 \end{aligned}$$

Nun müssen alle auftretenden Produkte mit einer neuen Variable ersetzt werden. In dem Gleichungssystem haben wir ein Produkt $x_a * x_b$. Wir definieren eine neue Variable $y_{ab} := x_a * x_b$. Wir erhalten damit folgendes lineares Gleichungssystem:

$$\begin{aligned} x_a \oplus x_b \oplus y_{ab} &= 1 \\ x_b \oplus y_{ab} &= 1 \end{aligned}$$

Nach Anwendung des Gauss-Algorithmus erhalten wir folgendes Gleichungssystem:

$$\begin{aligned} x_a &= 0 \\ x_b \oplus y_{ab} &= 1 \end{aligned}$$

Nun sieht man, dass genau 2 Lösungen existieren, die dieses lineare Gleichungssystem lösen:

$$\begin{aligned} x_a = 0, \quad x_b = 0, \quad y_{ab} = 1 \\ x_a = 0, \quad x_b = 1, \quad y_{ab} = 0 \end{aligned}$$

Nun muss überprüft werden, ob diese Lösungen auch das nichtlineare Gleichungssystem lösen. Wenn wir die erste Lösung verwenden, finden wir ein Widerspruch, so dass diese Lösung nicht brauchbar ist.

$$\begin{aligned} 0 &= x_a \\ 0 &= x_a \\ 1 &= y_{ab} = x_a * x_b \end{aligned}$$

Wenn wir die 2. Lösung einsetzen, bekommen wir keinen Widerspruch und sehen, dass wir damit die Lösung des nichtlinearen Gleichungssystem haben. Damit wurde dieses nichtlineare Gleichungssystem mit der Linearisierung gelöst.

3.1.3 Voraussetzungen

Es ist wichtig unter welchen Voraussetzungen ein positives Ergebnis mit der Linearisierung zu erwarten ist. Die erste Voraussetzung ist die, dass genügend linear unabhängige Gleichungen entstehen, wenn wir das nichtlineare Gleichungssystem in ein lineares umwandeln.

Es sei ein nichtlineares Gleichungssystem gegeben mit maximalem Grad d in allen Termen. So kann die maximale Anzahl der Variablen berechnet werden, die man erhält, wenn das nichtlineare Gleichungssystem in ein lineares Gleichungssystem umgewandelt wird. Ein Term, welches ein Produkt aus Variablen ist, wird Monom genannt. Die maximale Anzahl an verschiedenen Monomen, die in den Gleichungen auftreten können, ist gegeben durch:

$$M(n,d) := \binom{n}{0} + \ldots + \binom{n}{d} \tag{12}$$

wobei n die Anzahl der Variablen im nichtlinearen Gleichungssystem ist und d der maximale in allen Termen vorkommende Grad [Arm04a]. Die Linearisierung wandelt das nichtlineare Gleichungssystem in ein lineares Gleichungssystem, indem alle Monome als eine neue Variable behandelt werden. Sei $m := M(n,d)$, dann erhalten wir nach dem Linearisierungsschritt ein lineares Gleichungssystem mit m unbekannten Variablen. Somit kann festgelegt werden, dass bei der Linearisierung im ungünstigsten Fall $M(n,d) < n^d$ Gleichungen benötigt werden. Diese Anzahl ist im Vergleich zu den noch bevorstehenden Methoden höher, aber ist dennoch polynomiell bei fester Anzahl von d.

Ansonsten, wenn genügend linear unabhängige Gleichungen vorhanden sind, können die Standardalgorithmen der linearen Algebra oder der Numerik genutzt werden, um die Lösung zu bestimmen. Daher sind keine weiteren Vorraussetzung für die Nutzung dieser Methode vorhanden.

3.1.4 Komplexität

Mit dieser Methode ist es tatsächlich möglich, dass ein nichtlineares Gleichungssystem in polynomieller Zeit gelöst wird. Wie bereits in Abschnitt 3.1.3 festgestellt, erhalten wir im ungünstigsten Fall ein Gleichungssystem mit $\sim n^d$ Variablen, wobei n die Anzahl der Variablen im Original Gleichungssystem ist und d der maximale Grad aller vorkommenden Terme. Das bedeutet, dass die Anzahl der auftretenden Variablen und damit die maximal benötigten Gleichungen polynomiell in n und d sind.

Nun muss ein Algorithmus zum Lösen des linearen Gleichungssystems benutzt werden. Ein Standardalgorithmus ist der von Gauss. Dieser benötigt $\sim n^3$ Operationen für die Lösung eines linearen Gleichungssystem. Damit haben wir für die komplette Linearisierung eine Laufzeit von $O(n^{3d})$. Die Laufzeit ist für festes n und d polynomiell und damit gilt dieses Verfahren als schnell.

Natürlich können auch andere Verfahren zum Lösen des linearen Gleichungssystem statt dem Gauss-Algorithmus genutzt werden, die eventuell schneller das Gleichungssystem lösen können. So kann man eine etwas kürze Laufzeit erlangen.

Diese Laufzeit bezieht sich nur auf den ungünstigsten Fall. Des öfteren sind nicht alle möglichen Monome in den Gleichungen vorhanden, so dass die Anzahl der Variablen im linearen Gleichungssystem geringer ist und somit mit geringerer Laufzeit eine Lösung gefunden werden kann.

3.1.5 Vorteile & Nachteile

Die Linearisierungsmethode hat einen besonderen Vorteil, dass sie die Lösung in polynomieller Zeit angeben kann. Wie in Abschnitt der Komplexität erwähnt, erhalten wir eine Laufzeit von $O(n^{3d})$ mit der Problemgröße n, welche der Anzahl der Variablen im nichtlinearen Gleichungssystem entspricht, und d, die den maximalen Grad aller in den Gleichungen vorkommenden Terme angibt. Wie wir später erkennen, ist das eine recht schnelle Methode verglichen mit den anderen Methoden, die später vorgestellt werden.

Die Linearisierung hat jedoch einen kleinen aber bedeutsamen Nachteil. Sie benötigt im ungünstigen Fall $\binom{n}{0} + \ldots + \binom{n}{d} < n^d$ Gleichungen, damit das Gleichungssystem gelöst werden kann. Dies ist eine beträchliche Summe, die meist nicht gegeben ist. Meistens liegt so ein weit überdefiniertes Gleichungssystem nicht vor.

In praktischen Fällen, vor allem in algebraischen Angriffen, wird versucht mit wenig Kenntnissen und kurzer Laufzeit die Lösung eines Gleichungssystems zu erhalten. Mit dieser Methode erlangt man nur eine kurze Laufzeit, wenn genügend Gleichungen vorhanden sind. Da dies selten in algebraischen Angriffen erreichbar ist, ist diese Methode eher als unbrauchbar anzusehen.

3.2 XL Algorithmus

Im Jahre 2000 wurde der XL-Algorithmus von Nicolas Courtois, Alexander Klimov, Jacques Patarin und Adi Shamir eingeführt [CKPS00]. XL steht für "eXtended Linearization" und kann als eine Erweiterung der Linearisierung angesehen werden.

3.2.1 Idee

Die Idee beim XL-Algorithmus ist ähnlich der Linearisierung. Auch hier wird versucht, dass nichtlineare Gleichungssystem in ein lineares Gleichungssystem zu überführen. Der Unterschied ist aber, dass beim XL-Algorithmus deutlich weniger nichtlineare Gleichungen vorhanden sein müssen, um die Belegung der Variablen herauszufinden.

Das Problem bei einer zu geringen Anzahl an Gleichungen im nichtlinearen Gleichungssystem führt dazu, dass wir dann im entsprechenden linearen Gleichungssystem möglicherweise zu wenig linear unabhängige Gleichungen gegenüber den Variablen haben, die zu lösen sind. So hat der XL-Algorithmus einen anderen Ansatz als die Linearisierung. Im XL-Algorithmus bereitet man sich zuerst auf die Linearisierung vor, indem zuvor neue nichtlineare Gleichungen hergeleitet werden. Dies kann erreicht werden, indem an die vorhandenen Gleichungen Kombinationen von Variablen dazu multipliziert werden. Damit entstehen neue möglicherweise unabhängige Gleichungen, welche vor und nach der Linearisierung zu prüfen sind.

Durch diesen Ansatz werden deutlich weniger Gleichungen benötigt, um eine Lösung für die Belegung der Variablen zu erlangen, aber er nutzt trotzdem die Schnelligkeit der Linearisierung mit.

3.2.2 Funktionsweise

Bevor die eigentliche Funktionsweise des XL-Algorithmus vorgestellt wird, sind einige Definitionen einzuführen.

- Der maximale Grad, welcher in allem Termen jeder Gleichung vorkommt, wird als K bezeichnet.
- Sei $D \in \mathbb{N}$ die Eingabe des XL-Algorithmus sein. $\mathcal{I}_\mathcal{D}$ ist dabei die Menge, welche alle Gleichungen vom Grad $\leq D$ beinhaltet.
- Das nichtlineare Gleichungssystem liegt in Form $l_i(x_1, \ldots, x_n) = 0$ vor mit $i \in \{1, \ldots, m\}$. Die Anzahl der Gleichungen ist m und n ist die Anzahl der Variablen.

Die Funktionsweise des XL-Algorithmus lässt sich in 4 Schritten zusammenfassen:

1. **Multipliziere:** Generiere alle Produkte $\prod_{j=1}^{k} x_{i_j} * l_i \in \mathcal{I}_\mathcal{D}$ mit $k \leq D - K$.
2. **Linearisiere:** Sehe jedes Monom in x_i mit Grad $\leq D$ als eine neue Variable und löse das Gleichungssystem mit Hilfe des Gauss-Algorithmus. Dabei müssen die Monome so geordnet werden, dass alle Terme, welche eine Variable (sagen wir x_1) enthalten, zum Schluss eliminiert werden.
3. **Löse:** Nehmen wir an. dass Schritt 2 mindestens eine Gleichung beinhaltet, welche nur x_1 mit verschieden Exponenten beinhaltet. Löse diese Gleichung über eine endliche Menge (z.B. mit dem Berlekamp Algorithmus)
4. **Wiederhole:** Vereinfache die Gleichungen und wiederhole den Prozess, um die Werte der anderen Variablen mit diesem Schema herauszufinden.

Der XL-Algorithmus ist im eigentlichen Sinn zum Lösen quadratischer multivariater Gleichungen vorgesehen gewesen. Wir haben hier aber eine kleine Abänderung der Definition von [CKPS00] im ersten Schritt vorgenommen und sind so der Definition vom [Cou02] nachgegangen. Somit kann nun der XL-Algorithmus mit dieser Definition auch Gleichungen mit höherem Grad als $K = 2$ lösen. Zum anderen benötigt man keinen Wiederholungsschritt bei [Cou02], weil die anderen Variablen eigentlich auch mit dem linearen System, welches im 2. Schritt erhalten wird, berechnet werden können. Es ist nicht weiter nötig, wieder ein neues lineares Gleichungssystem aufzubauen für jeweils eine Variable. Wir haben aber dennoch diesen Schritt hier aufgeführt, denn sie gehört zur eigentlichen Definition des XL-Algorithmus.

Im Schritt 1 werden durch Multiplikationen neue nichtlineare Gleichungen hergeleitet, die linear unabhängig zu den bisherigen sind. Dabei wählen wir eine Anzahl an Polynomen vom Grad D-K, welche an alle Gleichungen multipliziert werden.

Sei $K = 2$ und $D = 4$. Zum anderen besteht das nichtlineare Gleichungssystem aus 3 Variablen (x, y, z). Dann bekommen wir die folgenden Polynome zum Multiplizieren: $(x, y, z, x^2, y^2, z^2, x{*}y, x{*}z, y{*}z)$. Je höher D ist, desto mehr Gleichungen können gewonnen werden, aber eine Erhöhung sollte mit Vorsicht begangen werden. Durch Erhöhung des Wertes D entstehen auch viele zu bestimmende Variablen im Linearisierungschritt 2.

Im Linearisierungsschritt werde analog zur Linearisierungsmethode alle auftretenden Monome durch eine neue Variable ersetzt. Somit erhalten wir ein lineares Gleichungssystem. Dabei bestimmt D das Verhältnis von Variablen und Gleichungen, die im linearen Gleichungssystem zu erwarten sind. Im folgenden Unterkapitel wird näher darauf eingegangen, welches D man bevorzugt wählen sollte.

Äquivalent zur Linearisierung soll auch hier das lineare Gleichungssystem mit dem Gauss Eliminationsverfahren genutzt werden, welche das lineare Gleichungssystem in

Dreiecksgestalt umformen soll. Dabei soll eine gewisse Reihenfolge beachtet werden, in der die Variablen auftreten. Eine auserwählte Variable und ihre Vorkommnisse in allen Termen sollen zum Schluss eliminiert werden, daher nutzt man das Kommutativgesetz der Addition und verschiebt die Terme mit dieser Variable. Dies hat den Sinn, dass man möglicherweise eine Gleichung finden kann, welche nur diese eine Variable beim Umformen enthält, so dass wir dann Schritt 3 anwenden können.

Im Schritt 3 wird dann verlangt, dass mindestens eine Gleichung nach dem Linearisierungsschritt vorhanden ist, womit wir dann weiterarbeiten. Diese Gleichung soll dann nach der Variable, die wir in der Linearisierung ausgesucht haben, aufgelöst werden. Dazu empfiehlt sich der Berlekamp Algorithmus.

Im letzten Schritt wird der Wert, welchen man für diese Variable im 3. Schritt erhalten hat, in das original nichtlineare Gleichungssystem eingesetzt. Dadurch wird ein neues Gleichungssystem mit n-1 Variablen gewonnen. Auf dieses neue nichtlineare Gleichungssystem wird dann wieder der Algorithmus durchlaufen. In späteren Veröffentlichungen des Erfinders dieses Algorithmus sei dieser Schritt nicht mehr nötig, denn es ist ausreichend auf dem vorhandenen linearen Gleichungssystem weiterzuarbeiten, um die Werte der anderen Variablen auszurechnen.

Nun folgt ein Beispiel, in dem ein nichtlineares Gleichungssystem mit dem XL-Algorithmus gelöst wird.

Beispiel 9 *Es ist folgendes zu lösende nichtlineare Gleichungssystem gegeben:*

$$\begin{array}{lll} (i) & x_1 + x_1x_2 & = & 1 \\ (ii) & x_2 + x_1x_2 & = & 0 \end{array}$$

Aus diesem Gleichungssystem erhalten wir folgende Werte für die Parameter des Algorithmus:
Das Gleichungssystem besitzt 2 Gleichungen $\rightarrow m = 2$.
Das Gleichungssystem besitzt 2 Unbekannte $\rightarrow n = 2$.
Der höchste vorkommende Grad in allen Termen der Gleichung lautet $2 \rightarrow K = 2$.
Wir wählen als weiteres Input des Algorithmus $D = 4$.

<u>Schritt 1:</u> Wir finden heraus, dass wir nur Polynome vom Grad 2 für die Multiplikation verwenden dürfen, denn es gilt: $D - K = 2$. *Somit haben wir als zulässige Polynome* $(x_1, x_2, x_1^2, x_2^2, x_1x_2)$. *Durch Multiplikation der Polynome an unsere Gleichung erhalten wir* $2 * 5 = 10$ *Gleichungen:*

$$\begin{array}{lllll} (i_a) & x_1^2 & + & x_1^2x_2 & = & x_1 \\ (i_b) & x_1x_2 & + & x_1x_2^2 & = & x_2 \\ (i_c) & x_1^3 & + & x_1^3x_2 & = & x_1^2 \\ (i_d) & x_1x_2^2 & + & x_1x_2^3 & = & x_2^2 \\ (i_e) & x_1^2x_2 & + & x_1^2x_2^2 & = & x_1x_2 \\ (ii_a) & x_1x_2 & + & x_1^2x_2 & = & 0 \\ (ii_b) & x_2^2 & + & x_1x_2^2 & = & 0 \\ (ii_c) & x_1^2x_2 & + & x_1^3x_2 & = & 0 \\ (ii_d) & x_2^3 & + & x_1x_2^3 & = & 0 \\ (ii_e) & x_1x_2^2 & + & x_1^2x_2^2 & = & 0 \end{array}$$

Schritt 2: In diesem Schritt ersetzen wir die Monome durch neue Variablen und erhalten folgendes Gleichungssystem:

$$\begin{array}{lccccc}
(i_a) & b & + & h & = & a \\
(i_b) & g & + & i & = & d \\
(i_c) & c & + & k & = & b \\
(i_d) & i & + & l & = & e \\
(i_e) & h & + & j & = & g \\
(ii_a) & g & + & h & = & 0 \\
(ii_b) & e & + & i & = & 0 \\
(ii_c) & h & + & k & = & 0 \\
(ii_d) & f & + & l & = & 0 \\
(ii_e) & i & + & j & = & 0
\end{array}$$

mit $a := x_1$, $b := x_1^2$, $c := x_1^3$, $d := x_2$, $e := x_2^2$, $f := x_2^3$, $g := x_1x_2$, $h := x_1^2x_2$, $i := x_1x_2^2$, $j := x_1^2x_2^2$, $k := x_1^3x_2$ *und* $l := x_1x_2^3$.

Wir sehen, dass die Variablen a, b und c von x_1 *abhängen, so versuchen wir nach diesen Variablen umzuformen, um für den Schritt 3 eine Gleichung zur Verfügung zu haben, die nur von* x_1 *abhängt. Nun erhalten wir durch Elimination und Einsetzung folgende Gleichungen:*

Aus (i_a)*:* $h = a - b$
Aus (i_c)*:* $k = b - c$
Aus (ii_c)*:* $k = -h \quad \rightarrow \quad b - c = -(a - b) \quad \rightarrow \quad b - c = b - a \quad \rightarrow \quad c = a$
Wir erhalten für den Schritt 3 die Gleichung $c = a$.

Schritt 3: Die Gleichung $c = a$ *wird resubstituiert und wir erhalten:* $x_1^3 = x_1$.
Diese Gleichung ergibt $x_1 = 1$ *oder* $x_1 = 0$. *Je nach Definitionsbereich der Variable* x_1 *können noch andere Werte angenommen werden.*

Schritt 4: Hier würden die möglichen Werte für x_1 *eingesetzt werden, um somit* x_2 *zu bestimmen, aber dieses kleine Beispiel benötigt keine weiteren Wiederholungen dieses Algorithmus.*

Ein anderes etwas allgemeineres Beispiel befindet sich in [CKPS00].

