Universität Ulm
Fakultät für Ingenieurwissenschaften und Informatik
Institut für Theoretische Informatik
James-Franck-Ring 27
89069 Ulm

Sommersemester 2008

Bachelorarbeit

Boolsche Gleichungssysteme, SAT Solver und Stromchiffren

vorgelegt von
Enrico Pilz
am 28. August 2008

Gutachter:
Prof. Dr. Uwe Schöning
Dipl. Inf. Tobias Eibach

Auf elektronischen Weg kann diese Arbeit über den Volltextserver der Universität Ulm abgerufen werden:

Pilz, Enrico: Boolsche Gleichungssysteme, SAT Solver und Stromchiffren.
Ulm, Univ., Bachelorarbeit, 2008 [online]
URL: http://vts.uni-ulm.de/doc.asp?id=6526
URN: urn:nbn:de:bsz:289-vts-65265

Gesetzt mit LaTeX und KOMA-Script.

Zusammenfassung

In dieser Arbeit wird ein algebraischer State-Recovery-Angriff auf zwei Stromchiffren Bivium und Trivium vorgestellt. Die Spezifikation der Chiffren wird auf zwei Arten in ein dünn besetztes Gleichungssystem umgesetzt und in eine äquivalente aussagenlogische Formel in KNF umgewandelt, die ein SAT Solver lösen kann. Um die Komplexität zu reduzieren, wird eine Untermenge der Variablen geraten. Verschiedene Strategien diese Menge auszuwählen werden vorgestellt und hinsichtlich verschiedener Parameter wie beispielsweise Größe des vereinfachten Gleichungssystems, Größe der KNF oder Laufzeit des SAT Solvers verglichen. Weiterhin wird eine Variante vorgestellt, die ausnutzt, dass SAT Solver Konflikte in Form von zusätzlichen Klauseln lernen können.

Mit den vorgestellten Methoden wurde für einen Angriff auf Bivium bei minimal bekannten Schlüsselstrom mit den Ratestrategien *ending* und *equations* eine Dauer von 2^{46} Sekunden ermittelt. Dies ist besser als Brute Force auf den Schlüssel, damit gilt diese Stromchiffre als gebrochen.

Für Trivium ergibt sich mit der Ratestrategie *equations* eine Komplexität von 2^{150}. Damit ist Trivium zwar nicht gebrochen, aber dieses Ergebnis ist eine Verbesserung.

Inhaltsverzeichnis

Danksagungen

Für die Betreuung danke ich Prof. Dr. Uwe Schöning und Tobias Eibach.

Während der Entstehung gaben einige Personen ihre unschätzbaren Kommentare zu dieser Arbeit ab. Mein Dank gilt wiederum Tobias Eibach sowie Heike Brauer, André Pilz und Andreas Helbrich. Weiterhin möchte ich den Erbauern von Sage, Python, MiniSat, RSat und LaTeX für ihre wundervollen Produkte danken. Ganz besonderer Dank gilt meinen Eltern. Ohne sie wäre ich heute nicht auf der Welt. Zuletzt möchte ich Thekla dafür danken, dass sie einfach da ist.

Ulm, den 27. August 2008 Enrico Pilz

1 Einleitung

Ein Ziel der Kryptographie ist die Verschlüsselung von Informationen. Dazu gibt es verschiedene Verfahren. Mit moderner Public-Key-Kryptographie kann die Kommunikation komplett über einen unsicheren Kanal stattfinden, allerdings können keine großen Datenmengen verschlüsselt werden, da die Berechnung nicht sehr schnell ist. Andererseits ist das One-Time-Pad absolut sicher, benötigt aber einen ebenso großen Schlüssel wie den Klartext. Für viele Anwendungen wie beispielsweise ein verschlüsseltes Telefongespräch oder Videoübertragung ist dies nicht praktikabel. Man möchte dort einen kleinen Schlüssel, schnelle Verschlüsselung und möglichst wenig Hardwareaufwand bei möglichst hoher Sicherheit.

Das One-Time-Pad wird als ein Strom echt zufälliger Bits erzeugt. Eine Abwandlung ist ein Pseudozufallszahlengenerator (PZZG), der mit einem Alice und Bob bekannten Seed initialisiert wird und daher auf beiden Seiten den gleichen Schlüsselstrom liefert, mit dem dann eine beliebig große Datenmenge verschlüsselt werden kann. Stromchiffren stellen einen solchen PZZG dar. Diese sind wieder interessant geworden, weil sie im Vergleich zu Blockchiffren sehr effizient in Hardware implementiert werden können. Trivium und die Vereinfachung Bivium sind zwei solche Stromchiffren.

Für einen kryptoanalytischen Angriff kann man davon ausgehen, dass der Schlüsselstrom z durch einen *known-plaintext-Angriff* bekannt ist. Außerdem dürfen ein paar Variablen des inneren Zustandes geraten werden, um die Komplexität einzuschränken und für Experimente zugänglich zu machen. Dies nennt man einen *guess-and-determine-Angriff*. Eine Stromchiffre stellt eine mathematische Abbildung dar und kann als ein Gleichungssystem über Variablen des inneren Zustandes beschrieben werden. Dieses Gleichungssystem kann verschieden erzeugt werden. Nach passenden Vereinfachungen wird es in eine aussagenlogische Formel überführt, die dann als Erfüllbarkeitsproblem von einem SAT Solver gelöst werden kann. Die Ausgabe eines vollständigen SAT Solvers kann „nicht erfüllbar“ oder, falls die Variablen richtig geraten wurden, „erfüllbar“ mit dazugehörigen Modell sein. Aus diesem Modell kann man dann den inneren Zustand ablesen und daraus den ursprünglichen Schlüssel berechnen.

1.1 Aufbau

Die Arbeit gliedert sich wie folgt: In Kapitel 2 werden die theoretischen Hintergründe zu kryptographischen Angriffen, Gleichungssystemen über GF(2), SAT Solvern und Stromchiffren erklärt, außerdem wird bewiesen, dass das MQ-Problem NP-vollständig ist.

In Kapitel 3 geht es um die verwendeten Methoden. Nach dem Gerät und den Hilfsmitteln werden die zwei Arten der Gleichungssysteme vorgestellt. Es folgen mögliche

Vereinfachungen und wie daraus eine KNF erzeugt wird. Einen großen Raum nimmt der Abschnitt über Variablen raten ein. Das Kapitel wird durch einen Abschnitt über Klauseln lernen abgeschlossen.

Im Kapitel 4 werden die experimentellen Ergebnisse aus Sicht der Gleichungssysteme und der Ratestrategien vorgestellt und kommentiert. Abgerundet wird die Arbeit durch den Vergleich mit einer Alternative, den Gröbnerbasen.

1.2 Zielsetzung

Im Vergleich zu den vorherigen Arbeiten [14, 13] soll hier der theoretische Hintergrund inklusive der Beweise vertieft werden. Während SAT Solver bisher meist als „black box“ betrachtet wurden, wird ein Einblick in deren Arbeitsweise gegeben. Daraus ergibt sich auch eine neue Angriffsvariante. Die verwendeten Ratestrategien wurden bisher nur nach Laufzeit der SAT Solver untersucht. Hier wird zusätzlich der Aufbau der Gleichungssysteme und der boolschen Formeln untersucht. Es wird parallel betrachtet, inwiefern sich die Erkenntnisse über Bivium auf Trivium anwenden lassen. Zusätzlich wurde eine Neuimplementierung in Sage, bisher C bzw. Java, vorgenommen. Der Leser soll so in die Lage versetzt werden, mit Hilfe des methodischen Teils diesen Angriff selbst zu implementieren oder die Implementierung des Autors zu erweitern [21].

1.3 Vorherige Ergebnisse

Raddum [25] hat Bivium eingeführt und bricht es mit einem Graphenalgorithmus in 2^{56} Sekunden. Mit diesem Ansatz würde ein Angriff auf Trivium 2^{164} Sekunden benötigen.

Maximov und Biryukov [18] raten bestimmte Variablen, wodurch die quadratischen Terme verschwinden und ein lineares Gleichungssystem übrig bleibt, das mit einem Aufwand von c gelöst werden kann. Sie geben für Bivium eine Komplexität von $c \cdot 2^{36.1}$ mit $c \approx 2^{14}$ an. Allerdings werden für diesen Angriff $2^{11.7}$ Bits Schlüsselstrom benötigt. Für Trivium liegt die Komplexität bei $c{\cdot}2^{83.5}$ mit $c \approx 2^{16.2}$, es werden $2^{61.5}$ Bits Schlüsselstrom benötigt.

McDonald, Charnes und Pieprzyk [20] arbeiten ebenfalls mit dem SAT Solver MiniSat und den kurzen Gleichungen aus Abschnitt 3.3, sie raten 34 Variablen. Sie berechnen eine voraussichtliche Dauer von 2^{52} Sekunden mit einem Schlüsselstrom von 1770 Bit. Sie verbessern dieses Ergebnis in [19] auf $2^{42.7}$ Sekunden. Für Trivium wird eine Komplexität von $2^{159.9}$ Sekunden angegeben.

Eibach, Pilz, Steck und Völkel [14, 13] verwenden verschiedene Ansätze. Mit einem Schlüsselstrom von 200 Bit, dem SAT Solver und 45 geratenen Variablen kommen sie auf eine Komplexität von $2^{43.9}$ Sekunden, mit Gröbnerbasen und 60 geratenen Variablen auf $2^{59.8}$ Sekunden, mit BDDs auf $2^{75.2}$ Sekunden. Ihre erschöpfende Schlüsselsuche liegt bei $2^{57.1}$ Sekunden. Der Angriff mittels Gröbnerbasen wird auf $2^{37.7}$ Sekunden bei 42 geratenen Variablen verbessert [15].

Gaj, Southern und Bachimanchi [16] geben für Trivium-64 ein Maximaldurchsatz von

$51.2 \approx 2^{35.6}$ GBit/s an. Trivium-64, weil pro Takt 64 Bits rotiert werden. Daraus kann man einen etwas besseren Wert als in [13] für die erschöpfende Suche berechnen, da die Berechnung komplett in Hardware ausgeführt werden kann. Um die ersten 288 Ausgabebits zu ermitteln muss man $1152 + 288 \approx 2^{10.5}$ Bits berechnen. Um 2^{79} Schlüssel durchzuprobieren benötigt man daher etwa $2^{79-(35.6-10.5)} = 2^{53.9}$ Sekunden. Für Bivium liegt dieser Wert etwas niedriger, da die Initialisierungsphase kürzer ist. Man muss nur $708 + 177 \approx 2^{9.8}$ Bits berechnen und kommt damit auf $2^{53.2}$ Sekunden.