3.2.3 Voraussetzungen

Es gibt zwei Punkte, an dem dieser Algorithmus möglicherweise scheitert und somit nicht genutzt werden kann. In Schritt 1 werden alle möglichen Monome von Grad $D - K$ an alle gegebenen Gleichungen multipliziert, um neue linear unabhängige Gleichungen zu gewinnen. Nun kann es passieren, dass wir nicht genügend linear unabhängige Gleichungen erstellen konnten, und der Gauss-Algorithmus nicht angewendet werden kann. Zum anderen brauchen wir in Schritt 3 eine Gleichung, bei der nur eine Variable mit allen ihren Potenzen bis zum Grad D auftaucht. Solch eine Gleichung kann mit dem Gauss-Eliminationsverfahren gewonnen werden, jedoch exisitieren dafür Voraussetzungen an der Anzahl der vorliegenden linear unabhängigen Gleichungen.

Wie erwähnt funktioniert der XL-Algorithmus so, dass alle möglichen Monome vom Grad $D-K$ an die m gegebenen Gleichungen l_i multipliziert werden. Wir erhalten somit den maximalen Grad aller Terme von D. Sei nun R die Anzahl der generierten Gleichungen durch XL und T die Anzahl aller möglichen Monome. Nach [Cou02] erhalten wir für R und T:

$$R = m * \left(\sum_{i=0}^{D-K} \binom{n}{i} \right) \approx m * \binom{n}{D-K} \tag{13}$$

$$T = \sum_{i=0}^{D} \binom{n}{i} \approx \binom{n}{D} \tag{14}$$

Nun muss R noch weiter untersucht werden, denn nicht komplett R kann verwendet werden. Einige Gleichungen sind linear abhängig von anderen. Sei nun *Free* die exakte Anzahl linear unabhängiger Gleichungen. Sicher ist, dass $Free \leq R$ sowie $Free \leq T$ gilt.

Nach [Cou02, CP03] kann der Gauss-Eliminationsverfahren eine Gleichung, welche nur eine Variable x_i enthält, gewinnen, wenn gilt: $Free \geq T - D$. Die Idee und Theorie ist, dass für gewisse D immer $R \geq T$ erfüllen lässt. Dann kann auch erwartet werden, dass der für den linear unabhängigen Teil von R, also *Free* gilt: $Free \approx T$. Somit muss D so gewählt werden, dass gilt $Free \geq T - D$, aber D sollte aus Komplexitätsgründen so gering wie möglich gewählt werden. Siehe dazu den folgenden Abschnitt.

3.2.4 Komplexität und Analyse

Die Komplexität des XL-Algorithmus besitzt den größten Faktor beim Anwenden des Gauss-Eliminationsverfahren im Schritt 2. Hier muss untersucht werden, wieviele Gleichungen durch den XL-Algorithmus generiert werden. Unter diesen Gleichungen wird auf den linear unabhängigen Teil, bezeichnet mit *Free*, der Gauss-Algorithmus angewendet. Dabei gilt $Free = \mu * R$, wobei μ den Faktor der linear unabhängigen Gleichungen aller generierten Gleichungen R bestimmt. Wir wissen bereits, dass gilt: $R \approx m * \binom{n}{D-K}$ und $T \approx \binom{n}{D}$. Nun erhalten wir nach der Bedingung $Free \geq T - D$ wie folgt:

$$\begin{array}{lccl}
 & Free & \geq & T - D \\
\iff & \mu * R & \geq & T - D \\
\iff & \mu * \left(m * \binom{n}{D-K} \right) & \geq & \binom{n}{D} - D \\
\iff & m & \geq & \dfrac{\binom{n}{D} - D}{\mu * \binom{n}{D-K}} \\
\iff & m & \geq & \dfrac{\dfrac{n!}{D!(n-D)!} - D}{\dfrac{\mu * n!}{(D-K)!(n-D+K)!}} \\
\iff & m & \geq & \dfrac{D!(n-D)!) - D}{\mu(D-K)!(n-D+K)!} \\
\iff & m & \geq & \dfrac{(n-D+K) * (n-D+K-1) * \cdots * (n-D+1)}{\mu * D * (D-1) * \cdots * (D-K+1)} - \lambda_{D,K}
\end{array}$$

mit $\lambda_{D,K} = \frac{D}{\mu(D-K)!(n-D+K)!}$. Unter Annahme, dass $D \ll n$ gilt, folgt:

$$m \geq \left(\frac{n}{\mu * D}\right)^K \tag{15}$$

Das Gauß-Eliminationsverfahren muss ein Gleichungssystem lösen mit der größt möglichen Anzahl an Monomen als Variablen, welches wir als T bezeichnet hatten. Mit (15) erhalten wir $D \approx \frac{n}{m^{1/K}}$. Somit kann T und damit die komplette Komplexität des Gauss-Eliminationsverfahren im Schritt 2 angegeben werden:

$$T^\omega \approx \binom{n}{D}^\omega \approx \binom{n}{n/m^{1/K}}^\omega \tag{16}$$

wobei ω den Exponenten des Gauss-Verfahrens angibt. Die Theorie gibt für $\omega \approx 2.376$ an. Eine alternative wäre der Strassen-Algorithmus, welcher eine Anzahl von $7 * T^{\log_2 7}$ Operationen benötigt, um eine Lösung zu liefern. Wenn nun m groß genug ist, können gute Werte bzgl. der Komplexität erreicht werden. In diesem Bereich ist man sich nicht einig, wie man m wählen sollte, aber unter Annahme $m = \epsilon n^K$ mit $\epsilon > 0$ erwartet man polynomielle Laufzeit. Nach [CKPS00] kann unter bestimmen Gegebenheiten eine subexponentielle Laufzeit erreicht werden. Dieses ist möglich, wenn $m \approx n$ gilt, aber $m-n$ trotzdem betragsmäßig groß ist.

Im Speziallfall, dass wir die Lösung eines nichtlinearen Gleichungssystem mit Definition und Wertebereich {0,1} suchen, setzen wir $K = 2$ und erhalten die gleichen Ergebnisse bzgl. der Komplexität. Wir erhalten aber den Vorteil, dass wir auch Lösungen erhalten können, also den XL-Algorithmus benutzen können, wenn $m = n$ vorliegt. Nach Simulationen und Beobachtungen wurde in [CP03] festgestellt, dass mit $D \ll 2^n$ solche gegebenen Gleichungssysteme gelöst werden können. Dieser Fall ist interessant bei algebraischen Angriffen, weil wir Gleichungen erhalten, bei denen Variablen die Zustände der LFSR darstellen. Diese sind Bits und nehmen nur Werte 0 und 1 an.

In [CP03] wurde mit Hilfe von Computersimulationen getestet, unter welchen Bedingungen XL bei m linear unabhängigen Gleichungen mit n unbekannten Variablen anwendbar ist. Dabei wurde die Eingabe D erhöht und beobachtet, welche statistischen Eigenschaften vorliegen und wie sie sich verändern. Alle Ergebnisse wurden mit quadratischen nichtlinearen Gleichungen in Körper {0,1} gewonnen.

n	10	10	10	10	10	20	20	20	20	64	64
m	10	14	16	17	18	20	40	60	65	512	1024
D	3	3	3	3	3	3	3	3	3	3	3
R	110	154	176	187	198	420	840	1260	1365	33280	66560
T	176	176	176	176	176	1351	1351	1351	1351	43745	43745
Free	110	154	174	175	175	420	840	1260	1350	33280	43744
μ	1.000	1.000	.9886	.9358	.8838	1.000	1.000	1.000	0.9890	1.000	.6572

Tabelle 1: Ergebnisse mit XL für $D = 3$

n	10	10	10	20	20	20	20	20	20	40
m	5	10	11	20	24	28	30	32	36	128
D	4	4	4	4	4	4	4	4	4	4
R	280	560	616	4220	5064	5908	6330	6752	7596	105088
T	386	386	386	6196	6196	6196	6196	6196	6196	102091
Free	265	385	385	4010	4764	5502	5865	6195	6195	96832
μ	.9464	.6875	.6250	.9502	.9408	.9313	.9265	.9175	0.8156	.9214

Tabelle 2: Ergebnisse mit XL für $D = 4$

Mit Hilfe der Abbildungen 1 und 2 lassen sich einige Aussagen und Behauptungen aufstellen: Der Fall, dass $m = n$ vorliegt, ist lösbar, wie wir es in der Abbildung 2 für $m = n = 10$ erkennen können. In [CP03] wurde gezeigt, dass auch $m = n = 20$ mit $D = 5$ gelöst wurde. Allgemein wurde damit die These aufgestellt, dass nichtlineare Gleichungen mit $m = n$ mit dem XL-Algorithmus mit $D \ll 2^n$ gelöst werden können.

Man erkennt an Hand der Abbildungen, dass der linear unabhängige Anteil μ von allen generierten Gleichungen R mit wachsendem m zwar abnimmt, aber der Wert *Free* absolut trotzdem zunimmt. Daher können bessere Ergebnisse mit mehr nichtlinearen Gleichungen als Input erreicht werden oder bei mangelnder Anzahl an Gleichungen m der XL-Algorihmus nicht angewendet werden. Denn wenn nicht $R \geq T$ gilt bzw. *Free* $\approx T$ kann keine Lösung mit dem Gauss-Eliminationsverfahren erwartet werden.

Häufig ist aber nicht der Fall gegeben, dass die Anzahl der Gleichungen erhöht werden kann, wenn der Bedarf besteht. In diesem Fall kann auch die Eingabe D erhöht werden, um ein positives Ergebnis mit dem XL-Algorithmus zu erreichen. In der Abbildung 1 kann abgelesen werden, dass bei $n = 20$ der XL-Algorithmus erst mit m=65 angewendet werden kann. Denn mit $m < 65$ können wir nicht die Gleichung $R \geq T$ erfüllen. Aber mit Erhöhung von D auf $D = 4$ lässt sich mit $m = 32$ die Gleichung erfüllen. Hieraus lässt sich schließen, dass durch Erhöhung von D alle nichtlinearen Gleichungen mit dem XL-Algorithmus gelöst werden können, solange $m \geq n$ gilt. Mit $m < n$ wurden bisher noch keine Simulationen mit Erfolg durchgeführt.

Jedoch geht die Erhöhung von D stark auf Kosten der Komplexität, denn die Anzahl der linear unabhängigen Gleichungen *Free*, die mit dem XL-Algorithmus generiert werden, steigen stark mit wachsendem D. Beispielsweise kann auch aus den Abbildungen herausgelesen werden, dass mit $n = 20$, $m = 65$ und $D = 3$ der Wert von *Free* 1350 beträgt. Für $D = 4$ werden nur 32 nichtlineare Gleichungen benötigt, aber wir erhalten *Free* $= 6195$. Wir erhalten folgende Anzahl an Operationen beim Gauss-Eliminationsverfahren:

$$\begin{array}{lrcll} \text{Für D=3:} & \mathit{Free}^{\omega} & = & 1350^{2.376} & = & 2.7 * 10^7 \quad \text{Operationen} \\ \text{Für D=4:} & \mathit{Free}^{\omega} & = & 6195^{2.376} & = & 1.023 * 10^9 \quad \text{Operationen} \end{array}$$

Wie man sieht, sind das beträchtliche Unterschiede. Mit wachsenden n steigt offensichtlich auch die Anzahl der Operationen drastisch, wie man es der Formel $\mathit{Free}^{\omega} \approx T^{\omega} \approx \binom{n}{D}^{\omega}$ entnehmen kann. Somit sollte D niedrig gewählt werden, falls möglich.

3.2.5 Vorteile & Nachteile

Mit dem XL-Algorithmus ist es möglich ein positives Ergebnis beim Lösen eines nichtlinearen Gleichungssystem zu liefern, trotz mangelnder Anzahl von Gleichungen. Die Gesamtanzahl der Gleichungen m sollte lediglich größer wie die Anzahl der Variablen n sein.

So können auch nichtlineare Gleichungssysteme, welche nicht überdefiniert sind, gelöst werden.

Nur muss möglicherweise D recht groß gewählt werden, damit genügend linear unabhängige Gleichungen generiert werden. Als obere Grenze wurde dabei in der Literatur $D \ll 2^n$ angegeben. Dies betrifft vor allem Systeme mit $m = n$. Mit wachsendem D kann der Algorithmus leider subexponentiell werden und würde dadurch ineffektiv werden. So würde eine Brute Force Methode ähnliche Zeitkomplexität wie der XL-Algorithmus mit sich bringen. Wie bereits erwähnt gilt dies nur für Systeme mit $m = n$ und eine subexponentielle Komplexität ist hier auch nicht zwingend. Aber mit $m > n$ sind polynomielle Laufzeiten zu erwarten.

Aufgrund dieser Eigenschaften wird XL oft angewendet, wenn es sich um das Lösen von nichtlinearen Gleichungssystem handelt. Auch in algebraischen Angriffen greift man des öfteren auf XL zurück, wenn die Werte der Variablen von nichtlinearen Gleichungen bestimmt werden sollen.

3.2.6 Modifikationen

Im Laufe der Zeit wurden viele Modifikationen des XL-Algorithmus veröffentlicht, die ähnlich wie XL funktionieren, aber optimiert sind und damit eine geringere Komplexität mitbringen oder auf Gleichungssysteme angewendet werden können, bei denen der XL-Algorithmus scheitert. Nun folgt zu den einzelnen Modifikationen eine kleine Zusammenfassung ihrer Funktionen. Genaueres kann in [CP03, CL05, Kle05, YCC04, CKPS00] nachgelesen werden.

XL': Mit XL' ist es möglich, dass der Algorithmus auf nichtlineare Gleichungssysteme angewendet werden kann, bei denen der Original XL-Algorithmus scheitert. Die Idee dabei ist, dass der Schritt 3 in XL in der Art modifiziert wird, dass nicht mehr eine Gleichung mit einer Variable erwartet wird, sondern ein Gleichungssystem mit $r \ll T$ Variablen. In GF(2) darf r bis zu 40 gewählt werden. Diese Auswahl an r Gleichungen wird dann mit der Brute Force Methode gelöst. Wir erhalten dadurch eine Komplexität von q^r, wobei q den Definitionsbereich der Variablen $x_i \in \{0, \ldots, q-1\}$ angibt. Die Werte, welche durch die Brute-Force Methode erhalten werden, werden in das Original Gleichungssystem eingesetzt, welches nun mit der Gauss-Eliminationsverfahren gelöst werden kann.

Der XL' Algorithmus kann angewendet werden, wenn folgende Gleichung erfüllt ist:

$$\mathit{Free} > T - \phi + r \quad \text{mit } \phi = \sum_{i=0}^{D} \binom{r}{i} \tag{17}$$

In XL war die Voraussetzung $\mathit{Free} > T - D$ gegeben, damit ein positives Ergebnis erwartet werden kann.

FXL: In XL kann mit $m = n$ hohe Werte für D erwartet werden, womit die Komplexität drastisch steigt. Die Idee von FXL ist dabei, dass bei solchen Situationen r Variablen mit r klein aus dem nichtlinearen Gleichungssystem gewählt werden. Dabei werden die Werte für r Variablen geraten und auf das nichtlineare Gleichungssystem mit $n - r$ Variablen der XL-Algorithmus ausgeführt. Für das Erraten der Variablenbelegung haben wir im ungünstigsten Fall die Komplexität von q^r zu erwarten. Zum anderen kann mit dieser Methode auch Gleichungen gelöst werden, wenn $m < n$ vorliegt.

Die Gesamtkomplexität liegt bei $\mathcal{O}(q^r * \binom{n-r}{D})$. Denn es wird q^r mal der XL-Algorithmus ausgeführt. Doch es können trotzdem subexponentielle Laufzeiten erwartet werden, selbst wenn $m \approx n$ gilt [CKPS00]. Auch in GF(2) wurde kein großer Erfolg erreicht mit dieser Methode [CP03].

XL2: XL2 nimmt Veränderungen am 3. Schritt von XL vor, ähnlich wie XL', aber es ist vielversprechender mit nur geringer zusätzlicher Komplexität. XL2 besitzt den Vorteil, dass es unter noch kleinerer Anzahl an generierten linear unabhängigen Gleichungen als die bisher bekannten Algorithmen anwendbar ist. Genauer kann ein Gleichungssystem mit XL2 gelöst werden, wenn gilt: $Free \geq T - T' + C$ mit $C > 1$. T' ist eine Untermenge von T und wie folgt festgelegt: $T' = |T'_i|$ mit $T'_i = \{t | t * x_i \in T\}$ für alle i.

Der Algorithmus läuft dabei wie folgt ab:

1. Mit dem Gauss-Eliminationsverfahren wird das Gleichungssystem in eine Form gebracht, die nur aus einer Linearkombination aus Termen in T'_i besteht. Dabei wurde die Variable x_i für T'_i gewählt.

2. Das gleiche Vorgehen wenden wir auf eine andere Variable x_j an.

3. Aus beiden Untermengen an Gleichungen C, welche wir aus den ersten beiden Schritten erhalten und nur Variablen aus T'_i bzw. T'_j besitzen, haben meist eine andere Gestalt als die aus dem Original Gleichungssystem. Daher können neue linear unabhängige Gleichungen gewonnen werden, indem wir die Gleichungen C mit x_i bzw. x_j multiplizieren.

Im letzten Schritt werden dabei $2 * C$ neue Gleichungen erwartet, die gewonnen werden und vorher nicht aus den gegebenen Gleichungen mit dem XL-Algorithmus gewonnen werden konnten.

Durch die zusätzlichen Operationen, die für das Generieren der neuen Gleichungen benötigt werden, kommt man auf folgende Komplexität $\mathcal{O}(T^\omega + T'^\omega)$. Und da T' um einiges kleiner als T ist, liegt die Komplexität in $\mathcal{O}(T^\omega)$, also gleich dem XL-Algorithmus.

Testergebnisse von XL2 können aus [CP03] entnommen werden.

XSL: Die Veränderung gegenüber XL liegt diesmal schon im ersten Schritt. In XL werden die Gleichungen durch alle möglichen Monome multipliziert, um eine maximale Anzahl an nichtlinearen Gleichungen zu erhalten. Die Idee in XSL ist diese, dass nur ausgewählte Monome zu den Gleichungen multipliziert werden, um somit weniger neue Monome zu bekommen. Denn durch die Linearisierung ist jedes Monom eine Variable, die dann zu lösen ist. Also haben wir durch XSL eine geringere Anzahl an Variablen und damit automatisch aber auch eine geringere Anzahl an Gleichungen, welche generiert werden.

Hinzu kommt noch ein zusätzlicher Schritt (T' genannt), in der neue zusätzliche linear unabhängige Gleichungen ohne zusätzliche Entstehung von Monomen gewonnen werden sollen. Weitere Details entnehmen sie den Arbeiten [CL05, Kle05].

3.3 Lösung durch Gröbner Basen

In den bisherigen Algorithmen wurde versucht neue linear unabhängige Gleichungen aus dem Originalsystem zu gewinnen, um damit möglichst viele Gleichungen zu finden, damit

der Gauss-Algorithmus angewendet werden kann. Der Ansatz hier, ist ein anderer. Mit Hilfe von Gröbner Basen wird das Original System durch Kombinationen und Reduktionen von Gleichungen in ein einfacheres System umgewandelt. Das Lösen dieses vereinfachten Gleichungssystems liefert äquivalente Werte zur Originallösung.

Gröbner Basen treten oft in Verbindung beim Lösen linearer oder nichtlinerarer Gleichungssysteme auf. Dabei beschränken sie sich meist auf eine endliche Menge als Körper, in welcher die Lösung zu finden ist. Sie besitzen gewisse Eigenschaften, die das Benutzen solcher Basen sinnvoll machen.

Es folgt im nächsten Abschnitt eine Idee, wie man mit Hilfe von Gröbner Basen ein nichtlineares Gleichunggsystem lösen kann. Dann folgen einige Definitionen und Beispiele bezüglich Gröbner Basen, weil diese nötig sind, um die Funktionsweise und Analyse dieser Variante zu verstehen.

3.3.1 Idee

Beim Lösen nichtlinearer Gleichungssysteme besitzt der Lösungsansatz mit Gröbner Basen ein anderes Vorgehen. Hier wird eine Menge an Generatoren gesucht, die eine Gröbner Basis zu gegebenen Polynomen angeben. Diese Polynome werden dabei aus dem vorliegenden Gleichungssystem gewonnen. Mit Hilfe der Gröbner Basis kann ein viel einfacheres Gleichungssystem erstellt werden, welches gewisse Eigenschaften mit sich führt. Diese werden in weiteren Abschnitten beschrieben.

Es existieren gewisse Techniken, womit neue Polynome gebildet werden oder auch die Anzahl der Variablen in einem Polynom reduziert werden können Es werden Schritt für Schritt Monome aus dem Gleichungssystem entfernt und versucht in eine bestimmte Form zu bringen. In dieser Form ist es leicht, die Lösung mit weitereren Schritten aus der linearen Algebra zu erhalten.

3.3.2 Vorbereitungen

In diesem Unterabschnitt werden einige wichtige Definitionen und Beispiele gegeben, die vorbereitend auf die folgenden Abschnitte sind. Dabei werden die Notationen von [JR95] oder auch [FA03] übernommen sowie die Beispiele vorgestellt.

Wir gehen von einem Polynomring $\mathcal{K}[x_1, \ldots, x_n]$ über einem beliebigen Körper $\mathcal{K}$ aus, welcher durch eine Menge von Polynomen gebildet wird:

$$f_k(x_1, x_2, \ldots, x_n) = \sum a_{p_1 \ldots p_n} * x_1^{p_1} \cdots x_n^{p_n} \tag{18}$$

wobei $a_{p_1 \ldots p_n} \in \mathcal{K}$, $p_i \in \mathbb{N}$ und $k = 1 \ldots m$. m gibt dabei die Anzahl der Polynome an. Wenn wir diese Polynome als Basis interpretieren, so erhalten wir die Menge $\mathcal{I}$ mit allen möglichen Linearkombination dieser Basis. Diese Menge nennt man *Ideal*. Die Basis des Ideals ist möglicherweise nicht optimal gewählt, da lineare Abhängigkeiten beinhalten sein könnten. Außerdem kann die Basis auch vereinfacht werden. Dafür jedoch wird eine Monomordnung in $\mathcal{K}[x_1, \ldots, x_n]$ benötigt.

Definition 1 *(Monomordnung)*
Eine Monomordnung in $\mathcal{K}[x_1, \ldots, x_n]$ ist eine Relation $\succ$ auf eine Menge von Monomen x^α, $\alpha = (\alpha_1, \ldots, \alpha_L) \in \mathbb{Z}^L$, bei der gilt:

(i) $\succ$ *ist eine totale Ordnung.*
(ii) *Falls $x^\alpha \succ x^\beta$ und $\gamma \in \mathbb{Z}^L$, dann folgt: $x^{\alpha+\gamma} \succ x^{\beta+\gamma}$.*
(iii) $\succ$ *ist wohlgeordnet.*

Beispiel 10 *(lexikographische Ordnung)*
Wir setzen $x_1 \succ x_2 \succ \ldots \succ x_n$ fest und damit gilt:

$$x_1^{p_1} \cdots x_n^{p_n} \succ x_1^{q_1} \cdots x_n^{q_n} \Leftrightarrow (p_1 > q_1) \vee ((p_i = q_i, i = 1, \ldots, s) \wedge (p_{i+1} > q_{i+1}))$$

mit $s < n$.
Zum Beispiel gilt $x_1^5 \succ x_1 x_2^6$ und $x_1^2 x_2^2 \succ x_1^2 x_3^2$.

Wir können eine Funktion bzw. ein Polynom in $\mathcal{K}[x_1, \ldots, x_n]$ auch so darstellen:

$$f = a_0 MON_0 + \cdots + a_l MON_l$$

wobei $MON_0 \succ MON_1 \succ \ldots \succ MON_l$ und l ist die Anzahl der Monome des Polynoms. Wir nennen das Polynom *normalisiert*, wenn $a_0 = 1$ gilt. Wir definieren das Leitmonom mit $LM(f) = MON_0$ und den Leitkoeffizienten mit $LC(f) = a_0$.

Beispiel 11 *Es gilt folgende lexikographische Ordnung: $x_1 \succ x_2 \succ x_3$. Damit erhalten wir:*

$f_1 =$	$3 * x_3 * x_2 + x_1$	*ist in ungültiger Form,*
$f_2 =$	$5 * x_1^2 + x_2 * x_3 + 3 * x_3 + 3$	*ist in gültiger Form, und*
$f_3 =$	$x_1 * x_2 + 2x_2^2$	*ist in gültiger und normalisierter Form*

Im weiteren nehmen wir an, dass die Polynome normalisiert sind. Mit diesem Wissen ist es nun möglich, eine Gröbner Basis zu definieren.

Definition 2 *(Gröbner Basis)*
Eine endliche Menge $G = \{g_1, g_2, \ldots, g_n\}$ eines Ideals $\mathcal{I}$ ist eine Gröbner Basis bezüglich einer Monomordnung $\succ$, wenn $\forall f \in \mathcal{I}$, $\exists g \in G$ gilt: LM(g) teilt LM(f).

Sei $\mathcal{I} \neq 0$ ein Ideal über einen Polynomring und $\succ$ eine Monomordnung, so erhalten wir folgende Eigenschaften [FA04]:

- Es existiert immer eine Gröbner Basis G zu $\mathcal{I}$ mit $\langle G \rangle = \mathcal{I}$, d.h. dass die Basis von G komplett $\mathcal{I}$ erzeugt.
- $\mathcal{I}$ hat eine eindeutige reduzierte Gröbner Basis G, wobei eine Basis reduziert ist, wenn gilt:
  (i) Alle Basiselemente $g_i \in G$ sind normalisiert.
  (ii) $\forall g \in G$ und m Monom von g, $\nexists f \in G$, so dass m von LM(f) geteilt wird.
- $G = \{X_0 + b_0, \ldots, X_{L-1} + b_{L-1}\}$, wenn $(b_0, \ldots, b_{L-1})$ die einzige Lösung des nichtlinearen Gleichunggsystem ist mit der Form:

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ f_m(x_1, \ldots, x_n) &= 0 \end{aligned}$$

- $G = \{1\}$, falls keine Lösung im definierten Feld existiert.

3.3.3 Funktionsweise

Beim Lösen eines nichtlinearen Gleichungssystems mit Hilfe von Gröbner Basen, muss man wie folgt vorgehen: Wir bringen das Gleichungssystem in die Form

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ f_m(x_1, \ldots, x_n) &= 0 \end{aligned}$$

wobei n die Anzahl der Variablen und m die Anzahl der Gleichungen bestimmt. Diese f_k mit $k \in \{1, \ldots, m\}$ sind dabei die Eingabe für welche eine Gröbner Basis bestimmt wird.

Es existieren einige Algorithmen, womit Gröbner Basen bestimmt werden, nachdem sie als Eingabe eine Anzahl von Polynomen erhalten halten. Der erste und einer der bekanntesten unter den Algorithmen, welche dieses Ziel verfolgt, ist von Buchberger [Buc65] eingeführt. Es folgten effizientere Algorithmen wie F_4 und F_5 [Fau02]. F_4 nutzt dabei auch lineare Algebra für die Bestimmung einer Gröbner Basis. F_5 besitzt noch andere Techniken, um Reduktionen zu 0 zu vermeiden.

Die Problematik bei der Bestimmung von Gröbner Basen liegt dabei unter anderem daran, dass sehr viele Operationen, vor allem im Buchberger Algorithmus, auf unnötige Operationen fallen. 90% der Operationen des Buchberger Algorithmus fallen auf Reduktionsaktionen mit negativem Ergebnis. Wie angesprochen wird durch Reduktion der gegebenen Polynome eine Gröbner Basis zu bestimmen versucht. Des öfteren ist der Fall, dass keine Reduktion mit den gewählten 2 Polynomen statt finden kann. Daher haben wir mit diesem negativen Ergebnis unnötige Rechenzeit verwendet. Dieser Problematik wendeten sich vor allem F_4 und F_5 zu.

Um eine Idee zu erhalten, wie eine Gröbner Basis erstellt werden kann, wird hier der Buchberger Algorithmus vorgestellt. Doch zuvor folgen einige Definitionen.

Das kleinste gemeinsame Vielfache zweier Monome $MON_i = x_1^{p_1} \cdots x_n^{p_n}$ und $MON_j = x_1^{q_1} \cdots x_n^{q_n}$ wird definiert durch

$$LCM(MON_i, MON_j) := x_1^{m_1} \cdots x_n^{m_n}$$

wobei $m_i = \max\{p_i, q_i\}$ für $i \in \{1, \ldots, n\}$.
Als Beispiel bekommen wir $LCM(x_1^4 * x_2 * x_3^6, x_1^2 * x_2^2 * x_3^2) = x_1^4 * x_2^2 * x_3^6$.

Definition 3 *(S - Polynom)*
Sei f_1,$f_2 \in \mathcal{K}[x_1, \ldots, x_n] \backslash \{0\}$. Dann wird

$$S(f_1, f_2) := \frac{LCM(LM(f_1), LM(f_2))}{LM(f_1)} * f_1 - \frac{LCM(LM(f_1), LM(f_2))}{LM(f_2)} * f_2$$

S-Polynom von f_1 und f_2 genannt.

Definition 4 *(M - Reduktion)*
Sei f_1,$f_2 \in \mathcal{K}[x_1, \ldots, x_n] \backslash \{0\}$. Und gilt weiter, dass $LC(f_1)$ ein Vielfaches von $LC(f_2)$ ist, dann ist ein vereinfachtes S-Polynom konstruierbar durch

$$M(f_1, f_2) := f_1 - \frac{LM(f_1)}{LM(f_2)} * f_2$$

Wenn f_1 durch $M(f_1, f_2)$ ersetzt wird, so nennt man dies eine M-Reduktion von f_1 oder "f_1 wurde durch f_2 reduziert". Wenn $M(f_1, f_2) = 0$, dann heißt es, dass f_1 auf 0 reduziert wurde.

Beispiel 12 *Seien die folgenden 4 Funktionen gegeben:*

$$\begin{aligned} f_1 &= x_1^2x_2^2 + x_2^2 \\ f_2 &= x_1^3 + x_1 * x_2^2 \\ f_3 &= x_1x_2^4 - x_2^2 \\ f_4 &= x_1x_2 - x_2 \end{aligned}$$

Wir erhalten

$$LCM(LM(f_1), LM(f_2)) = LCM(x_1^2x_2^2, x_1^3) = x_1^3 * x_2^2.$$

Nun berechnen wir $S(f_1, f_2)$ wie folgt:

$$\begin{aligned} S(f_1, f_2) &= \frac{x_1^3 * x_2^2}{x_1^2x_2^2} * (x_1^2x_2^2 + x_2^2) - \frac{x_1^3 * x_2^2}{x_1^3} * (x_1^3 + x_1 * x_2^2) \\ &= x_1 * (x_1^2x_2^2 + x_2^2) - x_2^2 * (x_1^3 + x_1 * x_2^2) \\ &= (x_1^3x_2^2 + x_1x_2^2) - (x_1^3x_2^2 + x_1 * x_2^4) \\ &= -x_1x_2^4 + x_1x_2^2 \end{aligned}$$

Weiter erkennen wir, dass $LM(f_3)$ ein Vielfaches von $LM(f_4)$ ist, daher reduzieren wir f_3 durch f_4 mit $M(f_3, f_4)$ wie folgt:

$$\begin{aligned} f_3 &:= (x_1x_2^4 - x_2^2) - \frac{x_1x_2^4}{x_1x_2} * (x_1x_2 - x_2) \\ &= (x_1x_2^4 - x_2^2) - x_2^3 * (x_1x_2 - x_2) \\ &= (x_1x_2^4 - x_2^2) - (x_1x_2^4 - x_2^4) \\ &= x_2^4 - x_2^2 \end{aligned}$$

In einer Basisversion des Buchberger Algorithmus werden Polynome, also die Basiselemente, soweit reduziert, dass am Ende eine Dreickecksform der Matrix A aus den Basispolynomen erlangt wird. Dabei bildet sich die Matrix A wie folgt:

$$\mathbf{A} = \begin{array}{c} MON_0 MON_1 \cdots \\ \begin{pmatrix} a_{1,0} & a_{,1,1} & \cdots \\ a_{2,0} & a_{2,1} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \end{array}$$

wobei in einer Zeile die Polynome $f_k = \sum_i a_{k,i} * MON_i$ stehen. MON_i sind dabei alle möglichen Monome vom Grad $\max_k deg(f_k)$.

Diese Variante reicht nicht im nichtlinearen Fall aus, weil nicht genügend Zeilen vorhanden sein werden, um eine Dreiecksform zu bilden. Daher brauchen wir sowohl Reduzierungen als auch neue Polynome müssen in die Basis aufgenommen werden. Diese Variante des Buchberger Algorithmus funktioniert wie folgt (näheres in [JR95]):

```
/* Eingabe: Ein Gleichungssystem G aus Polynomen */

F := G;
L_F := L_G;
i:= 1;
while ( i <= L_F -1) do
begin
  j := i+1;
  while ( j <= L_F) do
  begin
    if LCM(LM(f_i),LM(f_j)) != LM(f_i) * LM(f_j) then
      begin
        h := S(f_i,f_j);
        while (M-Reduktion_of_h_by_f_k_from_F_possible) do
          h:= M(h,f_k));
        if (h != 0) then
          add_polynomial_to_F(h); L_F := L_F + 1;
        if LM(f_j) = LCM(LM(f_i),LM(f_j))) then
          rem_polynomial_from_F(f_j); L_F := L_F -1;
          renumber_the_polynomials_in_F();
        else if LM(f_i) = LCM(LM(f_i),LM(f_j))) then
          rem_polynomial_from_F(f_i); L_F := L_F -1;
          renumber_the_polynomials_in_F();
          i := i-1; j:= L_F +1;
        else
          j := j +1;
      end
    else
      j:= j+1;
  end
  i := i +1;
end

/* Ausgabe: eine Groebner Basis F mit Ideal(F) = Ideal(G) */
```

Nun wird dieser Algorithmus mit einem kleinen Beispiel demonstriert.

Beispiel 13 *Wir führen den Buchberger Algorithmus mit folgendem Gleichungssystem als Eingabe aus:*

$$\begin{array}{rcccccccc} f_1 & = & x_1^2 & & & - & x_3 & = & 0 \\ f_2 & = & x_1^2 & - & x_2^2 & & & = & 0 \\ f_3 & = & & & x_2 & - & x_3^2 & = & 0 \end{array}$$

Als erstes kritisches Paar wählen wir f_1 und f_2 und berechnen zunächst das kleinste gemeinsame Vielfache. Wir erhalten $LCM(f_1, f_2) = x_1^2$. Somit bekommen wir folgendes S-Polynom h_1:

$$h_1 = S(f_1, f_2) = \frac{x_1^2}{x_1^2} * (x_1^2 - x_3) - \frac{x_1^2}{x_1^2} * (x_1^2 - x_2^2) = x_2^2 - x_3$$

Nun überprüfen wir, ob h_1 ein Vielfaches von einen der Basispolynome ist und finden

heraus, dass h_1 ein Vielfaches von f_3 ist. Eine M-Reduktion mit f_3 ergibt:

$$h_2 = M(h_1, f_3) = x_2^2 - x_3 - \frac{x_2^2}{x_2} * (x_2 - x_3^2) = x_2 x_3^2 - x_3$$

Eine weitere M-Reduktion ergibt:

$$h_3 = M(h_2, f_3) = x_2 x_3^2 - x_3 - \frac{x_2 x_3^2}{x_2}(x_2 - x_3^2) = x_3^4 - x_3$$

Nun wird f_3 aus der Basis entfernt und h_3 statt dessen eingefügt. Es müssen keine weiteren S-Polynome gebildet werden, denn es liegt bereits die verlangte Dreiecksform vor. Wir erhalten folgendes System nach den Algorithmus:

$$\begin{array}{lllllll} f_1 & = & x_1^2 & & - & x_3 & = 0 \\ f_2 & = & & x_2^2 & - & x_3 & = 0 \\ f_3 & = & & & & x_3^4 - x_3 & = 0 \end{array}$$

Nun kann es passieren, dass nach einem Durchlauf des Algorithmus noch nicht die verlangte Dreiecksform als Output zurückgegeben wird. In diesem Fall führt die mehrmalige Ausführung des Algorithmus zum gewünschten Ergebnis.

Die Funktionsweisen der Algorithmen F_4 und F_5 werden hier nicht im Detail beschrieben. F_5 enthält neue Ansätze, wie die Reduzierung eines Polynoms auf 0 verhindert werden soll. Der Algorithmus erstellt eine Matrix $\mathcal{M}_{D,m}^{acaulay}$ [FA03, FA04] und versucht diese mit linearer Algebra und Reduktionen sowie Einfügen neuer Basispolynome auf eine Dreiecksform zu bringen. Dabei ist D die Laufvariable, die den maximalen Grad der auftretenden Monome bestimmt. Die Variable D wird soweit erhöht, bis die Matrix $\mathcal{M}_{D,m}^{acaulay}$ auf eine Dreiecksform gebracht werden kann.