2 Hintergrund

Stromchiffren haben zwei Anwendungsgebiete: Wenn die Verschlüsselung außergewöhnlich schnell sein muss, oder wenn außergewöhnlich beschränkte Ressourcen zur Verfügung stehen. Dies war 2004 der Auslöser für das eSTREAM Projekt, mit dem geeignete Stromchiffren gesucht wurden [1]. Die Kandidaten sollten in eines der Anwendungsgebiete passen und sicher, performant, eine Analyse zugänglich, einfach und flexibel sein.

Zum Verschlüsseln wird bei einer Stromchiffre aus den geheimen *Schlüssel* K und den im Allgemeinen bekannten *Initialisierungsvektor* IV in der Art eines PZZG der *Schlüsselstrom*, ein fast beliebig langer Bitstring, erzeugt. Dieser wird als Quasi-One-Time-Pad mit dem Klartext, der als Bitstring vorliegt, per $\oplus$-Operation verknüpft und man erhält den verschlüsselten Text. Zum Entschlüsseln wird wieder der Schlüsselstrom erzeugt und mit dem verschlüsselten Text per $\oplus$-Operation verknüpft, man erhält den Klartext. Der IV wird verwendet, um die Verschlüsselung erneut zu initialisieren, ohne eine neuen geheimen Schlüsselaustausch vornehmen zu müssen.

2.1 Angriffsarten

Zur Betrachtung der Sicherheit einer Stromchiffre gibt es diverse Ansätze. Nach Kerckhoffs Prinzip geht man davon aus, dass die Algorithmen der Chiffre bekannt sind und nur der Schlüssel bzw. bei Stromchiffren der innere Zustand geheim ist. Ist ein Verschlüsselungsalgorithmus lange genug bekannt und untersucht und wurde kein erfolgreicher Angriff gefunden, gilt er als hinreichend sicher.

Bei einem *Brute Force* Angriff wird der gesamte Schlüsselraum systematisch durchsucht. Dazu wird ein Schlüssel gewählt, der entsprechende innere Zustand und dann der Schlüsselstrom werden erzeugt und mit dem gesuchten Schlüsselstrom verglichen bis eine Übereinstimmung gefunden wird. Mit Brute Force müssen bei einem 80-Bit Schlüssel im Mittel 2^{79} Schlüssel getestet werden. Man kann diesen Wert durch Parallelisierung mehrerer Berechnungen verbessern, wenn man einen großen Ausschnitt des Schlüsselstroms, hier etwa in der Größenordnung 2^{40}, zur Verfügung hat, weil das Entschlüsseln mehrerer Texte nicht schwieriger ist als einen Text zu entschlüsseln. Man kann einen Angriff als erfolgreich bewerten, wenn er um Größenordnungen besser als Brute Force ist.

Andere Angriffsarten zielen darauf ab, Bits des Schlüsselstroms mit einer Wahrscheinlichkeit größer $1/2 + \epsilon$ vorherzusagen oder den Schlüsselstrom von einer Zufallszahlenfolge unterscheiden zu können. Wiederum andere Verfahren zielen darauf ab, nicht den Schlüssel aufzudecken, sondern Kollisionen zwischen Initialisierungsvektoren bei gleichem Schlüssel herauszufinden, man nennt dies *correlation-attack*.

Ein *algebraischer Angriff* stellt ein die Chiffre beschreibendes Gleichungssystem auf,

welches den unbekannten inneren Zustand als Variablen enthält. Eine Lösung dieses Gleichungssystems wäre der innere Zustand. Dies nennt man eine *State-Recovery-Attack*. Der Vorteil ist, dass im Gegensatz zu anderen Verfahren nur sehr wenig Schlüsselstrom benötigt wird. In dieser Arbeit wird ein algebraischer Angriff auf den inneren Zustand von Bivium und Trivium vorgestellt.

Die hier betrachteten Chiffren haben einen inneren Zustand von 177 bzw. 288 Bits, d.h. ein Angriff auf den inneren Zustand mit Brute Force ist wesentlich ungünstiger als ein Brute Force Angriff direkt auf den Schlüssel.

2.2 Gleichungssysteme über GF(2)

Ein endlicher Körper mit q Elementen wird nach Évariste Galois als *Galois field* bzw. GF(q) bezeichnet. In dieser Arbeit werden nur Variablen über GF(2) betrachtet.

Ein *Monom* ist ein Produkt von Variablen, etwa

$$s_{175} \cdot s_{176},$$

wobei der *Grad* die Anzahl der unterschiedlichen Variablen ist.

Ein *Polynom* ist eine Summe von Monomen, etwa

$$s_{69} + s_{162} + s_{175} \cdot s_{176} + s_{177}.$$

Der Grad des Polynoms ist der maximale Grad der beteiligten Monome.

Polynome über GF(2) haben Eigenschaften, die man für eine schnellere Implementierung ausnutzen kann. Für die Multiplikation gilt $0 \cdot 0 = 0$ und $1 \cdot 1 = 1$, daher gilt $x^1 = x^2 = x^3 = \ldots$ für alle positiven Potenzen von x. Wenn es n Variablen gibt, können die Variablen eines Monoms als Bitstring der Länge n, ein Polynom als eine Menge solcher Bitstrings gespeichert werden. Die Anzahl der Monome vom Grad k berechnet sich als $\binom{n}{k}$. Konstante Faktoren vor einem Monom können nur den Wert 1, d.h. das Monom ist da oder den Wert 0, d.h. das Monom ist nicht da, annehmen. Zwei gleiche Monome löschen sich bei Addition und Subtraktion gegenseitig aus. Daher erhält man ein homogenes Gleichungssystem, d.h. auf einer Seite stehen nur Nullen. In Implementierungen muss dann nur das Polynom gespeichert werden. Wenn man eine Variable x in einem Monom setzt, wird die Variable gestrichen, falls $x = 1$, oder das Monom verschwindet, falls $x = 0$. Diese Vereinfachungen wurden durch das verwendete Algebrapaket PolyBoRi durchgeführt.

Das *MQ-Problem* (oder auch $\mathcal{MQ}$-Problem) ist ein Gleichungssystem

$$\begin{aligned} z_1 &= f_1(s_1, \ldots, s_n) \\ &\vdots \\ z_m &= f_m(s_1, \ldots, s_n) \end{aligned}$$

mit m Gleichungen und n Variablen, wobei die f_i Polynome höchstens vom Grad 2 darstellen. Es ist bei gegebenen z_i eine Belegung für die Variablen s_i gesucht, so dass alle

Gleichungen erfüllt sind. Ein Gleichungssystem gilt als *dünn besetzt* wenn der Anteil β der tatsächlich vorkommenden Monome kleiner als 1/100 ist [3].

Die Anzahl der möglichen Monome und damit die maximale Länge der Polynome steigt exponentiell mit wachsenden Grad. Durch Substitution läßt sich zeigen, dass sich ein Polynom vom Grad größer 2 auf mehrere Polynome vom Grad 2 abbilden läßt. Dies geht durch Einführung neuer Variablen. So kann etwa

$$z_1 = s_1 s_2 s_3 s_4$$

durch die drei Gleichungen

$$z_1 = v_6 s_4, \quad v_5 = s_1 s_2, \quad v_6 = v_5 s_3$$

dargestellt werden. Durch dieses Verfahren ist es gerechtfertigt, dass auch Gleichungssysteme mit Polynomen vom Grad größer zwei als Instanz des MQ-Problems behandelt werden können.

2.3 SAT Solver

In vorherigen Arbeiten wurden SAT Solver als „black box“ behandelt [20, 19, 12, 14, 13]. Hier wird ein etwas tieferer Einblick gegeben, der zu einer neuen Angriffsvariante führt. Die folgende allgemeine Darstellung basiert auf [17].

Eine *Variable* x ist ein Wahrheitswert mit dem Wertebereich {TRUE, FALSE}, den man auch mit $\{0, 1\}$ identifiziert. Ein *Literal* ist eine wahre oder negierte Variable, notiert mit x bzw. $\neg x$. Eine *Klausel* ist eine Disjunktion von Literalen, etwa $x \vee \neg y$. Die *Klauselnormalform* (KNF) ist eine Konjunktion von Klauseln, etwa $(x \vee \neg y) \wedge (\neg x \vee y)$. Jede *aussagenlogische Formel* kann in KNF dargestellt werden. Eine *Belegung* weist Variablen einen Wert aus $\{0, 1\}$ zu und kann partiell oder vollständig sein.

Mit $F' = F|_x$ wird die Formel bezeichnet, die aus F entsteht, wenn alle Vorkommen von x durch den Wahrheitswert von x ersetzt werden. Wenn hierbei in einer Klausel das zu x gehörende Literal zu TRUE evaluiert, wird die Klausel gestrichen, bei FALSE das Literal.

Beim *SAT-Problem* als Entscheidungsproblem wird gefragt, ob für eine aussagenlogische Formel in KNF eine Belegung für die Variablen existiert, so dass die Formel wahr wird. Dieses Problem liegt in $\mathcal{NP}$. Ein *k-SAT-Problem* ist ein SAT-Problem, bei dem jede Klausel aus maximal k Literalen besteht, es liegt für $k \geq 3$ ebenfalls in $\mathcal{NP}$.

In der praktischen Anwendung wird für den erfüllbaren Fall eine gültige Belegung der Variablen, das *Modell*, gesucht. Ein *SAT Solver* ist ein Programm zur Lösung des SAT-Problems. Die Eingabe ist eine KNF. Falls die Formel erfüllbar ist, wird SAT (auch *satisfiable*) und ein Modell, sonst UNSAT (*unsatisfiable*) ausgegeben.

Man unterscheidet *vollständige* und *unvollständige* SAT Solver. Die vollständigen können durch eine Suche über alle Belegungen immer ein gültiges Ergebnis garantieren. Im Gegensatz dazu arbeiten die *unvollständigen* mit probabilistischen Local-Search-Techniken und können nicht garantieren, dass eine Lösung gefunden wird. Weiterhin sind diese Verfahren zu langsam, wenn es keine Lösung gibt.