3.3.4 Voraussetzungen

Hier stellt sich die Frage, wann der Buchberger Algorithmus mit einem positivem Ergebnis terminiert. Wie im Kapitel 3.3.2 erwähnt wurde, gibt eine Gröbner Basis G mit $G = \{1\}$ keine Lösung des nichtlinearen Gleichungssystems. Das folgt nach dem Kriterium 4.1 von [B.B70]:

Kriterium 1 *Ein Polynomideal hat genau dann keine Nullstelle, wenn während der Ausführung des Algorithmus das Polynom $x_1^0 \dots x_n^0$ in die Idealbasis aufgenommen werden muss.*

Nach dem Kriterium lässt sich folgendes festlegen:
Ein System von Polynomen ist genau dann inkonsistent, wenn die erhaltene Gröber Basis eine Konstante als Basiselement besitzt.

Also können wir mit den Algorithmen, die mit Hilfe von Gröbner Basen arbeiten, mögliche Lösungen des nichtlinearen Gleichungssystem erhalten, wenn wir als Eingabe ein konsistentes System von Polynomen haben.

Nun kann es aber passieren, dass unendlich viele Lösungen bei gegebenem Gleichungssystem erhalten werden. Aus [JR95] sagt folgendes Kriterium, wann wir eine endliche Menge von Lösungen erwarten können.

Kriterium 2 *Das System aus polynomiellen Gleichungen besitzt genau dann eine endliche Menge an Lösungen, wenn jede Variable alleine (z.B. z^n) nur in einem Leitmonom der Basiselemente auftritt.*

Beim Ausführen des Buchberger Algorithmus erhalten wir eine Basis, die in Dreiecksform vorliegt. Daher kann dieses Kriterium nicht auftreten, weil ein Leitmonom jedes Generators eine andere Potenz beziehungsweise ein anderes Leitmonom besitzt.

Genauere Aussagen über die Möglichkeit der Anwendbarkeit des Algorithmus lassen sich für den F_5 Algorithmus treffen. Die Frage hier ist diese, mit welchem D eine genügend große Matrix $\mathcal{M}_{D,m}^{acaulay}$ erwartet werden kann, dass diese in eine Dreiecksform umgewandelt werden kann. Nach [FA04] gilt, dass eine Gröbner Basis mit Eingabe eines Ideals, welches aus $f_1, f_2, \ldots, f_n$ mit Grad $d_1, d_2, \ldots, d_n$ gebildet wurde, berechnet werden kann, wenn gilt:

$$D \leq d_1 * d_2 * \cdots d_n$$

wobei das System einer lexikographischen Monomordung unterliegt. Das ist natürlich nur eine obere Schranke, aber dieses D gibt die minimale Zahl an, für welche der F_5 Algorithmus eine Gröbner Basis liefert, die in Dreiecksform vorliegt. Nach [FA03] kann man eine kleinere obere Schranke von D finden. Wenn ein nichtlineares Gleichungssystem vorliegt mit folgender Gestalt

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ f_m(x_1, \ldots, x_n) &= 0 \end{aligned}$$

, dann kann zu einem Ideal $\mathcal{I}$, welches durch $f_1, \ldots f_m$ definiert wurde, eine Gröbner Basis mit $D \leq \lfloor \frac{n+1}{2} \rfloor$ berechnet werden. In praktischen Fällen können sogar noch geringere Werte für D erwartet werden. Dieses D bezieht sich auf den Fall, dass die Funktionen im Feld $GF(2)$ definiert sind.

Ein anderer Punkt, welcher analysiert werden sollte, ist die benötigte Anzahl an Gleichungen, mit denen ein positives Ergebnis von den Algorithmen zu erwarten ist. Wir können leicht eine minimale Anzahl an m Gleichungen angeben, ab der auf jeden Fall F_5 angewendet werden kann.

Sei $M(n, D)$ die Anzahl aller verschiedenen Monome, die auftreten können, mit Grad D. Wir erhalten

$$M(n, D) := \sum_{i=0}^{D} \binom{n}{i}.$$

Da bekannt ist, dass $D \leq \lfloor \frac{n+1}{2} \rfloor$ gilt, kann eine Gröbner Basis berechnen werden, wenn $m \geq M(n, \lfloor \frac{n+1}{2} \rfloor)$ gilt, da dies allein mit linearer Algebra in eine Dreiecksform gebracht werden kann. Im Feld F_2 gilt:

$$M(n, \lfloor \frac{n+1}{2} \rfloor) = n^{\lfloor \frac{n+1}{2} \rfloor}$$

In der Praxis jedoch werden viel weniger Gleichungen benötigt. Eine Theorie aus [FA03] besagt, dass $n+\epsilon$ Gleichungen ausreichen, um eine Lösung eines nichtlinearen Gleichungssystem zu erhalten. Diese These wurde experimentell bestätigt, aber es liegen bisher keine Beweise diesbezüglich vor.

3.3.5 Komplexität

Wir gehen hier nicht weiter auf die Komplexität des Buchberger Algorithmus ein. Wie bereits besagt, verwendet dieser Algorithmus bis zu 90% der Laufzeit dazu, Reduzierungen auf 0 durchzuführen. Das sind unnötige Rechenoperationen, die dafür verschwendet

werden. Daher analysieren wir die Komplexität des F_5 Algorithmus, welcher durch neue und verbesserte Ansätze keine Reduzierungen auf 0 mehr zulässt, beziehungsweise nur in sehr seltenen Fällen.

Wenn wir die Komplexität des F_5 Algorithmus einschätzen wollen, dann benötigen wir das maximale D, mit welchem der Algorithmus die Matrix $\mathcal{M}_{D,m}^{acaulay}$ in eine Dreiecksform bringen kann. Man kann im Grunde festlegen, dass die Komplexität des Algorithmus auf die Komplexität des Umformens der Matrix $\mathcal{M}_{D,m}^{acaulay}$ in eine Dreiecksform zurückgeführt werden kann.

Aus dem vorherigen Unterkapitel wissen wir, dass das maximale D, womit der Algorithmus terminiert, gegeben ist mit $D \leq \lfloor \frac{n+1}{2} \rfloor$. Damit bekommen wir für das Umformen der Matrix $\mathcal{M}_{D,m}^{acaulay}$ in eine Dreiecksform eine Komplexität von: $(\sum_{i=0}^{\lfloor \frac{n+1}{2} \rfloor} \binom{n}{i})^{\omega} = (n^{\lfloor \frac{n+1}{2} \rfloor * \omega})$ mit $\omega \leq 3$ der Faktor des verwendeten Algorithmus aus der linearen Algebra.

Im Fall, dass $n * \log_2(n)$ Gleichungen vorliegen, kann nach [FA03] eine Komplexität von $\mathcal{O}(\exp^{n \log(\log(n))/log(n)})$ erwartet werden. Dies besagt, dass im ungünstigsten Fall hier eine subexponentielle Laufzeit erwartet werden kann.

Im Allgemeinen lässt sich sagen, dass je mehr Gleichungen der Algorithmus als Eingabe erhält, desto schneller kann eine Gröbner Basis berechnet werden.

3.3.6 Vorteile & Nachteile

Die Algorithmen, die mit Hilfe von Gröbner Basen, eine Lösung des nichtlinearen Gleichungssystems suchen, haben immer dann Erfolg, wenn ein konsistentes System von Polynomen vorliegt. Wenn es eine Lösung zu einem Gleichungssystem gibt, so haben in allen getesteten praktischen Fällen die Algorithmen die Lösung mit Hilfe von Gröbner Basen berechnet. Daher kann man eigentlich immer von einem konsistenten System ausgehen, wenn eine Lösung zu einem nicht manipulierten System gesucht wird.

Nach einigen Studien fand man heraus, dass Lösungen zu Gleichungssystemen erwartet werden können, wenn mehr als $n + \epsilon$ Gleichungen vorliegen, wobei n die Anzahl der Variablen im Gleichungssystem ist. Doch kann man bei niedriger Anzahl von Gleichungen auch eine subexponentielle Laufzeit erwarten. Schon bei $n * \log_2(n)$ Gleichungen befindet sich die Komplexität in Ordnung $\mathcal{O}(\exp^{n \log(\log(n))/log(n)})$. Aber nach Studien von [Fau02], wurde gezeigt, dass auch eine Lösung mit dem F_5 Algorithmus erwartet werden kann, selbst wenn gleich oder weniger Gleichungen als Variablen gegeben sind. Es wurden aber keine Komplexitätspunkte in diesen Situationen untersucht. Aber man kann von subexponentieller Laufzeit ausgehen. Auch könnte eine sehr große Lösungsmenge erwartet werden, die nicht sonderlich hilfreich wäre.

Diese Komplexitätsangabe bezieht sich aber auf den ungünstigsten Fall. Im praktischen Fällen wurden immer viel kürzere Laufzeiten erreicht. Wie auch bereits angesprochen, kann die Funktion, die hinter diesen Gleichungen stecken, noch in vereinfachter Form dargestellt werden, so dass eine Lösung auch mit Grad $D \leq \max_i deg(f_i)$ berechnet werden kann. Genauer fand man heraus, dass $D \leq \lfloor \frac{n+1}{2} \rfloor$ bereits ausreichend ist, damit eine Lösung gegeben werden kann. Selbst dieser Wert ist eine obere Grenze und wird des öfteren unterschritten. Einige Beispiele werden in Kapitel 4 folgen.

3.4 Zusammenfassung und Vergleich der vorgestellten Algorithmen

Es wurden nun 3 Methoden vorgestellt, welche zur Bestimmmung der Lösung eines nichtlinearen Gleichungssystem verwendet werden können. Nun stellt sich die Frage, unter welchen Voraussetzungen welcher dieser Algorithmen genutzt werden sollte, um die Lösung zu bestimmen.

Die Linearisierungmethode wandelt das nichtlinearere Gleichungssystem in ein lineares Gleichungssystem um und muss im ungünstigsten Fall ein Gleichungssystem mit n^d Gleichungen lösen, wobei d den maximalen Grad aller Funktionen angibt. Mit Algorithmen aus der linearen Algebra kommt man zu einer Komplexität von $\mathcal{O}(n^{d*\omega})$ mit $\omega < 3$. Für ein festes n erhält man eine polynomielle Laufzeit, aber mit $\lim_{n\to\infty}$ erhalten wir eine exponentielle Laufzeit.

Die größte Problematik bei der Verwendung der Linearisierungsmethode ist, dass sehr viele Gleichungen benötigt werden, damit die Linearisierungmethode angewendet werden kann. Im ungünstigsten Fall treten im Gleichungssystem alle möglichen Monome bis zu einem Grad d auf. Wir erhalten damit $\sim n^d$ Monome, welche beim Linearisieren zu einer neuen Variable werden. Daher brauchen wir auch dementsprechend viele Gleichungen. Wenn diese Anzahl an Gleichungen nicht gewonnen werden kann, dann muss man auf die zwei verbliebenen Algorithmen ausweichen und nimmt so zusätzliche Laufzeit in Kauf.

Die Frage, die sich nun stellt, was die Gemeinsamkeiten und Unterschiede der Algorithmen, die mit Hilfe von Gröbner Basen agieren, und die des XL Algorithmus ist. Zu aller erst analysieren wir die Voraussetzungen beider, unter welchen Bedingungen sie eine Lösung eines nichtlinearen System bieten können.

Der XL Algorithmus erhält als Eingabe D, womit bestimmt wird bis zu welchem Grad die entstehenden Gleichungen nach der Multiplikation etlicher Monome sein dürfen. Es wurde gezeigt, dass immer eine Lösung des nichtlinearen Gleichungssystem erwartet werden kann, wenn wir D so wählen, dass $D \approx \frac{n}{m^{1/K}}$ gilt, wobei K den maximalen Grad, welcher in allen Termen vorkommt, bestimmt. Bei passender Wahl von D ist es also immer möglich das Gleichungssystem zu lösen, sofern allerdings $m \geq n$ gilt. Andererseits steigt die Komplexität drastisch mit wachsendem D.

Algorithmen, die mit Hilfe von Gröber Basen die Lösung eines nichtlinearen Gleichungssystem suchen, versuchen durch Reduktionen und Kombinationen von Gleichungen das Originalsystem umzuwandeln. Dabei soll eine Dreiecksform einer Matrix, welche in den Spalten die Monome und in den Zeilen die gegebenen Polynome besitzt, erreicht werden, womit leicht die Lösung errechnet werden kann. Im nichtlinearen Fall reichen nur Reduktionen nicht aus, sondern es müssen noch neue Polynome in die Basis hinzugefügt werden.

Der Buchberger Algorithmus kann immer dann ein positives Ergebnis liefern, wenn das gegebene Gleichungssystem konsistent ist. Dies trifft aber auch auf fast alle zu, daher kann ein Ergebnis mit dem Buchberger Algorithmus erwartet werden, wobei bei mangelnder Anzahl von Gleichungen viele Lösungen existieren könnten.

Der F_5 Algorithmus nutzt eine Matrix, deren Größe durch ein D bestimmt wird. Die gegebenen Gleichungen befinden sich in der Matrix und werden durch Reduktionen auf eine Dreiecksform gebracht. Dabei werden durch neue Kriterien die des Buchberger ersetzt und die Reduktion auf 0 fast komplett verhindert. Die Größe D wird dabei soweit erhöht, dass die dadurch erhaltene Matrix in Dreiecksform gebracht werden kann. Es findet sich

immer ein D für ein erfolgreiches Ergebnis mit F_5. Die obere Grenze für ein D lautet $D \leq d_1 * d_2 * \cdots d_n$, wobei $d_i = deg(f_i)$.

Nach einer Studie von [FA04] wurde die These bewiesen, dass wenn ein Gleichungssystem mit der Eingabe D mit dem XL Algorithmus gelöst wird, dann ist es möglich das selbe Gleichungssystem mit dem F_5 Algorithmus zu lösen, wobei seine Matrix den Parameter D nicht überschreitet. In Einzelfällen kann sogar eine geringeres D durch F_5 erwartet werden.

Nun kommen wir zum Komplexitätsvergleich der beiden Algorithmen. XL sowie F_5 erstellen eine Matrix, die am Ende mit linearer Algebra gelöst werden soll. Sei die Größe der Matrix gegeben durch N_D. Dann benötigt man eine Komplexität von N_D^ω. Die Algorithmen generieren dabei Matrizen von unterschiedlicher Größe. XL erstellt eine Matrix mit $\sum_{i=1}^{m} \sum_{k=0}^{D-degf_i} \binom{n}{k}$ Zeilen und $\sum_{k=0}^{D} \binom{n}{k}$ Spalten. Die durch F_5 generierte Matrix besitzt eine quadratische Größe. Die Anzahl der Zeilen und Spalten beträgt $\sum_{k=0}^{D} \binom{n}{k}$. Hier muss hinzugefügt werden, dass die Anzahl der Zeilen druch den XL Algorithmus die Gesamtanzahl der erhaltenen Gleichungen ist, d.h. mit den linear abhängigen Gleichungen.

In [FA04] wurde festgestellt, dass XL in Fällen $m = n + 2$ meistens längere Rechenzeit benötigt, da die Größe der Matrix eben größer ist als die der F_5 erhaltenen Matrix. Aber für eine größere Anzahl m besitzt XL eine bessere Laufzeit. Dies kommt dadurch zustande, dass die quantitative Anzahl der linear unabhängigen Gleichungen mit wachsendem m auch steigt und wir somit eine geringere Matrix enthalten. Somit kommt die Größe der Matrix dem F_5 näher. Nur kann es passieren, dass F_5 eine geringeres D für ein gegebenes Gleichungssystem erhält als das von XL.

Zusammenfassend kann mit Gröbner Basen eine kürzere Laufzeit gegenüber XL erwartet werden, je näher m an n ist. Mit wachsendem m sollte man auf den XL Algorithmus zurückgreifen. Ich empfehle [FA04] für weitere Details.

4 Algebraische Angriffe auf LFSR

Nachdem nun die linear rückgekoppelten Schieberegister vorgestellt wurden und uns gewisse Techniken zur Verfügung stehen, um nichtlineare Gleichungssysteme zu lösen, wird in diesem Kapitel der Fokus auf der Sicherheit der LFSR basierten Stromchiffren liegen.

Wir prüfen hier die Sicherheit der LFSR basierten Stromchiffren, indem Angriffe auf solche Systeme getätigt werden. Dabei folgen wir dem Prinzip Kerckhoffs, welche besagt, dass die Sicherheit eines Systems nur von der Sicherheit des geheimen Schlüssel abhängt. Das bedeutet in unserem Fall, dass die Sicherheit auf der Geheimhaltung des Initialisierungsvektors $\mathcal{K}$ der LFSR liegt. In unseren Angriffsmodellen sind sowohl die verwendeten booleschen Ausgabefunktionen f als auch die Schlüsselstrombits z_t des Schlüsselstromgenerators bekannt. Das Ziel liegt hier, den geheimen Schlüssel ausfindig zu machen.

Durch den geheimen Schlüssel ist es dem Angreifer möglich alle Zustände zu einem Zeitpunkt t selbst zu berechnen. Durch Kenntnis der Ausgabefunktion erhält der Angreifer alle Ausgabebits z_t und mit Hilfe des Chiffretextes c_t kann dieser nun auch den Klartext berechnen, denn es gilt:

$$z_t \oplus c_t = m_t \quad \forall t$$

Es gibt bereits viele andere Angriffe auf Stromchiffren wie Korrelationsangriffe, BDD basierte oder Backtracking Angriffe, aber wir konzentrieren uns in dieser Arbeit auf algebraische Angriffe. Die Idee bei algebraischen Angriffen liegt darin, ein System aus nichtlinearen Gleichungen zu lösen. Diese Gleichungen entstehen aus den Schlüssel- und Ausgabebits. In Kapitel 3 wurden schon bereits Verfahren vorgestellt, wie diese Systeme gelöst werden können.

Allgemein besteht ein algebraischer Angriff aus 3 Schritten:

1. Erstelle ein System aus Gleichungen mit Variablen aus dem Schlüssel $\mathcal{K}$ und den Ausgabebits z_t.

2. Setze die zur Verfügung stehenden Ausgabebits z_t in die Gleichungen ein.

3. Löse das erhaltene Gleichungssystem und gewinne dadurch den Schlüssel $\mathcal{K}$.

Sei F die Ausgabefunktion der Stromchiffre und z_t das Ausgabebit zum Zeitpunkt t. Sei weiter s_t die verwendeten Bits aus dem internen Zustand S_t der LFSR zum Zeitpunkt t. Dann ist der Angreifer fähig folgende Gleichungen aufzustellen:

$$F(s_t) \oplus z_t = 0 \quad \forall t > 0 \tag{19}$$

Der interne Zustand eines LFSR wechselt seinen Zustand nach jedem Zeitpunkt durch eine Feedback-Funktion L. Wir erhalten also den Folgezustand durch die Gleichung $S_{t+1} = L(S_t)$. Mit mehrmaligem Aufruf der Feedback-Funktion können die weiteren Folgezustände berechnet werden und es gilt: $S_t = L^t(S_0) = L^t(\mathcal{K})$. Das bedeutet, dass (19) umgeschrieben werden kann durch:

$$F(L^t(\mathcal{K})) \oplus z_t = 0 \quad \forall t > 0 \tag{20}$$

Nun haben wir das gewünschte Ergebnis des ersten Schrittes. Es liegt ein Gleichungssystem mit Unbekannten nur aus den Werten der Schlüsselbits vor. Mit Schritt 2 können nun die Ausgabebits eingesetzt werden, die ein Angreifer sich vorher beschafft haben müsste. Die Ausgabefunktion F und die Feedback-Funktion L sind dem Angreifer bekannt.

Das erhaltene Gleichungssystem ist möglicherweise durch die Ausgabefunktion F nichtlinear. Alle auftretenden Monome im kompletten Gleichungssystem besitzen aber eine obere Schranke für den Grad d mit $d := deg(F)$. Dieses nichtlineare Gleichungssystem kann je nach gegebener Situation durch Linearisierung, XL Algorithmus oder mit Hilfe von Gröbner Basen gelöst werden.

An einem kleinen Beispiel einer Stromchiffre soll ein algebraischer Angriff angesetzt werden.

Beispiel 14 *Sei die Stromchiffre aus Beispiel 1 gegeben. Diese besteht aus einem LFSR der Länge 4 mit internem Zustand* $S^{(t)} = (s_0^{(t)}, s_1^{(t)}, s_2^{(t)}, s_3^{(t)})$ *zum Zeitpunkt* t*. Die Feedback-Funktion ist gegeben durch* $L(S_t) := s_0 \oplus s_3$*. Die Ausgabe der Stromchiffre soll durch das Bit, welches durch Shiften des LFSR ausgegeben wird, bestimmt sein und daher folgt für die Ausgabefunktion* $F(S_t) := s_0$*. Der Angreifer bekam folgende Ausgabebits der Stromchiffre:* 101100100011110*.*

Nun kann der Angreifer durch das Wissen die folgenden Gleichungen aufstellen:

$$\begin{aligned} z_1 &= F(S_0) = s_0 \\ z_2 &= F(L(S_0)) = s_1 \\ &\vdots \end{aligned}$$

$$\begin{aligned} \Longrightarrow \quad z_1 &= s_0 \\ z_2 &= s_1 \\ z_3 &= s_2 \\ z_4 &= s_3 \end{aligned}$$

Der Angreifer hat nun die Variablen s_0*,* s_1*,* s_2 *und* s_3 *bestimmt und kennt damit den Initialisierungszustand* $S^{(0)}$*. Damit wurde der geheime Schlüssel* $\mathcal{K}$ *ausfindig gemacht.*

In diesem Beispiel haben wir schon mit 4 Ausgabebits den Schlüssel $\mathcal{K}$ berechnen können. Das gelang uns aus dem Grund, dass die Ausgabefunktion der Stromchiffre linear war. Sie hat die Ausgabe des LFSR noch nicht mal beeinflusst. Wie bereits im Kapitel 2 erwähnt sind gerade diese linearen Abhängigkeiten der LFSR ein Schwachpunkt für Angriffe.

Um die Sicherheit zu verbessern, wurden die Ansätze aus Kapitel 2 gebracht. In den folgenden Abschnitten werden Angriffe auf Stromchiffren vorgestellt, welche durch eine nichtlineare Funktion Angreifern erschweren soll, an den geheimen Schlüssel heranzukommen. Zum anderen werden Angriffe auf Combiner beschrieben. Des Weiteren werden dann die unregelmäßig getakteten Stromchiffren untersucht und mögliche Angriffe auf solche präsentiert.

4.1 Anwendung auf LFSR mit nichtlinearem Filtergenerator

Die Funktionsweise der LFSR mit nichtlinearem Filtergenerator wurde im Kapitel 2.1 vorgestellt. Es wird versucht durch eine nichtlineare Funktion Unregelmäßigkeiten in den Ausgabebits zu erreichen und es damit einem Angreifer zu erschweren, Informationen aus den Ausgabebits zu entnehmen.

Die Ausgabefunktion F nutzt dabei eine Untermenge der Bits aus dem internen Zustand eines LFSR. Ein Angreifer, der die verwendete Funktion F und die Feedback-

Funktion L des LFSR kennt, kann die folgenden Gleichungen aufstellen:

$$\begin{aligned} z_1 &= F(S^{(0)}) \\ z_2 &= F(L(S^{(0)})) \\ &\vdots \\ z_t &= F(L^t(S^{(0)})) \end{aligned}$$

Nach [FA03] gilt, dass jede erzeugte Gleichung nach diesem Schema uns neue Informationen liefert, welches zum Lösen des Gleichungssystems benötigt wird, solange die bisherigen nicht ausreichend sollten.

Wir haben schon im Kapitel 2.1 angeführt, dass einige Bits aus dem Initialisierungszustand bis zum Zeitpunkt n mehrmals auftreten können. Die Zahl n gibt dabei die Länge des LFSR wieder. Wir werden hier zeigen, dass diese Eigenschaft das System anfällig auf Angriffe macht und geben hier zunächst ein kleines Beispiel.

Beispiel 15 *Wir nehmen den Schlüsselstromgenerator aus Beispiel 2. Wir haben ein LFSR der Länge 4 mit internem Zustand* $S^{(t)} = (s_1^{(t)}, s_2^{(t)}, s_3^{(t)}, s_4^{(t)})$ *zum Zeitpunkt t. Die nichtlineare Ausgabefunktion ist gegeben durch* $F(S^{(t)}) := s_1^{(t)} * s_3^{(t)}$. *Die Feedback-Funktion L lautet* $L(S^{(t)}) := s_1^{(t)} \oplus s_4^{(t)}$. *Der Angreifer konnte folgende Ausgabebits ausfindig machen:* 110100100100000.

Nun lassen sich folgende Gleichungen aufstellen:

$$\begin{aligned} z_0 &= F(S^{(0)}) &&= s_1^{(0)} * s_3^{(0)} \\ z_1 &= F(S^{(1)}) &&= s_1^{(1)} * s_3^{(1)} \\ &\vdots \\ z_t &= F(S^{(t)}) &&= s_1^{(t-1)} * s_3^{(t-1)} \end{aligned}$$

Mit dem Wissen über die verwendete Feedback-Funktion können wir sagen, dass $s_4^{(t+1)} = L(S^{(t)}) = s_1^{(t)} \oplus s_4^{(t)}$. *Weiter gilt* $s_i^{(t+1)} = s_{i+1}^{(t)}$ *für* $i \in \{1, 2, 3\}$ *aufgrund des Shiftens. Nun lässt sich das Gleichungssystem umschreiben:*

$$\begin{aligned} z_1 &= s_1^{(0)} * s_3^{(0)} \\ z_2 &= s_2^{(0)} * s_4^{(0)} \\ z_3 &= s_3^{(0)} * (s_1^{(0)} \oplus s_4^{(0)}) \\ z_4 &= s_4^{(0)} * (s_1^{(1)} \oplus s_4^{(1)}) = s_4^{(0)} * (s_2^{(0)} \oplus (s_1^{(0)} \oplus s_4^{(0)})) \\ &\vdots \end{aligned}$$

Nachdem das Gleichungssystem aufgestellt wurde, werden die Ausgabebits z_t *eingesetzt. In diesem Beispiel reichen schon 2 Ausgabebits, um den geheimen Schlüssel* $\mathcal{K} = S^{(0)}$ *des Schlüsselstromgenerators zu erhalten. Wir bekommen folgende Ergebnisse:*

$$\begin{aligned} 1 &= s_1 * s_3 \quad \Rightarrow s_1 = 1, s_3 = 1 \\ 1 &= s_2 * s_4 \quad \Rightarrow s_2 = 1, s_4 = 1 \end{aligned}$$

Der Angreifer hat in diesem Beispiel mit nur 2 Ausgabebits den Schlüssel $\mathcal{K} = S^{(0)} = (1, 1, 1, 1)$ *durch einen algebraischen Angriff ausfindig machen können.*

Dieses ist ein einfaches Beispiel und es war vorherzusehen, dass es nicht genügend vor Angriffen gesichert ist. In realen Systemen werden andere Filterfunktionen genutzt, die höheren Grades als 2, wie in Beispiel 15, sind.

Wir erhalten ein Gleichungssystem, bei welchem der Grad durch die nichtlineare Funktion F definiert wird. Der maximale Grad einer Funktion $F(x_1, \ldots, x_k)$ beträgt k, also $deg(F) \leq k$. Ein Grad von genau k würde dann zu Stande kommen, wenn alle Eingaben miteinander multipliziert werden würden. Aus dem Grund, dass $x_1^2 = x_1$ in unserem Definitionsbereich gilt, können keine Funktionen höheren Grades existieren.

Wird vom ungünstigsten Fall ausgegangen, haben wir eine nichtlineare Funktion k-ten Grades. Besitzt das verwendete LFSR eine Länge von $n > k$, so kann ein Angreifer ein Gleichungssystem aufstellen. Wenn n^k Ausgabebits zur Verfügung stehen, kann das erhaltene Gleichungssystem mit der Linearisierungsmethode in $n^{k*\omega}$ Operationen gelöst werden. Noch schnellere Ergebnisse kann man mit Hilfe von Gröber Basen erwarten. Bei genügend vielen Ausgabebits erwartet man hier $\mathcal{O}(n^{\lfloor \frac{k+1}{2} \rfloor * \omega})$. XL liefert eine Komplexität von etwa $\mathcal{O}(\binom{n}{n/m^{1/K}}^{\omega})$. Je nach Anzahl gegebener Gleichungen m kann der jeweilige Algorithmus benutzt werden.

Aus Kapitel 3 wissen wir, dass die Komplexität polynomiell von der Anzahl der Variablen abhängt, aber exponentiell vom Grad d der unterliegenden Funktion F. Aus diesem Grund entstand ein neuer Ansatz, der versucht den maximalen Grad aller Gleichungen so weit wie möglich zu senken.

Zum Zeitpunkt t liegt die Gleichung $f(S^{(t)}) \oplus z_t = 0$ vor. Der Grad dieser Gleichung wird durch den Grad der Funktion F bestimmt. Es besteht die Möglichkeit den Grad dieser Gleichung zu reduzieren, indem wir eine Funktion g an beide Seiten der Gleichungen multiplizieren. Dabei muss gelten $deg(g) < deg(f)$ und $f * g \equiv 0$, um somit einen Vorteil dadurch zu verschaffen. Denn nun müssen wir ein System kleineren Grades lösen mit $f(S^{(t)}) * g(S^{(t)}) = 0$ für den Fall $z_t = 1$. Für den Fall $z_t = 0$ muss analog eine Funktion h gefunden werden, bei der $deg(h) < deg(f)$ und $(f(S^{(t)}) + 1) * h(S^{(t)}) = 0$ gilt. Solch eine Funktion g wird auch Annihilator von f genannt.

Wir bekommen durch Multiplikationen der Annihilatoren folgende Gestalt der Gleichungen:

$$\begin{array}{lll} f(S^{(t)}) * g(S^{(t)}) & = 1 & \text{falls } z_t = 1 \\ (f(S^{(t)}) + 1) * h(S^{(t)}) & = 0 & \text{falls } z_t = 0 \end{array}$$

$$\begin{array}{lll} \rightarrow \quad g(S^{(t)}) & = 0 & \\ h(S^{(t)}) & = 0 & \end{array}$$

Wir müssen weiter analysieren, unter welchen Bedingungen solche Funktionen g und h gefunden werden können. Zum anderen fragt sich, welche obere und untere Grenzen wir für den Grad d der Funktionen g beziehungsweise h erwarten.

In [AI04] wurden folgende Theoreme gezeigt und bewiesen, die uns hier weiterbringen: Sei $1_f := \{x \in \{0,1\}^n \mid f(x) = 1\}$. Analog lässt sich 0_f definieren. Sei folgende Menge $An(f) := \{g \mid f * g \equiv 0\}$ gegeben. Dann gelten folgende Thesen:

- $g \in An(f) \Leftrightarrow 1_f \subseteq 0_g$
- Sei g gegeben mit Grad $\leq d$, dann gilt: $2^{n-d} \leq |0_g| \leq (2^d - 1) * 2^{n-d}$.

Mit den beiden Punkten lässt sich sagen, ob eine Funktion g mit $deg(g) \leq d$ existieren kann, so dass $f * g \equiv 0$ gilt.

Nach [Pas] existiert immer eine solche Funktion g mit Grad $\lceil \frac{k}{2} \rceil$, so dass $f * g \equiv 0$ gilt, wobei n die Anzahl der verwendeten Variablen im Gleichungssystem ist. Nur das Finden solcher Funktionen g mit $deg(g) = d$ ist nicht ohne Kosten erreichbar. Durch Anwendung des Gauss Algorithmus an die Matrix $RM^f(d, n)$ erhalten wir die Komplexität:

$$2^{n-1} \left(\sum_{i=0}^{d} \binom{n}{i} \right)^2$$

wobei die Matrix $RM^f(d, n)$ in den Spalten alle möglichen verschiedenen Monome bis zum Grad d und in den Zeilen die Gleichungen stehen, welche eine 1 ausgeben. Diese Komplexität ist nicht zu unterschätzen, denn schon für $d = 7$ und $n = 15$ erhält man eine Komplexität von etwa 2^{43}. Trotzdem können gute Ergebnisse vor allem in der Praxis damit erreicht werden. Es ist auch möglich, dass mehrere solcher Funktionen g und h gefunden werden können, die diese erforderten Bedingungen erfüllen. Sei u die Anzahl solcher Funktionen $g_1, g_2, \ldots g_u$ und $h_1, h_2, \ldots h_u$, so kann man mit nur einer Ausgabe z_t bis zu u linear unabhängige Gleichungen erstellen. Das hat den Vorteil, dass nur $\binom{n}{d}/u$ Schlüsselstrombits benötigt werden.

Eine ähnliche Variante besteht darin eine Funktion g zu finden, welche die Bedingung $deg(f * g) < deg(f)$ und $deg(g) < deg(f)$ erfüllt. Auch hier können wir den maximalen Grad der vorkommenden Monome in den Gleichungen erniedrigen und erhalten bessere Werte für die Komplexität beim Lösen des Gleichungssystems. Nehmen wir an, wir finden eine solche Funktion, welche die erwähnten Bedingungen erfüllt, dann erhalten wir durch Multiplikation folgendes Gleichungssystem:

$$\begin{aligned} f(S^{(0)}) * g(S^{(0)}) &= z_0 * g(S^{(0)}) \\ f(S^{(1)}) * g(S^{(1)}) &= z_1 * g(S^{(1)}) \\ &\vdots \\ f(S^{(t)}) * g(S^{(t)}) &= z_t * g(S^{(0)}) \end{aligned}$$

Dieses Gleichungssystem besteht nun aus Monomen kleineren Grades. Genauer erhalten wir den Grad $\max\{deg(fg), deg(g)\}$.

In [Cou03] ist ein Theorem zur Existenz einer solchen Funktion g aufgestellt worden. Es besagt folgendes: Sei $f : GF(2)^k \to GF(2)$ eine boolesche Funktion. Dann existiert für alle $d, e \in \mathbb{N}$ mit $d + e \geq k$ eine boolesche Funktion g mit Grad e, so dass $deg(fg) = d$. Nachdem die Existenzfrage geklärt ist, müssen noch die Funktionen g für die verschiedenen nichtlinearen Filterfunktionen gefunden werden, um diese dann einzusetzen. Nach Studien wurde festgestellt, dass die schnellste Variante für einen Angriff (vorgestellt in [Cou03]) mit $e \approx k(1/\omega)$ und $d \approx k(1 - 1/\omega)$ erreicht wird. Die Komplexität für den kompletten Angriff liegt bei $\mathcal{O}(n^{k+1})$. Die Anzahl benötigter Schlüsselstrombits liegt bei $\binom{n}{k(1-1/\omega)}$.

Wenn man allerdings die kleinst mögliche Anzahl an Schlüsselstrombits verwenden möchte oder kann, dann sollte man $e = d \approx k/2$ wählen. Dadurch werden dem entsprechend längere Laufzeiten erreicht. Diese liegen bei $\mathcal{O}(n^{\frac{k\omega}{2}})$. Es werden $\binom{n}{\lceil k/2 \rceil}$ Schlüsselstrombits benötigt.

Welche Vorteile eine solche Vorbereitungsmaßnahme, den maximalen Grad der vorkommenden Monome in den Gleichungen zu reduzieren, mit sich bringt, wird bei einem kleinen Beispiel klarer. Nehmen wir die nichtlineare Filterfunktion f aus Toyocrypt (Beispiel 3), bei der $deg(f) = 63$ gilt. Sie nutzt ein LFSR mit Länge $l = 128$. Wenn wir keine

weiteren Maßnahmen vornehmen würden, müssten wir ein lineares Gleichungssystem mit $T \approx \binom{128}{63} \approx 2^{124}$ lösen. Wir erhielten eine Komplexität von $2^{124^\omega} \approx 2^{372}$. Dieser Angriff wäre so völlig unbrauchbar, da wir selbst mit der Brute Force Methode eine niedrigere Komplexität erreichen.

Wenn wir uns aber die nichtlineare Filterfunktion f genauer ansehen, dann kann man erkennen, dass $f(s_0, \ldots, s_{127}) * (1 + s_{23})$ vom Grad 3 ist. Nun haben wir ein Gleichungssystem mit Monome vom Grad ≤ 3 und es gilt $d = 3$ und $e = 1$. Wir hat dadurch eine Komplexität von

$$\mathcal{O}(\binom{n}{d}\binom{n}{e}) + \binom{n}{e}^{\omega} \approx 2^{20}$$

nach der Vorgehensweise in [Cou03]. Zu Toyocrypt existieren weitere Angriffe, jedoch weniger erfolgreich in Hinblick auf die Komplexität. Ein ähnlicher Ansatz in [CM03], welcher den Vorteil $e < d$ nicht nutzt, erhält eine Komplexität von $\mathcal{O}(2^{49})$.

4.2 Anwendung auf LFSR mit nichtlinearem Filtergenerator und Combinergenerator

Die Vorgehensweise bei LFSR mit nichtlinearem Filtergenerator und Combinergenerator ohne zusätzlichen Speicher ist ähnlich dem der Stromchiffren, die nur eine nichtlineare Filterfunktion verwenden. Der Unterschied ist dieser, dass der geheime Schlüssel hier nicht nur der Initialisierungszustand des einen LFSR ist, sondern der aller verwendeten LFSR. Auch hier wird die Filterfunktion genauer untersucht und die gleichen Methoden für einen Angriff wie im Kapitel 4.1 angewendet.