Es werden nur die vollständigen SAT Solver betrachtet. Diese basieren auf dem Davis-Putnam-Logemann-Loveland-Verfahren (DPLL), das rekursiv beschrieben wird, modifiziert nach [17]:

def DPLL(F, φ):
    (F, φ) = simplify(F, φ)
    if $\square$ **in** F: – Widerspruch
        return UNSAT
    if $F = \emptyset$: – alle Klauseln erfüllt
        output φ
        return SAT
    choose $\ell \notin \varphi$
    if DPLL($F|_\ell$, $\varphi \cup \{\ell\}$) = SAT:
        return SAT
    return DPLL($F|_{\neg\ell}$, $\varphi \cup \{\neg\ell\}$)

Das Verfahren wird mit einer aussagenlogischen Formel F in KNF und einer partiellen Belegung φ aufgerufen. Initial ist φ die leere Menge. Als erstes werden Vereinfachungen durchgeführt. Wenn die leere Klausel $\square$ in der Formel F auftritt, wird „nicht erfüllbar“ zurückgegeben. Wenn die Formelmenge leer ist, gibt es für alle Variablen eine gültige Belegung und es wird „erfüllbar“ und die Belegung zurückgegeben. Sonst wird eine noch nicht in der Belegung φ vorkommende Variable ℓ ausgewählt und das Verfahren wird rekursiv mit beiden Belegungen der Variable aufgerufen.

def simplify(F, φ):
    if $\square$ **in** F: – Widerspruch
        return (F, φ)
    if $\{x\}$ **in** F: – Einerklausel
        return simplify($F|_x$, $\varphi \cup \{x\}$)
    if p **in** F **and** p **is** pure: – pures Literal
        return simplify($F|_p$, $\varphi \cup \{p\}$)
    return (F, φ)

Ein *pures Literal* ist eine Variable, die in allen Klauseln mit dem gleichen Wahrheitswert vorkommt. Eine *Einheit* ist eine Klausel, die nur noch aus einem Literal besteht. Die Vereinfachungen bestehen aus rekursiven Entfernung der Einheiten und der puren Literale. Dadurch werden Äste des Suchbaumes mit nur unerfüllbaren Belegungen abgeschnitten, da Konflikte meist sehr früh erkannt werden.

Eine Erweiterung dieses Verfahrens enthält das Lernen von Klauseln (*clause learning*). Dabei werden Konflikte, die während der Suche auftreten als neue Klauseln in die Formel eingefügt. Es ergibt sich ein Vorteil, wenn DPLL mehrmals aufgerufen wird.

2.4 MQ-Problem ist NP-vollständig

Es ist bekannt, dass MQ in der Klasse der NP-vollständigen Probleme liegt. Der folgende Beweis zu Lemma 1 orientiert sich an [27, S. 28–30] und [2, S. 60–61], Lemma 2 stellt

eine Verallgemeinerung der dort vorgestellten Beweise dar.

Lemma 1 *Das MQ-Problem kann mit einem nichtdeterministischen Algorithmus in polynomieller Zeit gelöst werden.*

BEWEIS Seien n Variablen und m Gleichungen $f_1, \ldots, f_m$ mit dazugehörigen Konstanten $z_1, \ldots, z_m$ gegeben. Wähle zufällig eine Belegung der Variablen $g_1, \ldots, g_m$ und breche ab, wenn die Belegung alle Gleichungen erfüllt, sonst wiederhole den Algorithmus.

while True:
  $g_1, \ldots, g_n \in_{\mathrm{R}} \{0,1\}^n$
  if $\forall i, 1 \leq i \leq m : f_i(g_1, \ldots, g_n) = z_i$:
    return True

Der Test auf Erfüllbarkeit aller Gleichungen ist in polynomieller Zeit möglich. Bei Polynomen vom maximalen, konstanten Grad k gibt es maximal

$$\sum_{i=0}^{k} \binom{n}{i} = 1 + n + \binom{n}{2} + \ldots + \binom{n}{k} \leq k \cdot \binom{n}{k} = k \cdot \frac{n!}{k! \cdot (n-k)!} \leq \frac{n^k}{(k-1)!} \leq n^k$$

Monome pro Gleichung, wobei $\binom{n}{i}$ die Anzahl der möglichen Monome vom Grad i ist. Dies gilt insbesondere für den Grad 2, es gilt $1 + n + \binom{n}{2} \in O(n^2)$. Der Algorithmus terminiert genau dann, wenn es (mindestens) eine erfüllende Belegung gibt. Damit gilt die Behauptung. ∎

Lemma 2 *Das k-SAT-Problem kann mit polynomiellen Aufwand auf das MQ-Problem reduziert werden.*

BEWEIS Bestehe o.B.d.A jede Klausel im k-SAT-Problem aus k Literalen, weil jede kleinere Klausel durch Einführung von Konstanten entsprechend ergänzt werden kann. Man betrachte eine Klausel $(x_1 \vee x_2 \vee \ldots \vee x_k)$, wobei die Literale x_i wahr oder negiert vorkommen dürfen. Es gilt

$$(x'_1 \vee x'_2 \vee \ldots \vee x'_k) \Leftrightarrow x_1 x_2 \cdots x_k = 0,$$

wobei $\forall\, i : x'_i = \neg x_i$. Dies ist korrekt, denn die Klausel ist genau dann falsch, wenn alle x'_i falsch bzw. alle x_i wahr sind. Das Polynom ist genau dann falsch, wenn $\forall\, i : x_i = 1$ gilt und damit das Produkt 1 ist.

Der Grad des Polynoms ist k und es wird genau ein Monom erzeugt. Weiterhin kann mit dem beschriebenen Verfahren aus Abschnitt 2.2 durch neue Variablen und Substitution mit linearen Aufwand ein quadratisches Gleichungssystem erzeugt werden. Aus einem k-SAT-Problem mit m Klauseln erhält man also m Polynome von Grad k bzw. maximal $(k-1) \cdot m$ Polynome vom Grad zwei. Eine Lösung für das Gleichungssystem ist ebenfalls eine Lösung für das k-SAT-Problem. Damit gilt die Behauptung. ∎

Bemerkungen

- Für das 3-SAT-Problem kann eine Reduktion mit der Äquivalenz

$$(x \vee y \vee z) \Leftrightarrow (xyz + xy + xz + yz + x + y + z) = 1,$$

  vorgenommen werden. Allerdings ist für k-SAT die Längenzunahme dieses Polynoms exponentiell in k.

- Durch den Algorithmus in Abschnitt 3.5 wird eine polynomielle Reduktion des MQ-Problems auf das SAT-Problem angegeben.

Satz 1 *Das MQ-Problem ist NP-vollständig.*

Beweis Die Behauptung folgt aus den Lemmata 1 und 2. ∎

Wenn man P $\neq$ NP annimmt, ist eine Konsequenz des Satzes 1, dass es im Allgemeinen keine effizienten Algorithmen zur Lösung quadratischer Gleichungssysteme geben kann.

2.5 Stromchiffren

Im Folgenden werden die Stromchiffren Trivium und Bivium vorgestellt. Trivium besitzt drei Schieberegister. Bivium ist eine reduzierte Variante von Trivium und besitzt nur zwei Register.

2.5.1 Trivium

Diese Stromchiffre ist ein Finalist des eStream-Projektes in der Kategorie Hardware und basiert auf Designkriterien, die sich aus der Analyse von Blockchiffren ableiten [1, 5, 6, 7]. Einerseits wurde auf die Sicherheit besonderer Wert gelegt, andererseits ist diese Chiffre sehr effizient in Hardware implementierbar, weil pro Zyklus und Schieberegister bis zu 66 Bit berechnet und rotiert werden können.

Trivium besteht aus drei nichtlinearen Schieberegister der Längen 93, 84 und 111. In der Spezifikation werden alle drei Register über Variablen des inneren Zustandes mit $s_1, \ldots, s_{288}$ bezeichnet. Zum besseren Verständnis werden diese hier einzeln als $a_1, \ldots, a_{93}$, $b_1, \ldots, b_{84}$ und $c_1, \ldots, c_{111}$ benannt.

Pro Takt wird ein Ausgabebit z_i erzeugt und drei neue Werte t_1, t_2, t_3 für die Schieberegister wie folgt berechnet:

for i **from** 1 **to** N **do**
$t_1 \leftarrow a_{66} + a_{93}$
$t_2 \leftarrow b_{69} + b_{84}$
$t_3 \leftarrow c_{66} + c_{111}$
$z_i \leftarrow t_1 + t_2 + t_3$

$$
\begin{aligned}
t_1 &\leftarrow t_1 + a_{91} \cdot a_{92} + b_{78} \\
t_2 &\leftarrow t_2 + b_{82} \cdot b_{83} + c_{87} \\
t_3 &\leftarrow t_3 + c_{109} \cdot c_{110} + a_{69} \\
(a_1, a_2, \ldots, a_{93}) &\leftarrow (t_3, a_1, \ldots, a_{92}) \\
(b_1, b_2, \ldots, b_{84}) &\leftarrow (t_1, b_1, \ldots, b_{83}) \\
(c_1, c_2, \ldots, c_{111}) &\leftarrow (t_2, c_1, \ldots, c_{110})
\end{aligned}
$$

Man beachte wie t_1, t_2, t_3 verteilt werden. Die maximale Anzahl erzeugter Bits z_i im Schlüsselstrom ist auf $N \leq 2^{64}$ beschränkt.

Zur Initialisierung werden ein Schlüssel K und ein Initialisierungsvektor IV mit je 80 Bit in den initialen Zustand geladen, siehe auch [9].

$$
\begin{aligned}
(a_1, a_2, \ldots, a_{93}) &\leftarrow (K_{80}, \ldots, K_1, 0, \ldots, 0) \\
(b_1, b_2, \ldots, b_{84}) &\leftarrow (IV_{80}, \ldots, IV_1, 0, \ldots, 0) \\
(c_1, c_2, \ldots, c_{111}) &\leftarrow (0, \ldots, 0, 1, 1, 1)
\end{aligned}
$$

Danach wird Trivium $4 \cdot 288 = 1152$ Takte ohne Erzeugung eines Ausgabebits weitergerechnet, um einen möglichst gleichmäßig verteilten inneren Zustand zu erreichen. Auf diese Weise werden die initial 80 Unbekannte des Schlüssels K auf 288 Variablen des inneren Zustandes ausgeweitet.