Der Geffe Generator aus Beispiel 4 besitzt Grad 2. Daher muss man für einen Angriff den Grad der auftretenden Monome nicht erniedrigen, da diese bereits mit Grad 2 sehr niedrig sind. Die maximale Anzahl an zu erwartenden Monome liegt bei $\binom{n}{1}+\binom{n}{2} < \frac{n^2}{2}+n$. Wenn wir diese Anzahl an Schlüsselstrombits haben, können wir mit der Linearisierungmethode den Schlüssel $\mathcal{K}$ mit etwa $(\frac{n^2}{2} + n)^\omega$ Operationen berechnen. Wir erhalten also einen polynomiellen Angriff auf den Geffe Generator.

Ein möglicher Ansatz die Sicherheit dieser Stromchiffren zu erhöhen, wäre den Grad d der verwendeten Filterfunktionen zu erhöhen, wodurch die Effizienz dieser Stromchiffre leidet, daher greift man auf eine andere Variante zurück. Es wird versucht durch zusätzlichen Speicher Einfluss auf die Ausgabe der nichtlinearen Filterfunktion zu nehmen und Unregelmäßigkeiten hervorzurufen. Siehe dazu auch Kapitel 2.2.2 für die Funktionsweise solcher Stromchiffren.

Das Ziel ist wieder ein Gleichungssystem mit polynomieller Größe und niedrigem Grad zu erhalten, durch welches wir den geheimen Schlüssel $\mathcal{K}$ der Stromchiffre berechnen können. Wir nennen eine Stromchiffre, welche k LFSR und zusätzlich l Bits aus einem Speicher nutzt, einen (k, l) Combiner. Die nichtlineare Filterfunktion f besitzt somit als Eingabe die Ausgänge der LFSR $x_1^{(t)}, \ldots, x_k^{(t)}$ sowie die Speicherbelegungen $c_1^{(t)}, \ldots, c_l^{(t)}$ zum Zeitpunkt t. Das Gleichungssystem, welches wir erhalten, sieht wie folgt aus:

$$\begin{aligned} z_0 &= F(x_1^{(0)}, \ldots, x_k^{(0)}, c_1^{(0)}, \ldots, c_l^{(0)}) \\ z_1 &= F(x_1^{(1)}, \ldots, x_k^{(1)}, c_1^{(1)}, \ldots, c_l^{(1)}) \\ &\vdots \\ z_t &= F(x_1^{(t)}, \ldots, x_k^{(t)}, c_1^{(t)}, \ldots, c_l^{(t)}) \end{aligned}$$

Wir sehen, dass wir nun ein Gleichungssystem haben, in welchem die Unbekannten aus den geheimen Schlüssel $\mathcal{K}$ bestehen und neuerdings auch aus einem mit verwendetem Speicher. Dadurch haben wir mehr Unbekannte erhalten, was eine höhere Komplexität beim Lösen mit sich bringt.

Wenn diese Gleichungen genauer betrachtet werden, sieht man ein Problem. Es können nicht wirklich die Werte $c_1^{(t)}, \ldots, c_l^{(t)}$ durch die Werte $c_1^{(0)}, \ldots, c_l^{(0)}$ beschrieben werden. Dadurch haben wir die Problematik, dass nach t Takten die Anzahl der Unbekannten bei $n + t * l$ liegt, da nach jedem Takt l neue Unbekannte hinzugefügt werden. Da $n + t * l > t$ gilt, erhalten wir immer mehr Unbekannte als Gleichungen. Somit kann dieses Gleichungssystem, welches nach diesem Schema aufgebaut ist, niemals gelöst werden.

Es muss ein Weg gefunden werden, welcher verhindert, dass neue Unbekannte im System entstehen. Der Folgezustand der Speicherbelegungen $(c_1^{(0)}, \ldots, c_l^{(0)})$ hängt von dem momentanem Zustand und der Ausgänge der LFSR $x_1^{(0)}, \ldots, x_k^{(0)}$ ab. Es könnten nun alle Speicherbelegungen zu allen Zeitpunkten durch die Ausgänge der LFSR und dem Initialisierungszustand des Speichers angegeben werden. Wir erhalten:

$$\begin{aligned}
(c_1^{(1)}, \ldots, c_l^{(1)}) &= C(x_1^{(0)}, \ldots, x_k^{(0)}, c_1^{(0)}, \ldots, c_l^{(0)}) \\
(c_1^{(2)}, \ldots, c_l^{(2)}) &= C(x_1^{(1)}, \ldots, x_k^{(1)}, c_1^{(1)}, \ldots, c_l^{(1)}) \\
&\vdots \\
(c_1^{(t)}, \ldots, c_l^{(t)}) &= C(x_1^{(t-1)}, \ldots, x_k^{(t-1)}, c_1^{(t-1)}, \ldots, c_l^{(t-1)})
\end{aligned}$$

Um keine weiteren Bits aus dem Speicher verwenden zu müssen, können die Zustände der LFSR bis zum Zeitpunkt t mit in die Funktionen eingebracht werden. Wir können dann folgende Gleichung aufstellen:

$$(c_1^{(t)}, \ldots, c_l^{(t)}) = C_t(x_1^{(0)}, \ldots, x_k^{(0)}, \ldots, x_1^{(t-1)}, \ldots, x_k^{(t-1)}, c_1^{(0)}, \ldots, c_l^{(0)}) \tag{21}$$

Mit dieser Überlegung lässt sich die Ausgabe der Stromchiffre durch eine Funktion darstellen, welche als Eingänge den Initialisierungszustand der Speicherbits und die Ausgänge der LFSR besitzt. Die Schlüsselstrombits z_t können dann wie folgt definiert werden:

$$z_t = F_t(x_1^{(0)}, \ldots, x_k^{(0)}, \ldots, x_1^{(t)}, \ldots, x_k^{(t)}, c_1^{(0)}, \ldots, c_l^{(0)}) \tag{22}$$

Dieser Ansatz besitzt den Vorteil, dass nur noch $n + l$ Unbekannte auftreten. Somit kann durch genügend viele Schlüsselstrombits ein lösbares Gleichungssystem aufgestellt werden. Aber auch hier besteht eine Problematik. Diese liegt an den Funktionen F_t. Durch die vielen Eingabeparameter können dem entsprechend auch große Werte für den Grad der Funktionen F_t erwartet werden. Im ungünstigsten Fall kann die Anzahl der Monome bis 2^{n+l} sein und wäre somit exponentiell in n.

Der nächste Ansatz sollte möglichst kein Gleichungssystem liefern, welches exponentiell in der Anzahl der Unbekannten wächst oder einen hohen Grad besitzt. Die Lösung erreicht man durch Ad-hoc Gleichungen [Arm]. Hier wird ein Gleichungssystem aufgestellt, welches nur Unbekannte aus dem Schlüssel- und Ausgabebits enthält.

Die Idee besteht darin, die Speicherbits komplett aus dem Gleichungssystem zu entfernen. Sei ein (k, l) Combiner gegeben. Sei $\vec{x}^t = (x_1^{(t)}, \ldots, x_k^{(t)})$ und $\vec{c}^t = (c_1^{(0)}, \ldots, c_l^{(0)})$. Dann lässt sich das erste Ausgabebit wie folgt berechnen:

$$z_0 = F(\vec{x}^0, \vec{c}^0)$$

Nun berechnen wir den Folgezustand unserer Speichers mit

$$\vec{c}^1 = C(\vec{x}^0, \vec{c}^0)$$

Nehmen wir nun an, dass die Filterfunktion F linear im Feedbackbit ist. Dann gilt $F(\vec{x}, \vec{c}) = F'(\vec{x}) \oplus \vec{c}$. Dann kann das Ausgabebit z_1 auch wie folgt berechnet werden:

$$z_1 = F(\vec{x}^1, C(\vec{x}^0, z_0 \oplus \vec{x}^0)$$

Nach diesem Schema können wir ein Gleichungssystem aufstellen, welches nicht von dem internen Zustand des verwendeten Speichers abhängt. Aber man darf eigentlich nicht davon ausgehen, dass die Filterfunktion linear bezüglich des Feedbackbits ist, sondern nichtlinear für alle Zeitpunkte. Daher bekommen wir folgende Gleichungen:

$$\prod_{\vec{c} \in \{0,1\}^l} \{F(\vec{x}^t, c) \oplus z_t\} = 0$$

Wir wissen, dass ein $\vec{c}$ in jedem Takt t existieren muss, so dass die Gleichung erfüllt wird.

Sei nun $\breve{F} : (\{0,1\}^k)^r \to \{0,1\}$ so definiert, dass

$$\breve{F}(\vec{x}_1, \ldots, \vec{x}_r) = 0 \quad \forall (\vec{x}_1, \ldots, \vec{x}_r) \in (\{0,1\}^k)^r$$

gilt, wenn der dazugehörige (k,l) Combiner die Ausgabe $\vec{z}$ zu diesem $\vec{x}$ ausgibt. Dabei sind $(\vec{x}_1, \ldots, \vec{x}_r)$ die r-mal hintereinander ausgeführten Ausgaben der LFSR an die Stromchiffre, welche mit diesen Eingaben $z_1, \ldots z_r$ zu einem Initialwert des Speichers ausgibt.

Nach dem Theorem von [AK03] existiert für alle (k,l) Combiner eine Ad-hoc Gleichung $\breve{F} \neq 0$ vom Grad $d \leq \lceil k(l+1)/2 \rceil$ mit

$$0 = \breve{F}(x_1^{(t)}, \ldots, x_k^{(t)}, \ldots, x_1^{(t+l)}, \ldots, x_k^{(t+l)}, z_t, \ldots, z_{t+l})$$

Da solche Gleichungen immer existieren für beliebige (k,l) Combiner, können wir ein Gleichungssystem aufstellen, womit der geheime Schlüssel $\mathcal{K}$ ausgerechnet werden kann.

$$\begin{aligned} 0 &= \breve{F}(x_1^{(0)}, \ldots, x_k^{(0)}, \ldots, x_1^{(r-1)}, \ldots, x_k^{(r-1)}, z_0, \ldots, z_{r-1}) \\ 0 &= \breve{F}(x_1^{(1)}, \ldots, x_k^{(1)}, \ldots, x_1^{(r)}, \ldots, x_k^{(r)}, z_1, \ldots, z_r) \\ &\vdots \\ 0 &= \breve{F}(x_1^{(t)}, \ldots, x_k^{(t)}, \ldots, x_1^{(t+r)}, \ldots, x_k^{(t+r)}, z_t, \ldots, z_{t+r}) \end{aligned}$$

wobei r so gewählt werden soll, das solch eine Funktion existiert.

Dieser Ansatz bringt die Vorteile, dass die Anzahl der Monome im Gleichungssystem polynomiell in n sind. Zum anderen haben die Ad-hoc Gleichungen eine obere Grenze bezüglich ihren Grades. Sie ist bestimmt durch $\lceil k(l+1)/2 \rceil$. Dadurch haben wir ein Verfahren bekommen, wie wir beliebige (k,l) Combiner in polynomieller Zeit angreifen können.

In [AK03] ist ein Algorithmus angegeben, womit alle ad-hoc Gleichungen $\breve{F}$ mit Grad $\leq d$ mit Hilfe von r hintereinander folgenden Takten gefunden werden, so dass

$$0 = \breve{F}(x_1^{(t)}, \ldots, x_k^{(t)}, \ldots, x_1^{(t+r)}, \ldots, x_k^{(t+r)}, z_t, \ldots, z_{t+r}) \tag{23}$$

gilt.

Die obere Grenze von $\lceil k(l+1)/2 \rceil$ wird dabei oft unterboten. Dazu schauen wir uns die Stromchiffre E_0 aus dem Bluetoothsystem (Beispiel 5) an. Hier wird ein (4,4) Combiner genutzt. Also haben wir $k = l = 4$. Nach dem Theorem existieren ad-hoc Gleichungen von Grad ≤ 10. Aber in der Praxis können wir schon durch $r = 4$ hintereinander folgende Takte ad-hoc Gleichungen vom Grad 4 erreichen.

Die Anzahl aller Monome mit $n = 128$ und $d = 4$ liegt bei $\sim 128^4 = 2^{24}$. Mit 2^{24} Schlüsselstrombits z_t kann mit Hilfe der Linearisierungsmethode eine Lösung bestimmt werden. Wir brauchen dann $(2^{24})^\omega \approx 2^{67}$ Operationen.

Mit dem Ansatz Funktionen g zu finden, so dass diese ad-hoc Gleichung $\breve{F}$ die Bedingung $deg(\breve{F} * g) \leq deg(\breve{F})$ und $deg(g) < deg(\breve{F})$, kann sogar eine noch kürzere Anzahl an Operationen erwartet werden. In [Cou03] wurde dieser Ansatz getätigt und es wurde folgendes für einen Angriff auf E_0 ermittelt:

- Es werden $2^{23.4}$ hintereinander folgende Schlüsselstrombits benötigt
- Es werden 2^{37} Bits Speicher (= 16 Gigabytes) benötigt
- Der geheime Schlüssel $\mathcal{K}$ kann in $\mathcal{O}(2^{49})$ Operationen berechnet werden.

Dies ist nach meinem Wissen der schnellste Angriff auf den Schlüsselstromgenerator E_0.

Es ist wichtig zu wissen, dass das Bluetooth Verschlüsselungssystem E_0 den geheimen Schlüssel $\mathcal{K}$ alle 2745 Takte wechselt. Das bedeutet, dass maximal $2745 \approx 2^{11}$ Schlüsselstrombits erhalten werden können. Der schnellste Angriff auf E_0 benötigt $2^{23.4}$ Schlüsselstrombits und kann daher nicht verwendet werden. Es besteht aber die Möglichkeit, dass ein r gefunden werden kann, so dass ein geringerer Grad für das erhaltene Gleichungssystem gefunden wird. Wir wissen nur, dass wir bei $r = 4$ für die Funktion im Gleichungssystem den Grad 4 erhalten. Es ist jedoch nicht bewiesen, ob ein r existiert, so dass eine Funktion mit Grad 3 gefunden wird. Dann bräuchte man sowohl weniger Schlüsselstrombits als auch weniger Operationen für einen Angriff.

Es existiert auch ein BDD (Binary Decision Diagram) basierender Angriff, welcher mit nur 128 Schlüsselstrombits auskommt. Vorgestellt wurde der Angriff in [Kra02]. Aufgrund der hohen Laufzeit ($\approx 2^{77}$) wird hier nicht weiter darauf eingegangen.

4.3 Anwendung auf unregelmäßig getaktete LFSR

Ein Angriff auf eine Stromchiffre, welche unregelmäßig getaktete LFSR nutzt, ist problematischer als die bisherigen Fälle. Denn es ist schwer den Takt bestimmenden LFSR anzugreifen, da seine Ausgaben nicht von Außen entnommen werden können. Den geheimen Schlüssel kann man erst dann bestimmen, wenn man bereits das LFSR der Datenkontroll-Komponente mitsamt seinen Initialisierungszustand und Ausgaben kennt.

In [CM03] wurden Methoden vorgestellt, wie die Taktkontroll-Komponente in einem Angriff umgangen werden kann. Die Idee wird hier nochmals erläutert.

Im Allgemeinen geht man bei Stromchiffren mit unregelmäßig getakteten LFSR so vor, dass der Initialisierungszustand des Takt bestimmenden LFSR geraten wird. Anschließend wird dann das Daten erzeugende LFSR nach bisherigen Methoden angegriffen. Das bedeutet, dass eine Funktion g gefunden wird, bei welcher $deg(g) < deg(f)$ und $deg(f * g) < deg(f)$ gilt, wobei f die nichtlineare Filterfunktion ist. Wenn das Takt bestimmende LFSR eine Länge von l besitzt, dann muss der Angriff im ungünstigsten Fall 2^l mal wiederholt werden. Das bedeutet, dass die Komplexität mit 2^l multipliziert wird.

Wir führen die Notation ein, dass das Takt bestimmende LFSR durch $LFSR_a$ und das Daten erzeugende LFSR durch $LFSR_b$ beschrieben wird.

Eine andere Variante ist es statt den Angriff 2^l mal zu wiederholen, das $LFSR_a$ $2^l - 1$ mal weiter zu takten und dann das Gleichungssystem mit bekannten Methoden zu lösen, als wäre die Datenerzeugungs-Komponente synchron getaktet. Dies wird aus dem Grund erreicht, da wir wissen, dass das $LFSR_b$ nach einer Periodenlänge den gleichen Wert ausgibt, und somit ausgewählte Ausgabebits wie aus synchron getakteten LFSR behandelt werden können.

Wenn man allerdings nur jedes $(2^l - 1)$-ten Schlüsselstrombit verwendet, benötigt man mehr Schlüsselstrombits. Genauer multipliziert sich die Anzahl der benötigten Schlüsselstrombits mit 2^l, wenn man einen sehr langen Abschnitt an Schlüsselstrombits hat. Alternativ, wenn die Ausgabebits z_t an besonderen Positionen

$$t = a + b * (2^l - 1) \text{ mit } a, b \in \mathbb{N} \tag{24}$$

bekannt sind, ist keine erhöhte Anzahl an Schlüsselstrombits nötig.

Es wird nun die Stromchiffre LILI-128 vorgestellt und weiter wird dann ein Angriff auf diese Stromchiffre beschrieben.

Beispiel 16 *LILI-128*
Der Schlüsselstromgenerator LILI-128 wurde beim NESSIE-Projekt (New European Schemes for Signatures, Integrity, and Encryption) eingereicht. LILI-128 besteht aus 2 LFSR, welche jeweils die Funktionen der Taktsteuerung und der Datenerzeugung haben.

Das Takt bestimmende LFSR besitzt die Länge $l_a = 39$ *und das Daten erzeugende LFSR die Länge* $l_b = 89$. *Die Feedback-Polynome der LFSR sind wie folgt definiert:*

$$L_a(X) := X^{39} + X^{35} + X^{33} + X^{31} + X^{17} + X^{15} + X^{14} + X^2 + 1$$
$$L_b(X) := X^{89} + X^{83} + X^{80} + X^{55} + X^{53} + X^{42} + X^{39} + X + 1$$

Die nichtlineare Filterfunktion f_a *des Takt bestimmenden LFSR ist definiert durch*

$$\begin{array}{rcl} f_a : \mathbb{F}_2 & \longrightarrow & \{1, 2, 3, 4\} \\ (s_{12}, s_{20}) & \longmapsto & 2s_{12} + s_{20} + 1 \end{array}$$

Die Ausgabe der f_a *bestimmt wie oft das LFSR aus der Datenerzeugung getaktet werden soll bis die Ausgabe erfolgt. In diesem Fall wird ein bis viermal getaktet und dann folgt ein neues Schlüsselstrombit.*

Die nichtlineare Filterfunktion f_b *des Daten erzeugenden LFSR nutzt den Inhalt der Zellen* $0, 1, 3, 7, 12, 20, 30, 44, 65, 80$. *Sei*

$$x = (x_1, \ldots, x_{10}) = (s_0, s_1, s_3, s_7, s_{12}, s_{20}, s_{30}, s_{44}, s_{65}, s_{80}) \tag{25}$$

,dann ist f_b *wie folgt definiert:*

$$
\begin{aligned}
f_b(x_1,\ldots,x_{10}) &= x_2 + x_3 + x_4 + x_5 \\
&+ x_6x_7 + x_1x_8 + x_2x_8 + x_1x_9 + x_3x_9 + x_4x_{10} + x_6x_{10} \\
&+ x_3x_7x_9 + x_4x_7x_9 + x_6x_7x_9 + x_3x_8x_9 + x_6x_8x_9 \\
&+ x_4x_7x_{10} + x_5x_7x_{10} + x_6x_7x_{10} + x_3x_8x_{10} \\
&+ x_4x_8x_{10} + x_2x_9x_{10} + x_3x_9x_{10} + x_4x_9x_{10} + x_5x_9x_{10} \\
&+ x_3x_7x_8x_{10} + x_5x_7x_8x_{10} + x_2x_7x_9x_{10} + x_4x_7x_9x_{10} \\
&+ x_6x_7x_9x_{10} + x_1x_8x_9x_{10} + x_3x_8x_9x_{10} + x_4x_8x_9x_{10} \\
&+ x_6x_8x_9x_{10} + x_4x_6x_7x_9 + x_5x_6x_7x_9 + x_2x_7x_8x_9 + x_4x_7x_8x_9 \\
&+ x_4x_6x_7x_9x_{10} + x_5x_6x_7x_9x_{10} + x_3x_7x_8x_9x_{10} \\
&+ x_4x_7x_8x_9x_{10} + x_4x_6x_7x_8x_9 + x_5x_6x_7x_8x_9 \\
&+ x_4x_6x_7x_8x_9x_{10} + x_5x_6x_7x_8x_9x_{10}
\end{aligned}
$$

Weitere Details befinden sich in [Lan06].

Nun folgt eine Beschreibung für einen Angriff auf die Stromchiffre LILI-128 aus Beispiel 16. Unser erster Ansatz ist es, eine Funktion g zu finden, die ein Gleichungssystem mit Monomen niedrigeren Grades als f_b liefert. Betrachtet man sich den Teil der Filterfunktion f_b vom Grad 5 und 6, so erkennt man, dass in jedem Monom das Produkt x_7x_9 enthalten ist. Durch Faktorisieren erhalten wir

$$x_7x_9(x_4x_6x_{10} + x_5x_6x_{10} + x_3x_8x_{10} + x_4x_8x_{10} + x_4x_6x_8 + x_5x_6x_8 + x_4x_6x_8x_{10} + x_5x_6x_8x_{10})$$

Durch diese Erkenntnisse stellt man fest, dass f_b multipliziert mit $(x_7 \oplus 1)$ oder $(x_9 \oplus 1)$ ein Ausdruck mit Grad 5 liefert. Man stellt sogar fest, dass $f_b(x_9 \oplus 1)$ noch weiter faktorisiert werden kann. Es ist erkennbar, dass alle Terme vom Grad 4 und 5 dieses Ausdruckes den Faktor x_{10} enthalten. Somit erhalten wir zusammenfassend mit $g(x) = (x_9 \oplus 1) * (x_{10} \oplus 1)$ die gesuchte Funktion mit $deg(fg) = 4$ und $deg(g) = 2$.

Durch weitere Analysen wurde in [CM03] festgestellt, dass 14 verschiedene Funktionen g_i existieren, so dass $deg(fg) = 4$ und $deg(g) = 4$ gilt. Mit dieser Erkenntnis teilt sich die Anzahl der benötigten Schlüsselstrombits um einen Faktor 14.

Nachdem wir beobachtet haben, dass wir ein Gleichungssystem mit Monomen vom Grad ≤ 4 erhalten und das LFSR aus der Datengenerierung eine Länge von 89 besitzt, folgt für die Anzahl an auftretenden Monome $M = \sum_{i=1}^{4} \binom{89}{4} \approx 2^{21}$. Wenden wir beispielsweise die Linearisierungsmethode an, so benötigen wir $m = (2^{21})/14 \approx 2^{18}$ Schlüsselstrombits.

Wenn wir den Ansatz nehmen, dass wir den Initialisierungszustand der Taktkontroll-Komponente raten, dann erhalten wir eine Komplexität von

$$2^{39} * 2^{21*\omega} \approx 2^{89} \tag{26}$$

Wenn man allerdings die zweite Variante nutzt und jedes $(2^{39}-1)$-te Schlüsselstrombit betrachtet, so benötigt man keine 2^{39} Wiederholungen des Angriffs mehr für das Erraten des Inhalts der Taktkontroll-Komponente, und erhält somit eine Komplexität von

$$2^{21*\omega} \approx 2^{50} \quad \text{für } \omega \approx 2.376 \tag{27}$$

Die Anzahl an benötigten Schlüsselstrombits bei der letzteren Variante liegt bei $2^{18} * 2^{39} = 2^{57}$ hintereinander folgenden Schlüselstrombits. Wenn man jedoch ausgewählte

Schlüsselstrombits mit Positionen nach dem Schema in (24) kennt, dann benötigt man hier auch nur 2^{18} Schlüsselstrombits.

In [Cou03] wählte man jedoch nur eine Funktion g, bei welcher $deg(fg) = 4$ und $deg(g) = 2$ gilt, und erhält eine höhere Anzahl von $2^{21.3+39}$ hintereinander folgenden Schlüsselstrombits. Aber da $deg(g) = 2 < 4$ gilt, erreicht man eine geringere Laufzeit. Mit diesem Ansatz ist es möglich einen Angriff auf die Datenerzeugungs-Komponente mit einer Laufzeit von $\mathcal{O}(2^{31})$ auszuführen. Weiter wurde in diesem Paper hingewiesen, dass nach Kenntnis über den geheimen Schlüssel $\mathcal{K}_b$ der Datenerzeugungs-Komponente, der geheime Schlüssel $\mathcal{K}_a$ des LFSR aus der Taktkontroll-Komponente mit weniger als 2^{20} Operationen herausgefunden werden kann.

4.4 Vergleich der Angriffe auf die verschiedenen Designsansätze der LFSR

In einem algebraischen Angriff wird ein Gleichungssystem versucht aufzustellen, womit das Lösen dieses Gleichungssystem äquivalent zum Finden des geheimen Schlüssels $\mathcal{K}$ ist. Es wurden die verschiedenen Ansätze für das Aufstellen eines Gleichungssystem je nach gegebenem Design der Stromchiffre vorgestellt.

In Stromchiffren mit nichtlinearer Filterfunktion kann der maximale vorkommende Grad der Monome durch die lineare Filterfunktion noch weiter erniedrigt werden. Das Gleichungssystem

$$\begin{aligned} z_0 &= F(S^{(0)}) \\ z_1 &= F(S^{(1)}) \\ &\vdots \\ z_t &= F((S^{(t)})) \end{aligned}$$

kann durch eine passende Funktion g mit $deg(f * g) \leq deg(f)$ und $deg(g) \leq deg(f)$ auch wie folgt aufgestellt werden:

$$\begin{aligned} f(S^{(0)}) * g(S^{(0)}) &= z_0 * g(S^{(0)}) \\ f(S^{(1)}) * g(S^{(1)}) &= z_1 * g(S^{(1)}) \\ &\vdots \\ f(S^{(t)}) * g(S^{(t)}) &= z_t * g(S^{(t)}) \end{aligned}$$

Das dadurch resultierende Gleichungssystem ist kleineren Grades und kann somit durch geringeren Aufwand durch die vorgestellten Methoden im Kapitel 3 gelöst werden.

Ein ähnlicher Ansatz funktioniert durch Finden von Funktionen g, so dass $f*g \equiv 0$ und $deg(g) < deg(f)$ gilt. Wir erreichen hiermit auch eine Erniedrigung des Ausgangsgrades des Gleichungssystems.

Wenn jedoch die Stromchiffre unregelmäßig getaktet ist, wird es problematischer. Wir kennen keine Ausgaben der Taktkontroll-Komponente, sondern nur die Schlüsselstrombits der Datenerzeugungskomponente. Hier rät man entweder die Ausgabe der Taktkontroll-Komponente oder man nimmt nur die Schlüsselstrombits in das Gleichungssystem auf, welches resultiert, nachdem eine Periode des LFSR aus der Taktkontroll-Komponente durchlaufen wurde. Im erstem Fall multipliziert sich die Laufzeit mit 2^l, wobei l die Länge des LFSR der Taktkontroll-Komponente bestimmt. Im letzteren Fall multipliziert sich die Anzahl der Schlüsselstrombits mit 2^l. Wenn es jedoch möglich ist, Schlüsselstrombits auf

bestimmten Positionen zu entnehmen, dann tritt keine Erhöhung der benötigten Anzahl der Schlüsselstrombits ein. Auch bei einem solchen Design stellen wir ein Gleichungssystem durch die nichtlineare Filterfunktion der Datenerzeugung-Komponente auf. Es wird analog zum Design, in welchem nur Schutz durch die nichtlineare Filterfunktion gegeben ist, eine Funktion g gesucht, mit Hilfe derer ein niedrigerer Grad der zu erwartenden Monome erreicht werden soll.

Im Falle, dass ein Combiner vorliegt, muss man jedoch anders vorgehen. Zwar kann analog zu den bisherigen Fällen bei speicherlosen Combiner vorgegangen werden, aber bei Nutzung von zusätzlichem Speicher treten einige Hindernisse auf. Würden wir analog wie bisher vorgehen, bekämen wir eine Gleichungssystem nach folgendem Schema:

$$\begin{aligned} z_0 &= F(x_1^{(0)}, \ldots, x_k^{(0)}, c_1^{(0)}, \ldots, c_l^{(0)}) \\ z_1 &= F(x_1^{(1)}, \ldots, x_k^{(1)}, c_1^{(1)}, \ldots, c_l^{(1)}) \\ &\vdots \\ z_t &= F(x_1^{(t)}, \ldots, x_k^{(t)}, c_1^{(t)}, \ldots, c_l^{(t)}) \end{aligned}$$

Es ist zwar möglich eine Funktion g zu finden, womit wir den Grad verkleinern könnten, aber durch die Speicherstellen $c^{(0)}, \ldots c^{(t)}$ erhalten wir mehr Variablen als Gleichungen zu jedem Zeitpunkt t. Daher wird hier ein anderer Ansatz gewählt.

Es existiert eine Funktion $\breve{F}$, welche r hintereinander folgende Schlüsselstrombits als Eingabe enthält und mit der folgendes Gleichungssystem zu jedem Zeitpunkt t aufgestellt werden kann:

$$0 = \breve{F}(x_1^{(t)}, \ldots, x_k^{(t)}, \ldots, x_1^{(t+r)}, \ldots, x_k^{(t+r)}, z_t, \ldots, z_{t+r}) \tag{28}$$

Diese Funktion ist völlig unabhängig von den zusätzlich verwendeten Speicherstellen und besitzt einen maximalen Grad von $\lceil k(l+1)/2 \rceil$. Da die Anzahl der Variablen nun polynomiell wächst und der maximal vorkommende Grad eines Monoms begrenzt ist, kann ein Gleichungssystem aufgestellt und wie bisher gelöst werden.

Nun folgt eine Übersichtstabelle von algebraischen Angriffen auf vorgestellte Stromchiffren, in der die benötigten Schlüsselstrombits und Laufzeiten eingetragen sind.

Kryptosystem	Toyocrypt		LILI-128			E_0		
n	128	128	39	39	39	128	128	128
d	3	3	4	4	4	4	4	4
Attacke	[CM03]	[Cou03]	[CM03]	[CM03]	[Cou03]	[AK03]	[Arm04b]	[Cou03]
Data	2^{18}	2^{18}	2^{18}	2^{57}	2^{60}	2^{23}	2^{23}	2^{24}
Memory	2^{37}	2^{14}	2^{43}	2^{43}	2^{24}	2^{46}	2^{37}	2^{37}
Komplexität	2^{49}	2^{20}	2^{96}	2^{57}	2^{37}	2^{68}	2^{54}	2^{49}

Tabelle 3: Übersichtstabelle aller vorgestellten algebraischen Angriffe

4.5 Mögliche Gegenmaßnahmen gegen algebraische Angriffe

Die Sicherheit einer Stromchiffre beruht meist auf der nichtlinearen booleschen Filterfunktion, welche auch häufig der Angriffspunkt in algebraischen Angriffen ist. Wir wollen hier analysieren, welche Bedingungen an die Filterfunktionen gestellt werden müssen, damit sie optimale Sicherheit gegen algebraische Angriffe bieten.

Mit unregelmäßig getakteten Stromchiffren versucht man Nichtlinearität nicht allein durch die Filterfunktion in den Ausgaben zu erlangen. Aber da es möglich ist, die Taktkontroll-Komponente nach der vorgestellten Methode im Kapitel 4.3 zu ignorieren, beruht die Sicherheit der Stromchiffre gegen Angriffe doch nur auf der Filterfunktion.

Es existieren Angriffe anderer Art wie beispielsweise Korrelationsangriffe auf diese Art von Stromchiffren. so dass bereits Eigenschaften der Filterfunktionen zur Sicherung vor Angriffen definiert wurden. Zum einen sollte die Ausgabe der Filterfunktion $f : F_2^n \to F_2$ ausgewogen sein. Das bedeutet, dass $|\{x \in F_2^n \,|\, f(x) = 1\}| = |\{x \in F_2^n \,|\, f(x) = 0\}|$ gilt, also dass die Anzahl der Einsen und Nullen in der Ausgabe gleichmächtig sind.

Des Weiteren sollte die Filterfunktion einen genügend hohen Grad besitzen, um eine hohe lineare Komplexität zu garantieren. Damit zusammenhängend sollte auch eine hohe Nichtlinearität aus bekannten Gründen vorliegen. Zu dem bereits erwähnten Korrelationsangriffen wurde ein Kriterium für optimale Sicherheit gegen solche Angriffe durch die Korrelation-Immunität definiert. Die Ordnung der Korrelation-Immunität sagt etwas aus, in wiefern die Eingaben mit den Ausgaben statistisch zusammenhängen. Die Ordnung sollte bei 1 liegen, so dass die Ausgabe der Funktion statistisch unabhängig von der Eingabe ist.

Mit diesen vordefinierten Kriterien an die boolesche Filterfunktion ist es für einen algebraischen Angriff nicht ausreichend. Daher wurde der Begriff Algebraische Immunität eingeführt, mit Hilfe dieser Zahl die Sicherheitsstufe einer Funktion gegen algebraische Angriffe definiert werden soll.

Wie gezeigt wird in algebraischen Angriffen versucht eine Funktion g zu finden, mit Hilfe ein Gleichungssystem gewonnen werden soll, welches Monome kleineren Grades besitzt. So kann das eigentliche Gleichungssystem

$$f(S^{(t)}) = z_t$$

durch das folgende ersetzt werden:

$$(g * f)(S^{(t)}) = g(S^{(t)}) * z_t$$

Durch die Eigenschaft $deg(f * g) < deg(f)$ und $deg(g) < deg(f)$ kann das Gleichungssystem mit geringerer Komplexität gelöst werden.

Die Komplexität der algebraischen Angriffe wird vor allem durch den Grad solcher Funktionen g bestimmt. Daher sollten solche Filterfunktion gewählt und verwendet werden, bei welcher möglichst keine Funktion g mit niedrigem Grad gefunden werden kann.

Die Existenz solcher Funktionen g mit kleinerem Grad hängt unmittelbar mit der Existenz von Vielfachen der Filterfunktion mit niedrigem Grad zusammen. Wie bereits im Kapitel 4.1 eingeführt, können Funktionen g und h gefunden werden, so dass $f * g = 0\ \forall z_t = 1$ und $f * h = 0\ \forall z_t = 0$ gilt. In [Can06] wurde der Zusammenhang zwischen diesen beiden Ansätzen erläutert. Falls $g_1(x) * f(x) = g_2(x)$ mit $deg(g_i) \leq d$ für $i = 1, 2$ gilt, erhalten wir durch Multiplikation mit f(x) die Gleichung

$$g_1(x)(f(x))^2 = g_2(x)f(x) = g_1(x)f(x) = g_2(x)$$

, so dass $g_2(x)(1 + f(x)) = 0$ gilt.

Somit kann man folgern, dass wir Funktionen g_1 mit $deg(f * g_1) \leq d$ und $deg(g_1) \leq d$ genau dann finden, wenn wir eine Funktion g_2 finden, bei welcher $f * g_2 = 0$ und $deg(g_2) \leq d$ gilt. Diese Funktionen g, bei denen $g(x)f(x) = 0$ gelten, werden Annihilator von f

genannt. Übernehmen wir die Definitionen vom Kapitel 4.1 mit $An(f) := \{g \mid f * g \equiv 0\}$, so ist die Algebraische Immunität wie folgt definiert.

Definition 5 *Algebraische Immunität*
Die Algebraische Immunität $AI(f)$ einer Funktion f ist der niedrigste Grad einer Funktion aus der Menge $An(f) \cup An(1+f)$. Formal ausgedrückt:

$$AI(f) = \min\{deg(g) \mid (f * g \equiv 0) \vee ((1+f) * g \equiv 0)\}$$

Je höher die Algebraische Immunität einer Funktion ist, desto höher die Komplexität eines algebraischen Angriffes für die Bestimmung des geheimen Schlüssels beziehungsweise des Initialisierungszustands. Daher sollte das Ziel verfolgt werden, dass bei der Wahl einer Filterfunktion ihre Algebraische Immunität maximal ist.

Im Kapitel 4.1 wurde erwähnt wie solche Funktionen $g \in An(f)$ mit $deg(g) \leq d$ gefunden werden können. Die Anzahl von solchen Funktionen in $An(f)$ beträgt 2^k, wobei k die Dimension des Kerns der Matrix $RM^f(d,n)$ ist [Can06, CAB+06]. Sei $wt(f) := |\{x \in F_2^n \mid f(x) = 1\}|$, dann ist der Kern der Matrix nicht trivial, wenn

$$\sum_{i=0}^{d} \binom{n}{i} > wt(f)$$

gilt. Synonym dazu sind Funktionen $g \in AN(1+f)$ mit $deg(g) \leq d$ auffindbar, wenn gilt:

$$\sum_{i=0}^{d} \binom{n}{i} > 2^n - wt(f)$$

Wie zu erkennen gilt für ungerade n, dass nur ausgewogene Funktionen eine optimale Algebraische Immunität erlangen können. Dies war ja bereits eine der Voraussetzungen, die an die nichtlineare Filterfunktion gestellt war. Für gerade n gilt als Voraussetzung für eine optimale Algebraische Immunität, dass die Funktion die Gleichung

$$\sum_{i=0}^{\frac{n}{2}-1} \binom{n}{i} < wt(f) < \sum_{i=0}^{\frac{n}{2}} \binom{n}{i}$$

erfüllt.