2.5.2 Bivium

Bivium ist eine Vereinfachung von Trivium [25]. Es wird auf das dritte Register verzichtet. Die Initialisierung erfolgt mit

$$
\begin{aligned}
(a_1, a_2, \ldots, a_{93}) &\leftarrow (K_{80}, \ldots, K_1, 0, \ldots, 0) \\
(b_1, b_2, \ldots, b_{84}) &\leftarrow (IV_{80}, \ldots, IV_1, 0, \ldots, 0)
\end{aligned}
$$

und dann wird $4 \cdot 177 = 708$ Takte ohne Ausgabe weitergerechnet. Analog erfolgt die Schlüsselstromgenerierung:

for i **from** 1 **to** N **do**

$$
\begin{aligned}
t_1 &\leftarrow a_{66} + a_{93} \\
t_2 &\leftarrow b_{69} + b_{84} \\
z_i &\leftarrow t_1 + t_2 \\
t_1 &\leftarrow t_1 + a_{91} \cdot a_{92} + b_{78} \\
t_2 &\leftarrow t_2 + b_{82} \cdot b_{83} + a_{69} \\
(a_1, a_2, \ldots, a_{93}) &\leftarrow (t_2, a_1, \ldots, a_{92}) \\
(b_1, b_2, \ldots, b_{84}) &\leftarrow (t_1, b_1, \ldots, b_{83})
\end{aligned}
$$

3 Methoden

Die sehr einfache und elegante Struktur von Trivium und Bivium ermöglicht eine kostengünstige Hardwareimplementierung mit wenigen Gattern. Dies hat den für die Kryptoanalyse günstigen Nebeneffekt, dass ein die Stromchiffre beschreibendes Gleichungssystem nur dünn besetzt ist. Dies macht den hier beschriebenen Angriff durchführbar: Es wird ein Paar aus Schlüssel und Schlüsselstrom als Vorgabe erzeugt. Danach wird ein Gleichungssystem über GF(2) mit dem Schlüsselstrom als Bekannte und dem Schlüssel als Unbekannte aufgestellt, vereinfacht und in eine KNF umgewandelt, die von einem SAT Solver gelöst wird. Dabei werden zur Reduzierung der Komplexität einige Werte aus dem Schlüssel vorgegeben bzw. zufällig gesetzt.

3.1 Gerät und Hilfsmittel

Dieser Arbeit liegt eine Implementierung in Sage 3.0.1 zugrunde [26]. Die verwendete Programmiersprache ist Python 2.5 [24]. Für die Gleichungssysteme über GF(2) wurde das Paket PolyBoRi verwendet [23, 4].

Als SAT Solver wurden MiniSat 2.0 beta [11] sowie RSat 2.01 [22] mit dem Präprozessor SatElite [10] verwendet. Diese Entscheidung stützt sich auf Vergleichtests mit Instanzen aus der Problemdomäne zwischen sechs SAT Solvern, die in den SAT Wettbewerben der letzten Jahre in der Spitzengruppe liegen oder lagen.

Die experimentellen Berechnungen wurden auf einem Rechner mit 2.4 GHz CPU und 512 MB Arbeitsspeicher ausgeführt.

3.2 Erzeugen langer Gleichungen

Zunächst werden zwei Extremfälle für die Erzeugung der Gleichungen betrachtet. Für die langen Gleichungen wird für jedes erzeugte Bit z_i rekursiv eine Gleichung aufgestellt. Für die ersten Gleichungen ist dafür keine Rekursion nötig. Bei den späteren Gleichungen wird eine nicht verfügbare Variable als die Gleichung eingefügt, mit der diese Variable berechnet wird.

Man betrachte als Beispiel Bivium. Im Schritt i wird die Gleichung

$$z_i = a_{66-i} + a_{93-i} + b_{162-i} + b_{177-i}$$

hinzugefügt. Für $i \leq 65$ kann dies direkt mit den Variablen des inneren Zustandes beschrieben werden. Für $i = 66$ wird auf a_0 zugegriffen. Diese Variable wurde beim ersten Takt als

$$a_0 = a_{69} + b_{162} + b_{175} \cdot b_{176} + b_{177}$$

erzeugt und wird hier so eingefügt, dass die Darstellung

$$\begin{aligned} z_{66} &= (a_0) + a_{27} + b_{96} + b_{111} \\ &= (a_{69} + b_{162} + b_{175} \cdot b_{176} + b_{177}) + a_{27} + b_{96} + b_{111} \\ &= a_{27} + a_{69} + b_{96} + b_{111} + b_{162} + b_{175} \cdot b_{176} + b_{177} \end{aligned}$$

gilt. Die anderen Gleichungen werden ebenfalls rekursiv erzeugt.

Pro Bit des Ausgabestroms wird eine Gleichung erzeugt, das bedeutet für Bivium werden mindestens 177 Gleichungen in 177 Variablen und für Trivium 288 Gleichungen in 288 Variablen erzeugt. Alle Gleichungen können mit Variablen aus dem inneren Zustand beschrieben werden, es müssen keine zusätzlich eingeführt werden. Zwar steigt der Grad der Gleichungen durch wiederholtes rekursives Einsetzen exponentiell an, allerdings wird ohne weitere Vereinfachungen bei Bivium in der Gleichung für z_{177} nur Grad drei und bei Trivium in der Gleichung für z_{288} nur Grad fünf erreicht.

3.3 Erzeugen kurzer Gleichungen

Pro Takt wird ein Ausgabebit und pro Schieberegister ein neuer Wert erzeugt. Die kurzen Gleichungen beschreiben diesen Sachverhalt. Die neuen Werte für die Schieberegister werden dabei als neue Variablen eingeführt.

Es ergeben sich pro Schritt für Bivium drei Gleichungen und zwei neue Variablen, für Trivium vier Gleichungen und drei neue Variablen. Beispielsweise werden für Trivium im Schritt i folgende vier Gleichungen erzeugt:

$$\begin{aligned} z_i &= c_{66-i} + c_{111-i} + a_{66-i} + a_{93-i} + b_{69-i} + b_{84-i} \\ a_{(-i)} &= c_{66-i} + c_{111-i} + c_{110-i} \cdot c_{109-i} + a_{69-i} \\ b_{(-i)} &= a_{66-i} + a_{93-i} + a_{92-i} \cdot a_{91-i} + b_{78-i} \\ c_{(-i)} &= b_{69-i} + b_{84-i} + b_{83-i} \cdot b_{82-i} + c_{87-i} \end{aligned}$$

Diese Gleichungen sehen für jeden Schritt gleich aus, d.h. der Grad ist konstant zwei und es treten maximal sechs Literale pro Gleichung auf. Allerdings sind mehr Gleichungen und Variablen nötig, um das Gleichungssystem komplett zu beschreiben. Am Ende der Register werden Variablen berechnet, die in keine weitere Berechnung mehr einfließen. Diese Gleichungen müssen nicht aufgestellt werden. Man spart bei Bivium 133 und bei Trivium 198 Gleichungen.

Es wurden die beiden Extreme für die Gleichungserzeugung vorgestellt. Man kann auch eine Variante verwenden, die dazwischen liegt [19].

3.4 Vereinfachung nichtlinearer Gleichungssyteme

Aufgrund der Eigenschaften boolscher Gleichungssysteme aus Abschnitt 2.2 sind bereits triviale Vereinfachungen möglich. Bei den hier betrachteten Gleichungen ergeben sich weitere Möglichkeiten.

3.4.1 Enthaltung

Bei den langen Gleichungen aus Abschnitt 3.2 kommt es vor, dass eine längere Gleichung eine kürzere Gleichung enthält, etwa

$$\begin{aligned} z_1 &= s_{66} + s_{93} + s_{162} + s_{177} \\ z_{151} &= s_{66} + s_{93} + s_{162} + s_{177} + s_{10} \cdot s_{11} + \textit{30 weitere Monome} \end{aligned}$$

Die zweite Gleichung kann zu

$$z_{151} + z_1 = s_{10} \cdot s_{11} + \textit{30 weitere Monome}$$

vereinfacht werden. Bei Bivium können 27, bei Trivium 105 dieser Vereinfachungen vorgenommen werden.

3.4.2 Linearisierung

Bei den Gleichungen treten Monome vom Grad ≥ 2 auf, etwa

$$a_0 = a_{69} + b_{69} + b_{82} \cdot b_{83} + b_{84}$$

Das Monom $b_{82} \cdot b_{83}$ wird durch eine neue Variable v substituiert. Man erhält

$$\begin{aligned} a_0 &= a_{69} + b_{69} + v + b_{84} \\ v &= b_{82} \cdot b_{83} \end{aligned}$$

Die erste Gleichung ist jetzt eine lineare Gleichung, was für Algebrapakete sehr nützlich ist, weniger für SAT Solver. Eine Gleichung der zweiten Bauart kann mit in der Länge des Monoms linear vielen Klauseln dargestellt werden.

Falls das Monom in den Gleichungen mehrmals vorkommt, muss es nur einmal durch v ersetzt werden. Bei weiteren Vorkommen kann die schon erzeugte Variable v verwendet werden. Hier ist dies nur bei den langen Gleichungen nützlich, bei den kurzen tritt jedes Monom zweiten Grades höchstens einmal auf.

3.4.3 Cutting Number

Eine lineare Gleichung der Form

$$s_1 + s_2 + \ldots + s_n$$

wird in 2^{n-1} Klauseln umgesetzt, für jede Belegung mit geradzahliger bzw. ungeradzahligen Anzahl der wahren Literale. Beispielsweise würde

$$s_1 + s_2 + s_3 + s_4 + s_5 + s_6 + s_7 + s_8 + s_9$$

in $2^{9-1} = 256$ Klauseln umgesetzt werden. Diese langen Gleichungen können durch Substitution und neue Variablen verkürzt werden. Die *Cutting number* gibt dabei an,

wieviele Monome eine Gleichung maximal enthalten darf. Bei einem Wert von 4 würde sich die Vereinfachung

$$\begin{aligned} v_1 &= s_1 + s_2 + s_3 \\ v_2 &= s_4 + s_5 + s_6 \\ v_3 &= s_7 + s_8 + s_9 \\ v_4 &= v_1 + v_2 + v_3 \end{aligned}$$

ergeben. Jetzt reichen $4 \cdot 2^{4-1} = 32$ Klauseln. Der Wert für die Cutting number wurde für diese Arbeit auf 3 festgesetzt.

Nach diesen Vereinfachungen besteht das Gleichungssystem aus kurzen Gleichungen bestimmter Struktur. Da die quadratischen Terme nicht entfernt werden können, handelt es sich um eine Instanz des MQ-Problems. Dieses wird als nächstes in ein äquivalentes SAT-Problem überführt.

3.5 KNF Erzeugung

Nach den vorgestellten Vereinfachungen treten Gleichungen folgender Strukturen auf:

$$\begin{aligned} 0 &= a + b \\ 1 &= a + b \\ v &= a + b \\ v &= a \cdot b \\ v &= a \cdot b \cdot c \\ v &= a \cdot b \cdot c \cdot d \\ v &= a \cdot b \cdot c \cdot d \cdot e \end{aligned}$$

wobei die letzten beiden Formeln nur bei den langen Gleichungen bei Trivium auftreten. Die entsprechende Darstellung in KNF kann mit Karnaugh-Veitch-Diagrammen oder dem Quine-McCluskey-Algorithmus berechnet werden, beispielsweise wird $v = a \cdot b$ zu

$$(\neg v \vee a) \wedge (\neg v \vee b) \wedge (v \vee \neg a \vee \neg b).$$

Da die Strukturen feststehen, wurden die entsprechenden Darstellungen in KNF einmalig berechnet und fest einprogrammiert. Die Ausgabe für den SAT Solver erfolgt im DIMACS-Format [8], beispielsweise wird $s_9 = s_7 \cdot s_5 \cdot s_3 \cdot s_1$ zu

```
-9 7 0
-9 5 0
-9 3 0
-9 1 0
 9 -7 -5 -3 -1 0
```

wobei positive Werte für wahre Literale, negative Werte für negierte Literale stehen und die Null das Ende der Klausel kennzeichnet. Es könnten die Konstanten TRUE und FALSE als 1, -2 codiert und in den Gleichungen zur Vereinfachung bei der Formelerzeugung verwendet werden [20]. Bei dem hier verwendeten Verfahren ist dies nicht nötig.

3.6 Variablen raten

Zur Reduktion der Komplexität des Problems wird eine bestimmte Menge der Variablen des inneren Zustandes geraten. Wenn n Variablen $s_i \in \{0, 1\}$ geraten werden, gibt es dafür 2^n mögliche Belegungen, von denen man im Mittel die Hälfte testen muss, bevor man die richtige Belegung findet. Der Vorteil ist, dass ein vereinfachtes Gleichungssystem schneller zu lösen ist. Mit mehr Hardware und damit Kostenaufwand können bis zu 2^n Instanzen eines Problems parallel getestet werden.

Die Experimente zeigen, dass die Ausführungszeit des SAT Solvers erheblich von der gewählten Ratestrategie, weniger von der Anzahl der geratenen Variablen abhängt.

3.6.1 Ratezeitpunkt

Während der Erzeugung der KNF können die Variablen zu verschiedenen Zeitpunkten geraten werden. Raten direkt nach dem Aufstellen der Gleichungen ermöglicht, dass alle weiteren Verarbeitungsschritte schon auf einem einfacheren Gleichungssystem stattfinden. Eine Festlegung während der Vereinfachung und Verkürzung der Gleichungen ermöglicht, die Variablen auszuwählen, deren gesetzter Wert eine maximale Vereinfachung bringt. Das Raten der Variablen, nachdem die KNF erzeugt wurde und bevor diese an den SAT Solver gegeben wird, ermöglicht, die Vorverarbeitungsphase für alle geratene Belegungen nur einmal durchzuführen zu müssen. Hier werden die Variablen gleich nach dem Aufstellen der Gleichungen geraten, um maximale Vereinfachungen zu ermöglichen.

Alternativ könnte man dem SAT Solver die Variablen erst während der Ausführungszeit geben, abhängig davon, welcher Wert den größten Effekt hat.

3.6.2 Ratestrategien

Es bieten sich verschiedene Ratestrategien an. Diese wurden aus Analyse der Gleichungen für Bivium gewonnen. In Abbildung 3.1 bzw. 3.2 sind die Mengen der geratenen Variablen examplarisch für Bivium mit 42 bzw. Trivium mit 144 Variablen dargestellt. Bei

- *beginning* werden die Variablen am Anfang des inneren Zustandes geraten,
- *even distribution beginning* werden die zu ratenden Variablen gleichmäßig auf die Anfänge der einzelnen Register verteilt,
- *less frequent* werden die Variablen geraten, die in den langen Gleichungen am wenigsten häufig vorkommen,
- *ending* werden die Variablen am Ende des inneren Zustandes geraten,
- *even distribution ending* werden die zu ratenden Variablen gleichmäßig auf die Enden der einzelnen Register verteilt,

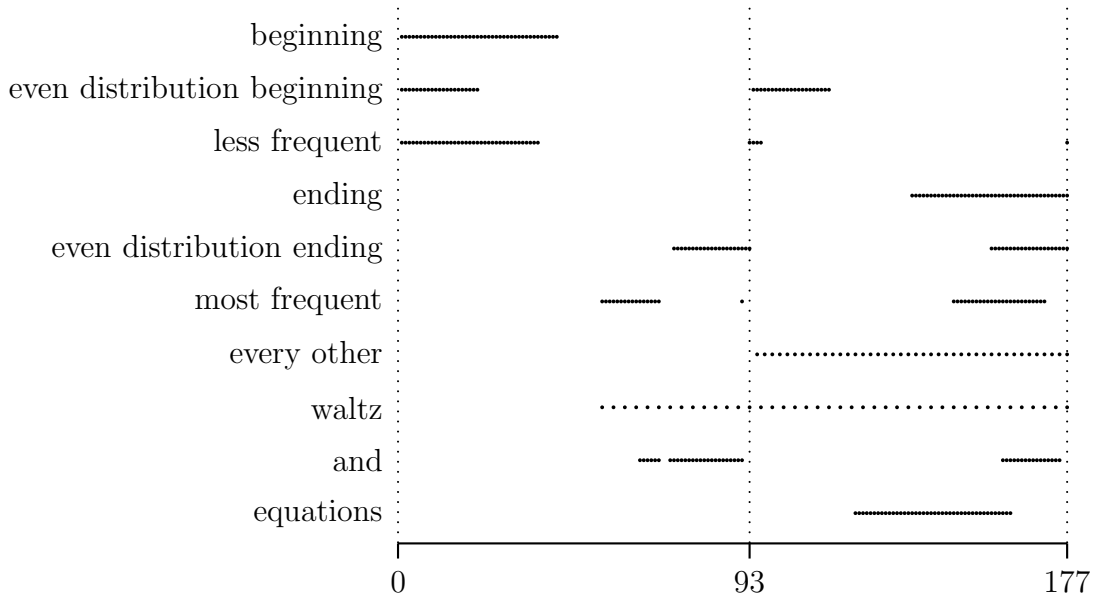

Abbildung 3.1: Ratestrategien für Bivium, 42 Variablen geraten

- *most frequent* werden die Variablen geraten, die in den langen Gleichungen am häufigsten vorkommen,
- *every other* wird jede zweite Variable vom Ende her gezählt geraten,
- *waltz* wird jede dritte Variable vom Ende her gezählt geraten,
- *and* werden die Variablen geraten, die am häufigsten in den Monomen auftreten,
- *equations* werden die Variablen so ausgewählt, dass sich weitere Variablen direkt ableiten lassen.

Die Ratestrategie *beginning* wurde für Vergleichszwecke untersucht. *ending* hat den Vorteil, dass ein Großteil der Gleichungen nur noch zwei unbestimmte Variablen enthält und zu Klauseln mit nur zwei Literalen umgewandelt wird. Der Vorteil von *and* ist, dass einige Gleichungen linear werden [18]. Eine Variante mit den XOR-Gattern wendet [19] an. Dadurch ergibt sich im Gegensatz zu *and* ein Vorteil für den SAT Solver, weil man das Ergebnis von $x \cdot y$ mit einer Wahrscheinlichkeit von $p = 0.75$ richtig raten kann, von $x + y$ dagegen nur mit $p = 0.5$.

Bei *most* und *less frequent* wurden die Häufigkeiten der langen Gleichungen gewählt, da hier viel deutlicher abzulesen ist, welche Variablen den meisten Einfluss haben, beispielsweise erstrecken sich bei Trivium bei den kurzen Gleichungen die Häufigkeiten zwischen 1 und 8, bei den langen Gleichungen zwischen 41 und 102.

Bei *equations* wurde eine Submenge der einfachsten Gleichungen der Form

$$z_i = a + b + c + d$$

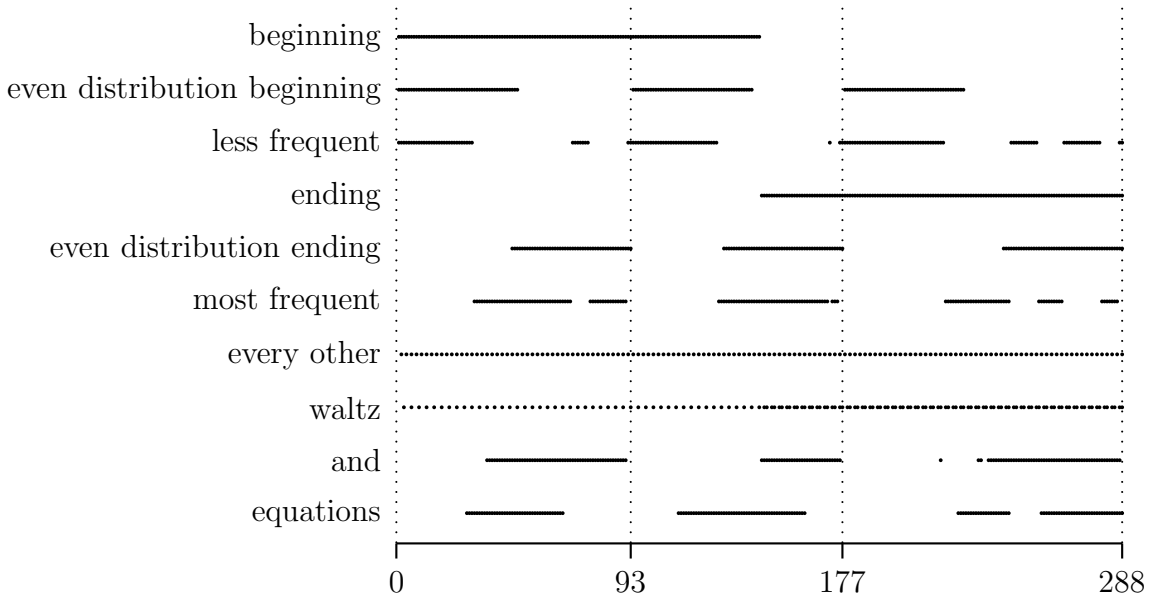

Abbildung 3.2: Ratestrategien für Trivium, 144 Variablen geraten

gewählt, von denen bei Bivium bei den langen Gleichungen 66 und bei den kurzen 177 vorkommen. Durch Raten von drei Variablen ist die Übrige ebenfalls bestimmt. Es gibt verschiedene Varianten, eine Untermenge der Gleichungen auszuwählen. Es wurde diejenige ausgewählt, bei der die meisten Variablen gesetzt werden.