Eine obere Grenze für die Algebraische Immunität einer Funktion f ist durch $AI(f) \leq \lceil \frac{n}{2} \rceil$ gegeben, wobei n die Anzahl verwendeter Variablen angibt [Pas, CAB+06]. Also suchen wir Funktionen, die diese obere Grenze der Algebraischen Immunität erlangen. Für den kryptographischen Aspekt ist die Anzahl der gefunden Funktionen $g_i \in An(f) \cup An(1+f)$ mit $deg(g_i) = AI(f)$ nicht sonderlich wichtig. Die Laufzeiten für einen algebraischen Angriff ändern sich dadurch nicht, aber je höher die Anzahl solcher Funktionen sind, desto weniger Schlüsselstrombits werden benötigt, da jedes Schlüsselstrombit für jede Funktion verwendet werden kann und in das Gleichungssystem aufgenommen werden kann [CAB+06].

In [Can06] wurde untersucht, wie die Verteilung der Algebraischen Immunität von ausgewogenen Funktionen ist. Im Test wurden 601.080.390 ausgewogene Funktionen mit 5 Variablen als Eingang auf ihre Algebraische Immunität geprüft. Dabei wurde folgende Beobachtung gemacht:

$AI(f)$	1	2	3
Anzahl an ausgewogenen Funktionen	62	403.315.208	197.765.120
Verhältnis von ausgewogenen Funktionen	10^{-7}	0,671	0,329

Tabelle 4: Ergebnisse des Tests auf Algebraische Immunität

Hier kann man erkennen, dass etwa 33% der Funktionen eine optimale Algebraische Immunität haben. Es wurden in einigen Beispielen die Filterfunktionen der Stromchiffren untersucht und dabei wurde meist eine geringe Algebraische Immunität in diesen Funktionen vorgefunden. Wenn man jedoch zufällig ausgewählte ausgewogene Funktionen betrachtet, wurde festgestellt, dass die Algebraische Immunität unter $0,22n$ gegen 0 konvergiert, wenn $n \to \infty$ betrachtet wird [CAB+06]. Die maximale Algebraische Immunität erreichen 28,9% der ausgewogenen Funktionen mit n ungerade. Die restlichen 71,1% haben eine Algebraische Immunität von $\frac{n-1}{2}$, also fast der optimale Wert.

Für ein gutes Design einer Filterfunktion gehört nicht nur die Sicherheitsaspekt dazu, sondern sie sollte leicht in der Hardware implementierbar sein. Denn wegen ihrer schnellen Implementierung werden oft Stromchiffren mit linear rückgekoppelten Schieberegister gewählt. Daher müssen ausgewogene Funktionen mit optimaler Algebraischer Immunität gefunden werden, die aber mit möglichst wenigen Gattern implementiert werden können. Schon mit $n = 15$ Variablen muss die Repräsentierung einer Funktion geschickt in einer kompakten Form gewählt werden.

Bekannt für schnelle Implementierungen sind die symmetrischen Funktionen. Denn diese booleschen Funktionen haben bereits eine bekannte Implementierung, deren Anzahl an Gattern linear zu der Anzahl an Eingabevariablen sind. In [DMS06] werden mögliche Konstruktionspläne für eine boolesche Funktion mit maximaler Algebraischen Immunität vorgestellt. Dabei erstellen sie symmetrische Funktionen und stellen die These auf, dass die in dem Paper vorgestellte symmetrische Funktion die einzige mit maximaler Algebraischen Immunität ist. Doch es wird darauf hingewiesen, dass allgemein symmetrische Funktionen als unsicher in der Kryptographie gelten und nicht in realem System verwendet werden sollen.

Das Konstruieren von Funktionen mit maximaler Algebraischer Immunität ist allgemein ein kostspieliges Angehen. Allein die Algebraische Immunität einer gegeben Funktion anzugeben ist bei $n > 25$ nicht mehr praktisch möglich. In [NGK06] ist ein Verfahren angegeben, womit eine obere Grenze der Algebraischen Immunität einer Funktion angegeben werden kann. Die allgemeine obere Grenze von $\lceil \frac{n}{2} \rceil$ beziehungsweise $\lceil \frac{n+1}{2} \rceil$ bei ungeradem n ist des öfteren zu hoch angesetzt. Mit dem Verfahren von [NGK06] können obere Grenzen für Funktionen mit mehr als 25 Eingabevariablen angegeben werden. Aber weiterhin können keine exakte Angaben zur Algebraischen Immunität bei hoher Anzahl an Variablen gemacht werden.

Es bleiben noch einige offene Fragen und Punkte, die weiter analysiert werden müssen. Zum einen die genannte Problematik bei hoher Anzahl an Eingabevariablen. Es ist praktisch nicht möglich die exakte Algebraische Immunität einer Funktion anzugeben, wenn diese mehr als 25 Variablen als Eingabe besitzen. Selbst das Konstruieren von Funktionen mit maximaler Algebraischer Immunität ist nicht ausreichend bewerkstelligt worden. Es wurden einige Verfahren vorgestellt, aber diese sind nicht ohne weiteres im realem Kryptosystem verwendbar.

Zum anderem haben wir noch keine Gegenmaßnahmen für algebraische Angriffe auf

Combiner mit Speicher genannt. Im Kapitel 4.2 wurden ad-hoc Gleichungen gefunden, so dass die Bits aus dem Speicher nicht berücksichtigt werden müssen. Es werden r hintereinander folgende Schlüsselstrombits verwendet, so dass ein Gleichungssystem nach folgendem Schema gewonnen wird:

$$0 = \breve{F}(x^{(t)}, \ldots, x^{(t+r)}, z_t, \ldots, z_{t+r}) \quad \forall t > 0$$

Diese Funktion $\breve{F}$ kann durchaus kleinen Grad besitzen. Eine obere Grenze für die Funktion ist mit $\lceil \frac{n(l+1)}{2} \rceil$ gegeben, wobei l die Anzahl der verwendeten Bits aus einem zusätzlichen Speicher angibt. Offentsichtlich werden jedoch Funktionen kleineren Grades gefunden, wie beispielsweise beim Angriff auf die Stromchiffre E_0.

Es liegt bisher kein generelles Vorgehen für eine maximale Resistenz gegen dieses Vorgehen. Auch hier müssen noch weitere Forschungen betrieben werden und die Zusammenhänge zwischen dem Kryptosystem und daraus resultierenden ad-hoc Gleichungen gefunden werden.

5 Zusammenfassung

Um Angriffe auf Stromchiffren mit linear rückgekoppelten Schieberegistern zu beschreiben, wurden zu allererst die Designs solcher Stromchiffren vorgestellt. LFSR werden wegen ihrer schnellen und einfachen Implementierung in Hardware und aufgrund ihrer statistischen Eigenschaften und leichten Analysierbarkeit verwendet. Die Initialisierungswerte des internen Zustand eines LFSR ist der geheime Schlüssel. Da die Ausgaben der LFSR linear abhängig vom internen Zustand sind, sind sie anfällig auf Angriffe jeglicher Art.

Um Nichtlinearität in den Ausgaben der Stromchiffre zu erhalten, existieren verschiedene Ansätze. Zum einen kann der interne Zustand des LFSR als Eingabe für eine nichtlineare Filterfunktion dienen. Somit sind Unregelmäßigkeiten in der Ausgabe erlangbar. Ein weiterer Ansatz erfolgt mit einem Combiner. Hier dienen die Ausgaben mehrerer LFSR als Eingabe für eine nichtlineare Filterfunktion. Es ist auch möglich zusätzlich einen Speicher zu nutzen und diesen als weiteren Eingang in die Filterfunktion aufzunehmen, um noch weitere Abhängigkeiten in der Ausgabe zu erhalten. Der letzte Ansatz verwendet unregelmäßig getaktete LFSR. Hier folgt nicht wie bisher zu jedem Takt die Ausgabe, sondern eine Taktkontroll-Komponente bestehend aus einer Anzahl an LFSR gibt den Takt für die Datenerzeugungskomponente vor. Dadurch werden auch Unregelmäßigkeiten in den Ausgaben der Stromchiffre erlangt. Alle 3 vorgestellten Typen versuchen sich vor Angriffen durch nichtlineare boolesche Filterfunktionen zu schützen. Unterschiede sind diese, dass der Ansatz mit Combiner noch zusätzlichen Speicher nutzt und die unregelmäßig getakteten Stromchiffren durch eine Taktkontroll-Komponente die LFSR, die der Ausgabe dienen, steuern kann.

In einem algebraischen Angriff auf Stromchiffren solcher Art versucht man ein Gleichungssystem mit Wissen der verwendeten Filterfunktion und bekannten Schlüsselstrombits zu erstellen, womit der geheime Schlüssel $\mathcal{K}$ durch Lösen des Gleichungssystems gefunden werden kann. Um dieses nichtlineare Gleichungssystem zu lösen, wurden 3 Verfahren vorgestellt.

Zum einen existiert die Linearisierung. Hier werden alle Monome, die ein Produkt aus mehreren Variablen sind, durch eine neue Variable ersetzt. Das dadurch erhaltene lineare Gleichungssystem kann nach den bekannten Verfahren der Linearen Algebra gelöst werden.

Zum anderen kann man , wenn diese Anzahl an Gleichungen nicht vorhanden sein, auf die Verfahren wie XL Algorithmus oder Algorithmen, die mit Hilfe von Gröbner Basen arbeiten, zurückgreifen. Der XL Algorithmus versucht durch Multiplikationen an den gegebenen Gleichungen neue linear unabhängige Gleichungen zu gewinnen. Die Algorithmen, welche mit Hilfe von Gröbner Basen arbeiten, funktionieren durch Reduktionen und geschickte Kombinationen der Gleichungen. Sie gewinnen eine Idealbasis mit welcher ein einfacheres Gleichungssystem gewonnen wird.

Der Ansatz durch unregelmäßigen Takt Sicherheit gegen jegliche Angriffe zu erlangen, ist nicht gelungen. Wir haben gezeigt, dass die Taktkontroll-Komponente einfach ignoriert werden kann. Einerseits könnte man den Initialisierungzustand dieser Komponente raten, doch dieser Ansatz lässt die Komplexität des algebraischen Angriffes stark erhöhen. Ein besserer Ansatz besteht darin, an bestimmten Positionen, und zwar immer nach einer Periode der Taktkontroll-Komponente, die Schlüsselstrombits zu verwenden. Somit kann ein algebraischer Angriff auf die Datenerzeugungskomponente erfolgen, als wäre diese synchron getaktet.

Die Stromchiffren, welche einen Combiner mit zusätzlichen Bits aus einem Speicher nutzen, sind problematischer. Die Anzahl an Variablen im Gleichungssystem steigt schneller als die Anzahl an gegebenen Gleichungen, aufgrund der neu hinzukommenden unbekannten Bits aus dem Speicher. Wir haben gezeigt, dass auch hier ein algebraischer Angriff erfolgen kann, indem ad-hoc Gleichungen gefunden werden, die nur die Ausgaben der LFSR und die Schlüsselstrombits als Unbekannte besitzen.

Um sich gegen algebraische Angriffe zu schützen, sollten die verwendeten Filterfunktionen der Stromchiffren gewisse Eigenschaften vorzeigen. Zum einen sollte die Filterfunktion stark nichtlinear sein und ausgewogen. Zum anderen sollte sie auch einen hohen Grad besitzen. Doch diese Eigenschaften sind bereits von anderen existierenden Angriffen hergeleitet und schützen nicht ausreichend gegen algebraische Angriffe. Daher wurde ein neuer Wert definiert, der sich Algebraische Immunität nennt. Je höher die Algebraische Immunität ist, desto höhere Komplexitäten werden für einen algebraischen Angriff erwartet.

Das Ausrechnen der Algebraischen Immunität oder sogar das Konstruieren von Funktionen mit maximaler Algebraischer Immunität benötigt noch weitere Forschungen. Momentan können nur zu Funktionen mit $n < 25$ Eingaben die Algebraische Immunität in praktisch ausführbarer Laufzeit angegeben werden. Die bisherigen Konstruktionsvorschläge für Funktionen sind nicht so weit, dass sie auch in realem Kryptosystem angewendet werden könnten.

6 Literatur

Literatur

[AI04] Frederik Armknecht and Theoretische Informatik. On the existence of low-degree equations for algebraic attacks frederik armknecht, August 05 2004.

[AK03] Frederik Armknecht and Matthias Krause. Algebraic attacks on combiners with memory. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 162–175. Springer, 2003.

[Arm] Frederik Armknecht. Stream ciphers cryptanalysis - algebraic attacks. Vortrag in der Universität Mannheim.

[Arm04a] Frederik Armknecht. Algebraic attacks on stream ciphers. In *ECCOMAS*, 2004.

[Arm04b] Frederik Armknecht. Improving fast algebraic attacks. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 65–82. Springer, 2004.

[B.B70] B.Buchberger. Ein algorithmisches Kriterium für die Lösbarkeit eines algebraischen Gleichungssystems. *Aequ. Math.*, 4:374–383, 1970.

[Buc65] Bruno Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenrings nach einem nulldimensionalen Polynomideal.* PhD thesis, 1965.

[CAB+06] A. Canteaut, D. Augot, A. Biryukov, A. Braeken, C. Cid, H. Dobbertin, H.Englund, H. Gilbert, L. Granboulan, H. Handschuh, M. Hell, T. Johansson, A. Maximov, M. Parker, T. Pornin, B. Preneel, M. Robshaw, and M. Ward. D.STVL.4 – Ongoing Research Areas in Symmetric Cryptography. ECRYPT Report, Jan. 2006.

[Can03] Anne Canteaut. A5/1. *encyclopedia*, 2003.

[Can06] A. Canteaut. Open problems related to algebraic attacks on stream ciphers. *Lecture Notes in Computer Science, Springer*, 3969:1–10, 2006. Workshop on Coding and Cryptography - WCC 2005.

[CKPS00] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *EUROCRYPT*, pages 392–407, 2000.

[CL05] Cid and Leurent. An analysis of the XSL algorithm. In *ASIACRYPT: Advances in Cryptology – ASIACRYPT: International Conference on the Theory and Application of Cryptology.* LNCS, Springer-Verlag, 2005.

[CM03] Nicolas Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2003.

[Cou02] Nicolas Courtois. Higher order correlation attacks, XL algorithm and cryptanalysis of toyocrypt. In Pil Joong Lee and Chae Hoon Lim, editors, *ICISC*, volume 2587 of *Lecture Notes in Computer Science*, pages 182–199. Springer, 2002.

[Cou03] Nicolas T. Courtois. Fast algebraic attacks on stream ciphers with linear feedback. *Lecture Notes in Computer Science*, 2729:176–194, 2003.

[CP03] Nicolas T. Courtois and Jacques Patarin. About the XL algorithm over $GF(2)$. *Lecture Notes in Computer Science*, 2612:141–157, 2003.

[DMS06] Deepak Kumar Dalai, Subhamoy Maitra, and Sumanta Sarkar. Basic theory in construction of boolean functions with maximum possible annihilator immunity. *Des. Codes Cryptography*, 40(1):41–58, 2006.

[EJ05] Håkan Englund and Thomas Johansson. A new distinguisher for clock controlled stream ciphers. In Henri Gilbert and Helena Handschuh, editors, *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 181–195. Springer, 2005.

[FA03] Jean-Charles Faugère and Gwénolé Ars. An algebraic cryptanalysis of nonlinear filter generators using gröbner bases, 2003.

[FA04] Jean-Charles Faugère and Gwénolé Ars. Comparison of XL and gröbner basis algorithms over finite fields, 2004.

[Fau02] Jean Charles Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In Teo Mora, editor, *ISSAC 2002: Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, July 07–10, 2002, Université de Lille, Lille, France*, pages 75–83, pub-ACM:adr, 2002. ACM Press.

[FL01] Scott R. Fluhrer and Stefan Lucks. Analysis of the e_0 encryption system. In Serge Vaudenay and Amr M. Youssef, editors, *Selected Areas in Cryptography*, volume 2259 of *Lecture Notes in Computer Science*, pages 38–48. Springer, 2001.

[Gmb02] Dirk Fox - Secorvo Security Consulting GmbH. Bluetooth Security - Sicherheitsmechanismen des Bluetooth Standards. Secorvo White Paper, Sept. 2002. Version 1.1.

[JR95] C. Jager and D. Ratz. A combined method for enclosing all solutions of nonlinear systems of polynomial equations. *Reliable Computing*, 1(1):41–64, 1995.

[Ker02] Prof. Dr. Ina Kersten. Algebra. Vorlesungsskript in der Georg-August-Universität Göttingen, 2002.

[Kle05] Elizabeth Kleiman. The xl and xsl attacks on baby rijndael, 2005.

[Kra02] Matthias Krause. BDD-based cryptanalysis of keystream generators. *Lecture Notes in Computer Science*, 2332:222–??, 2002.

[Lan06] Barbara Lucie Langer. Stromchiffren - Entwurf, Einsatz und Schwächen, Juli 2006.

[Luc05] Stefan Lucks. Kryptographie 1. Vorlesung in der Universität Mannheim, SS05. Kapitel 4.

[NGK06] Y. Nawaz, G.Gong, and K.Gupta. Upper bounds on algebraic immunity of boolean power functions. to be published in Lecture Notes in Computer Science, FSE 2006. Springer-Verlag.

[Nyb05] Prof. Kaisa Nyberg. Cryptography and data security. Vorlesung in Helsinky University of Technology, Spring 05. Lecture 4.

[Pas] Enes Pasalic. Stream ciphers - design and security. http://www2.mat.dtu.dk/people/E.Pasalic/LecturesPdf/AlgAttacksF.pdf.

[Rob95] M. J. B. Robshaw. Stream ciphers. Technical report, July 25 1995.

[Sar02] Palash Sarkar. The filter-combiner model for memoryless synchronous stream ciphers. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 533–548. Springer, 2002.

[SC0X] Bimal Roy Sandeepan Chowdhury, Subhamoy Maitra, editor. *Practical Design of Nonlinear Combiner Model as a Software Stream Cipher*, Indian Statistical Institute, 200X.

[Wik06] Wikipedia. Stream ciphers. http://www.wikipedia.org, Juli 2006.

[YCC04] Yang, Chen, and Courtois. On asymptotic security estimates in XL and grobner bases-related algebraic cryptanalysis. In *ICIS: International Conference on Information and Communications Security (ICIS), LNCS*, 2004.