Es ist ebenfalls möglich, statt *even distribution* eine leichte Ungleichverteilung vorzunehmen, da die Register verschieden lang sind.

Vermutlich sind *beginning*, *ending*, *every other* und *waltz* für einen Vergleich zwischen Bivium und Trivium nicht geeignet, da hier zwei und dort drei Register betrachtet werden müssen.

Die Ratestrategien können in folgenden Fragestellungen bzw. Maßzahlen miteinander verglichen werden:

- Wie viele Variablen muss man raten, damit das Gleichungssystem der kurzen Gleichungen aus Abschnitt 3.3 durch rekursives Einsetzen vollständig bestimmt ist?
- Wie groß ist die Klauselmenge bzw. die Anzahl der Variablen in der KNF, wenn n Variablen geraten werden?
- Wie groß ist die Laufzeit des SAT Solver bei n geratenen Variablen?
- Sind die Angriffe auf Bivium und Trivium hinsichtlich der Ratestrategien vergleichbar?

Sind die Antworten bekannt, kann man die optimale Ratestrategie bestimmen.

3.6.3 Anzahl der geratenen Variablen

Wenn die optimale Ratestrategie bestimmt ist, ist die Anzahl der zu ratenden Variablen anzupassen. In [14] wurde für die Ratestrategie *ending* ein Minimum bei 45 Variablen bestimmt, d.h. wenn man mehr oder weniger Variablen n rät, verschlechtert sich die erwartete Gesamtlaufzeit des SAT Solvers.

3.6.4 Korrektheit der geratenen Variablen

Es stellt sich auch die Frage, welche Werte man für die Variablen rät. Seien die korrekten Werte s_i, die geratenen Werte s'_i. Wenn man korrekt rät, also $s'_i = s_i$ für g Variablen setzt, wird der SAT Solver als Ergebnis erfüllbar und ein Modell mit den restlichen Variablen ausgeben. Wenn man zufällige Werte rät, etwa $s'_i =_R \{0, 1\}$, also eine erschöpfende Schlüsselsuche über g Variablen simuliert, sollte das Ergebnis mit hoher Wahrscheinlichkeit unerfüllbar lauten, außer man rät zufällig die richtige Belegung.

Die Werte korrekt zu raten ist angebracht, wenn einige Bits des inneren Zustandes bekannt sind, beispielsweise durch einen *Side-Channel-Angriff*. Die Werte zufällig zu raten ermittelt die Komplexität für eine Suche über einen kleineren Schlüsselraum. Die erwartete Laufzeit über den gesamten Schlüsselraum ergibt sich aus der Laufzeit der Instanz multipliziert mit der Anzahl aller Möglichkeiten, die Variablen zu raten.

Man könnte weiterhin für Testfälle alle Variablen s'_i mit 0 oder 1 initialisieren. Zur Aufdeckung von Symmetrien kann man $s'_i = 1 \oplus s_i$ setzen.

3.7 Klauseln lernen

Für das folgende Experiment soll für wiederholte Ausführungen ausgenutzt werden, dass ein SAT Solver aus Konflikten Klauseln lernen kann. Dafür wurde die Schnittstelle von MiniSat modifiziert [11]. Das normale Interface sieht vereinfacht so aus:

```
S.initialize(g)                    – g Variablen werden geraten
sat = S.solve()
if sat:
    output SAT
else:
    output UNSAT
```

MiniSat liest die Klauseln von der Standardeingabe und ruft dann den SAT Solver auf, dessen Rückgabewert die Erfüllbarkeit angibt.

Die Methode solve() kann auch mit einer Menge von Annahmen aufgerufen werden. In diesem Fall ist der Rückgabewert anders zu interpretieren. Bei TRUE ist ein Modell gefunden und die Klauselmenge samt Annahmen erfüllbar. Bei FALSE ist die Klauselmenge mit den Annahmen nicht erfüllbar, S wurde zurückgesetzt und kann mit neuen Annahmen erneut aufgerufen werden. Es wird angenommen, dass die gelernten Klauseln eines Lösungsversuchs bei weiteren Ausführungen die Ergebnisfindung beschleunigen. Das modifizierte Interface sieht vereinfacht so aus:

```
S.initialize(g − w)                    – g − w Variablen werden geraten
assumptions = [a_1, a_2, a_3, ..., a_2^w]   – 2^w Belegungen für n Variablen
for i from 1 to 2^w do
    sat = S.solve(assumptions[i])
    if sat:
        output SAT
        return
output UNSAT
```

Es werden w weniger Variablen geraten, dafür wird die Lösungsmethode 2^w mal mit den verschiedenen Belegungen für die w Variablen aufgerufen. Wenn eine erfüllende Belegung gefunden wurde, wird das Verfahren mit dem Ergebnis „erfüllbar“ abgebrochen. Wenn alle Annahmen nicht erfüllbar sind, wird „nicht erfüllbar“ ausgegeben.

Für eine Beurteilung dieser Methode wird bei der optimalen Ratestrategie der normale Algorithmus ausgeführt mit n Variablen und mit $n-w$ Variablen. Der modifizierte Algorithmus wird mit $n-w$ gesetzten Variablen und w anzunehmenden Variablen aufgerufen, innerhalb dessen wird für alle Belegungen der w Variablen der normale Algorithmus mit n Variablen ausgeführt. Dabei bleiben die gelernten Klauseln der vorherigen Ausführungen erhalten. Anschließend werden die Laufzeiten der drei Varianten verglichen.

Die vorgestellten Methoden wurden implementiert [21]. Für die Ratestrategien und passende Anzahlen geratener Variablen wurden jeweils mehrere Testinstanzen erstellt und experimentell verschiedene Parameter ermittelt.

4 Auswertung

Um die Varianz der Laufzeiten der SAT Solver abzuschätzen wurden pro Testfall mehrere Instanzen erstellt und getestet. Man hat es hier mit einem Trade-Off zu tun. Wenn man einen breiten Überblick möchte, muss man sich auf wenige Fälle beschränken. Wenn man genaue Werte möchte und sich nur auf wenige Fälle konzentriert, muss man in Kauf nehmen, dass man unter Umständen Randfälle übersieht, die sehr geringe Laufzeiten haben.

In dieser Arbeit wurde für den breiten Überblick entschieden. Für die Ratestrategien wurde pro Testfall fünf Instanzen durchgerechnet. Das ist sehr wenig, aber vertretbar, da die Varianz der erhaltenen Werte gering ist. Trotzdem muss ein kleiner Fehler in Kauf genommen werden.

4.1 Auswertung der Gleichungssysteme

Für einen minimalen Schlüsselstrom wurden die kurzen und die langen Gleichungssysteme für Bivium und Trivium aufgestellt. Sie wurden zunächst ohne Variablen zu raten hinsichtlich Anzahl der Variablen, der Anzahl der Gleichungen, der Anzahl der Monome, der Gesamtlänge der Gleichungen und dem maximalen Grad verglichen. Die Gesamtlänge ist die Summe der Monome pro Gleichung. Bei der Anzahl der Monome werden mehrfach auftretende Monome nur einmal gezählt. Diese Zahl ist besser, weil wiederholt auftretende Monome nur einmal substituiert werden. Das Ergebnis steht in Tabelle 4.1.

	Bivium ($n = 177$)		Trivium ($n = 288$)	
	kurz	lang	kurz	lang
Anzahl Variablen	398	177	954	288
Anzahl Gleichungen	398	177	954	288
Anzahl Monome	620	889	1621	45979
Gesamtlänge	1898	2462	5207	82253
Maximaler Grad	2	3	2	5

Tabelle 4.1: Vergleich Bivium/Trivium hinsichtlich langer und kurzer Gleichungen bei minimalem Schlüsselstrom, n: Länge des inneren Zustandes

Bei den kurzen Gleichungen gibt es bei Bivium und Trivium mehr Variablen und Gleichungen, dafür weniger Monome und die Gleichungen sind insgesamt kürzer. Bei

Bivium ist der Unterschied gering, bei Trivium sind die langen Gleichungen um den Faktor 16 größer und scheinen damit auch die 16fache Komplexität zu haben. Dies liegt vor allem am höheren Grad der Gleichungen.

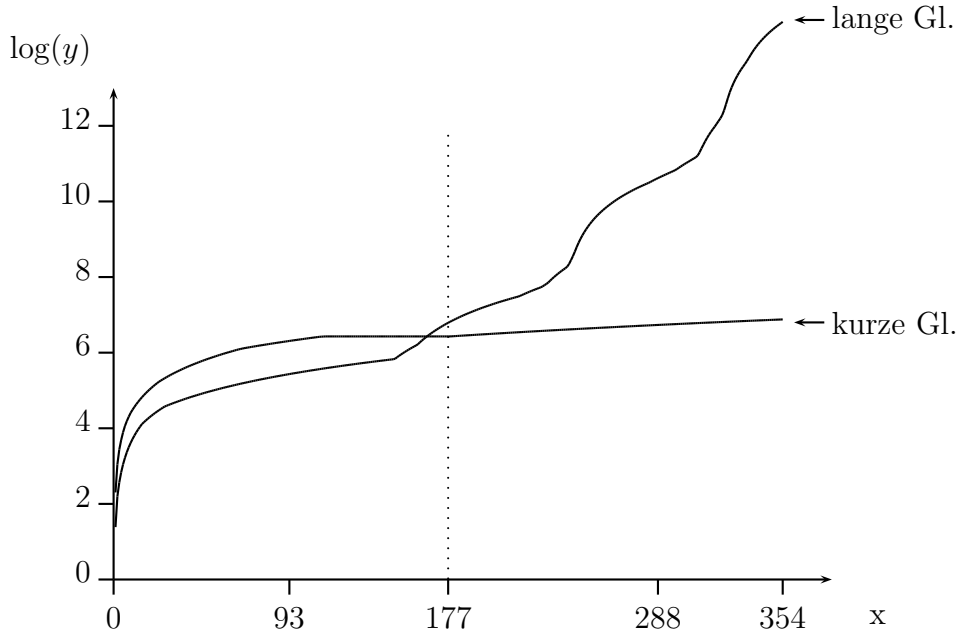

Abbildung 4.1: Bivium. Anzahl der Monome bei den langen und kurzen Gleichungen, x: Anzahl der Gleichungen, y: Anzahl der Monome

Zur Verdeutlichung ist in Abbildung 4.1 der Logarithmus für die Anzahl der Monome y gegen die Länge des Schlüsselstrom x für Bivium aufgetragen. Man erkennt, dass die Werte für einen Schlüsselstrom von 177 Bit bei den kurzen und langen Gleichungen etwa gleich sind, danach steigen die Werte für die langen Gleichungen exponentiell an. Abbildung 4.2 zeigt dies für Trivium. Bei einem Schlüsselstrom von 288 Bit sind die kurzen Gleichungen erheblich einfacher als die langen Gleichungen.

Offensichtlich sollte man die Anzahl der Monome und die Gesamtlänge unter den Testbedingungen genauer betrachten. Für die Experimente wird eine bestimmte Anzahl von Variablen geraten, um die Laufzeiten im durchführbaren Rahmen zu halten. Der Vergleich der Anzahl der Monome wurde für 42 geratene Variablen bei Bivium und 144 geratene Variablen bei Trivium wiederholt. Das Ergebnis steht in Tabelle 4.2. Für Bivium sind die langen Gleichungen nun günstiger, für Trivium konnte der Unterschied etwa auf den Faktor fünf gesenkt werden.

Weitere Analysen zeigen, dass für die langen Gleichungen die Anzahl der Monome und damit die Länge der Gleichungen mit der Anzahl der geratenen Variablen stark abfällt, siehe Abbildungen 4.3 und 4.4.

Anhand dieser Daten ist anzunehmen, dass für Bivium mit 42 geratenen Variablen die langen Gleichungen bessere Ergebnisse zeigen müssten. Bei den langen Gleichungen

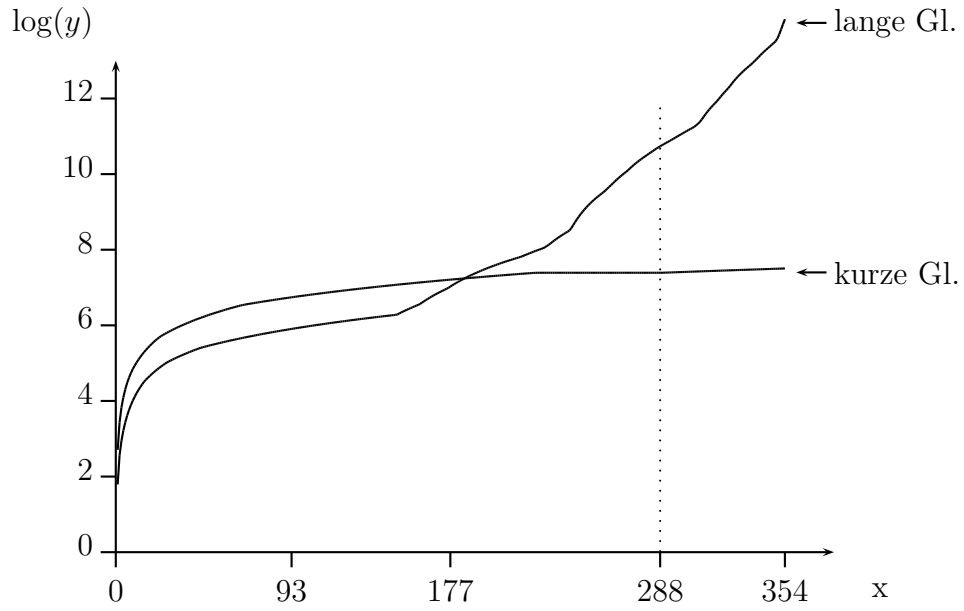

Abbildung 4.2: Trivium. Anzahl der Monome bei den langen und kurzen Gleichungen, x: Anzahl der Gleichungen, y: Anzahl der Monome

ist dieser Effekt stärker, da weniger Variablen vorhanden sind und damit mehr Monome betroffen sind, die durch auf 0 geratene Variablen wegfallen.

Mit der Formel aus Abschnitt 2.4 kann die Anzahl der maximal möglichen Monome berechnet werden. Für Bivium mit $n = 177$ Variablen mit maximalen Grad $k = 3$ sind 15 754 Monome möglich, für Trivium mit $n = 288$ und $k = 5$ sind 284 701 993 Monome möglich. Mit den Werten aus den Tabellen 4.1 und 4.2 berechnet man, wie dünn besetzt die entsprechenden Gleichungssysteme sind. Für Bivium gilt $\beta \approx 0.16$, für Trivium gilt $\beta \approx 0.0003$. Damit ist nach Definition aus [2] zwar nur das Gleichungssystem für Trivium dünn besetzt, aber die Ergebnisse rechtfertigen, dass die Methode auch für Bivium angewendet wird.

4.2 Auswertung der KNF

Für jede Stromchiffre und Ratestrategie wurde für eine bestimmte Anzahl zu ratender Variablen die KNF-Darstellungen untersucht. In Tabelle A.2 sind die Werte dargestellt.

Wenn sich die Laufzeit des SAT Solvers nur nach der Größe der KNF richten würde, müßten für jede Ratestrategie die kurzen Gleichungen deutlich schnellere Laufzeiten zeigen als die langen Gleichungen. Für Bivium ist bei *most frequent* die KNF mit ≈ 3520 Klauseln und ≈ 1920 Variablen am kleinsten, es folgen *waltz* und *even distribution beginning*. Für Trivium liegt das Minimum bei *even distribution beginning* mit ≈ 9500 Klauseln und ≈ 5100 Variablen, es folgen *most frequent* und *equations*. Eine Korrelation

	Bivium ($n = 177, g = 42$)		Trivium ($n = 288, g = 144$)	
	kurz	lang	kurz	lang
Anzahl Monome	535	439	1333	7105
Gesamtlänge	1726	1522	4576	20679

Tabelle 4.2: Vergleich Bivium/Trivium hinsichtlich langer und kurzer Gleichungen bei minimalem Schlüsselstrom mit geratenen Variablen, n: Länge des inneren Zustandes, g: Anzahl der geratenen Variablen

zu den Laufzeiten der SAT Solver läßt sich nur bei Trivium beobachten.

4.3 Vergleich der Ratestrategien

In [7] wird angegeben, dass 195 Variablen ausreichen, um das kurze Gleichungssystem für Trivium komplett nur durch rekursives Einsetzen zu bestimmen. Dieses Maß wurde hier für die verschiedenen Ratestrategien berechnet. Aus Tabelle A.1 geht hervor, bei welcher Ratestrategie man wieviele Variablen minimal benötigt, um das Gleichungssystem vollständig zu bestimmen. Diesen Wert kann man als Maß dafür nehmen, inwiefern die geratenen Variablen auf die anderen Variablen einwirken.

Für Trivium liegt das Minimum bei der Ratestrategie *and* mit 192 Variablen, gefolgt von *even distribution ending* mit 195 Variablen. Am wenigsten Einfluss haben die Variablen der Ratestrategie *less frequent*, es sind 224 Variablen nötig. Für Bivium ist die Ratestrategien *equations* mit 76 nötigen Variablen eindeutig Sieger. Es folgen *even distribution beginning* und *less frequent*, am schlechtesten schneidet *beginning* ab. Man sieht hier, dass Bivium und Trivium in dieser Hinsicht nicht vergleichbar sind.

Für die folgenden Messungen wurde auf eine Nachkommastelle gerundet, da mehr Stellen nicht signifikant sind. Die Rechnungen wurden jeweils mit MiniSat und RSat durchgeführt. Im Mittel ist MiniSat um etwa 10% schneller, so dass nur diese Werte hier dargestellt werden. Werte kleiner als eine Sekunde sind nicht dargestellt. Es wurde versucht, die Anzahl der zu ratenden Variablen so anzupassen, dass die Laufzeit einer Instanz etwa eine Minute beträgt, um einerseits statistisch sinnvolle Aussagen zu treffen und andererseits möglichst viele Messungen zu ermöglichen. Die Ergebnisse der Messungen stehen im Anhang in den Tabellen Tabelle A.3 bis A.6.

In Tabelle A.3 werden für Bivium die Ratestrategien betrachtet, in denen korrekt (SAT), in Tabelle A.4 in denen zufällig (UNSAT) geraten wurde. Insgesamt sind die Laufzeiten für UNSAT etwas schlechter, da ein SAT Solver erst den gesamten Suchraum explorieren muss, bevor er sicher sein kann, dass keine Lösung existiert.

Für die meisten Ratestrategien sind die Laufzeiten der langen und kurzen Gleichungen etwa vergleichbar, außer für *less frequent*, *even distribution beginning* und *beginning*. Dies sind die Fälle, in denen selten vorkommende Variablen geraten wurden, so dass weniger Monome entfernt wurden. *waltz* nimmt einen Sonderfall ein. Zwar ist der Unterschied

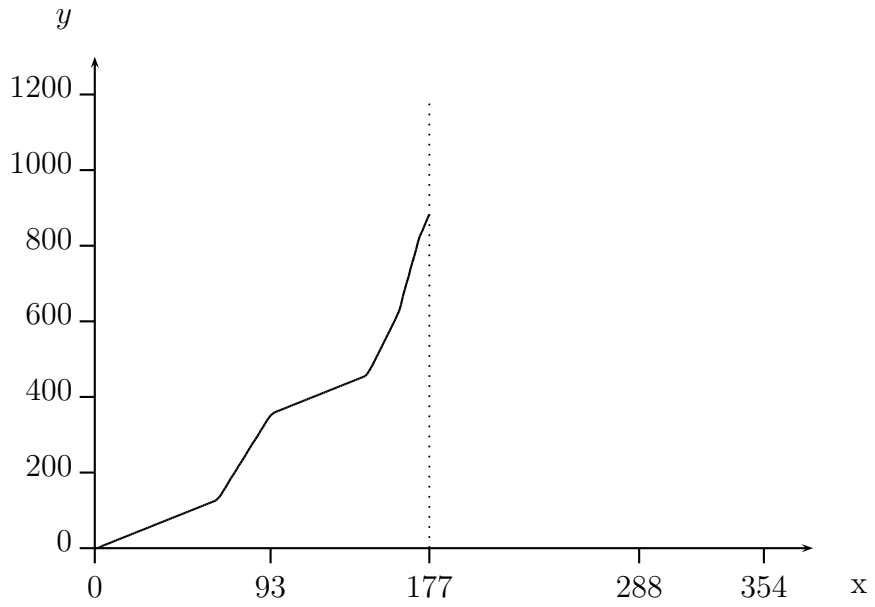

Abbildung 4.3: Bivium. Anzahl der Monome bei den langen Gleichungen, wenn Variablen geraten werden. x: Anzahl der nicht geratenen Variablen, y: Anzahl der Monome

zwischen den kurzen und langen Gleichungen ebenfalls groß, aber die Werte für die langen Gleichungen scheinen unabhängig von der Anzahl der geratenen Variablen zu sein. Der Effekt tritt bei weniger geratenen Variablen nicht mehr auf. Außerdem ist dies die mit Abstand langsamste Ratestrategie.

Die Ratestrategien mit den besten Laufzeiten sind *ending* und *equations*, dicht gefolgt von *even distribution ending*. Es ergibt sich eine erwartete Laufzeit für SAT von 2^{44} Sekunden und für UNSAT von 2^{46}. Dies deckt sich mit den Aussagen in den vorherigen Arbeiten.

Die Ergebnisse für Trivium stehen in Tabelle A.5 für korrekt, in Tabelle A.6 für zufällig geratene Variablen. Wie bei Bivium sind die nicht erfüllbaren Fälle etwas langsamer.

Größere Unterschiede in den Laufzeiten zwischen den kurzen und den langen Gleichungen treten wie bei Bivium bei *less freq*, *even distribution beginning* und *beginning* und zusätzlich bei *most frequent* auf. Die schnellsten Ratestrategien sind für den erfüllbaren Fall *equations*, *even distribution beginning*, *even distribution ending* und *most frequent* jeweils bei den kurzen Gleichungen mit einer erwarteten Laufzeit von bis zu 2^{149} Sekunden. Für den nicht erfüllbaren Fall werden für *equations* bis zu 2^{150} Sekunden erreicht. *waltz* liegt nun im Mittelfeld, wahrscheinlich weil anteilig mehr Variablen geraten werden.

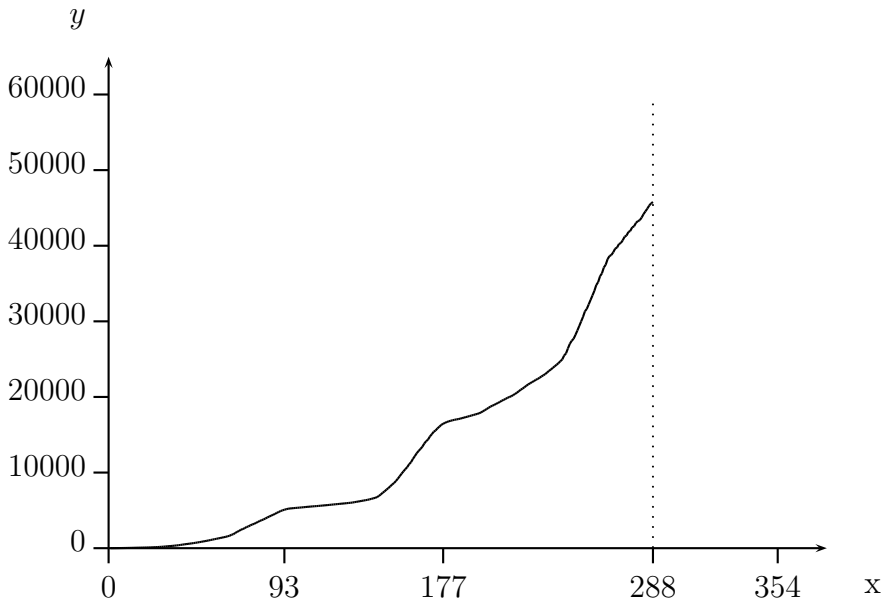

Abbildung 4.4: Trivium. Anzahl der Monome bei den langen Gleichungen, wenn Variablen geraten werden. x: Anzahl der nicht geratenen Variablen, y: Anzahl der Monome

4.4 Ergebnis Klauseln lernen

Die Idee des Klauseln lernens wurde für Bivium in der Ratestrategie *ending* für den erfüllbaren und den nicht erfüllbaren Fall getestet. Es wurden je 30 Testfälle berechnet.

Für den erfüllbaren Fall beträgt die gemittelte Laufzeit des normalen SAT Solvers bei 36 geratenen Variablen 372.1 Sekunden, dies entspricht einer erwarteten Gesamtlaufzeit von 2^{45} Sekunden. Iteriert man zusätzlich über die Belegungen weiterer 10 Variablen, so beträgt die Gesamtlaufzeit der Modifikation im Mittel 485.3 Sekunden, was einer nur wenig höheren Gesamtlaufzeit entspricht. Wenn man diese 10 Variablen auch vorgibt, ergibt sich eine gemittelte Laufzeit von 1.3 Sekunden pro Instanz, was einer Gesamtlaufzeit von 2^{46} Sekunden entspricht.

Für den nicht erfüllbaren Fall und 38 geratenen Variablen benötigt MiniSat im Mittel 238.1 Sekunden, dies entspricht einer Gesamtlaufzeit von 2^{46} Sekunden. Für 44 Variablen werden im Mittel 8.7 Sekunden benötigt, dies entspricht einer Gesamtlaufzeit von 2^{47} Sekunden. Der modifizierte SAT Solver benötigt für 38 zufällig geratene Variablen und 6 zusätzlich anzunehmenden Variabeln im Mittel 563.2 Sekunden, was zu einer Gesamtlaufzeit von 2^{47} Sekunden führt.

Die schnellste Variante scheint zu sein, dem SAT Solver nur wenig Variablen vorzugeben. Andererseits ist dieses Ergebnis aufgrund der geringen Stichprobenanzahl und der geringen Laufzeit im SAT-Fall bei 46 Variablen als vorläufig zu betrachten. Ebenso

sollte man die Untersuchung über alle Ratestrategien, den nicht erfüllbaren Fall und auch für Trivium ausdehnen.

4.5 Fazit

Die untersuchten Fragestellungen lassen sich wie folgt beantworten:

- *Wie viele Variablen muss man raten, damit das Gleichungssystem der kurzen Variablen aus Abschnitt 3.3 durch rekursives Einsetzen vollständig bestimmt ist?*

  Mit den verwendeten Ratestrategien sind für Bivium 76 Variablen und für Trivium 192 Variablen nötig, um das Gleichungssystem vollständig zu bestimmen. Dieser Wert ist allerdings nicht als untere Grenze anzusehen. Bei einer nicht betrachteten Ratestrategie könnten weniger Variablen nötig sein.

- *Wie groß ist die Klauselmenge bzw. die Anzahl der Variablen in der KNF, wenn n Variablen geraten werden?*

  Für Bivium liegt das Minimum bei etwa 3500 Klauseln und 1900 Variablen, wenn 44 Variabeln geraten werden. Für Trivium und 144 geratenen Variablen liegt das Minimum bei 9500 Klauseln und 5100 Variablen.

  Für eine detailierte Übersicht wird auf Tabelle A.2 verwiesen.

- *Wie groß ist die Laufzeit des SAT Solver bei n geratenen Variablen?*

  Wenn die Variablen korrekt geraten werden können, kann für Bivium eine Gesamtlaufzeit von 2^{44} Sekunden bei 36 gesetzten Variablen, für Trivium von 2^{149} bei 144 gesetzten Variablen erzielt werden. Wenn die Variablen zufällig geraten werden müssen, ergibt sich für Bivium eine Gesamtlaufzeit von 2^{46} bei 44 geratenen Variablen und für Trivium von 2^{150} bei ebenfalls 144 geratenen Variablen. Für die detailierte Auflistung wird auf die Tabellen A.3 bis A.6 verwiesen.

- *Sind die Angriffe auf Bivium und Trivium hinsichtlich der Ratestrategien vergleichbar?*

  Bei den Ratestrategien zeigen sich teilweise unerwartete Unterschiede. Es werden daher nur die mit den schnellsten Laufzeiten betrachtet.

  Die beste Ratestrategie für Bivium ist *ending*. Diese liegt bei Trivium nur im Mittelfeld. Dies erklärt sich durch den Sprung von zwei auf drei Register und die veränderte Struktur der Gleichungen. Die zweitebeste Ratestrategie *equations* für Bivium ist die schnellste für Trivium. Die Annahme, dass man die Variablen der einfachsten Gleichungen setzen muss, zeigt für beide Stromchiffren gute Ergebnisse. Daher sind die Angriffe für diese Ratestrategie vergleichbar.

- *Kann durch die Methode des „Klauseln lernens“ eine Verbesserung erzielt werden?*

  Das vorläufige Ergebnis spricht dagegen. Allerdings sind mehr Messungen auf einer breiteren Ebene nötig.

4.6 Alternative: Gröbnerbasen

Das verwendete Algebrapaket PolyBoRi wurde zur schnellen Berechnung von *Gröbnerbasen* über boolschen Gleichungssystemen geschaffen. Es folgen einige Messwerte für die verschiedenen Ratestrategien. Für den theoretischen Hintergrund wird auf die Literatur verwiesen [23, 4].

McDonald et. al. ermitteln mit Magma und dem F4-Algorithmus für Bivium eine Komplexität von 2^{68} und für Trivium von 2^{189} [19]. Eibach et. al geben für Bivium mit 60 geratenen Variablen eine Dauer von $2^{59.8}$ Sekunden an [13]. Sie verbessern dieses Ergebnis mit 42 geratenen Variablen auf $2^{37.7}$ Sekunden [15].

In Tabelle A.7 ist ein Vergleich der Ratestrategien gezeigt. Für die Ratestrategie *even distribution ending* wurde eine Gesamtlaufzeit von 2^{56} Sekunden geschätzt. Allerdings wurde nur für Bivium pro Ratestrategie die Anzahl der Variablen gesucht, ab der die Laufzeit ungünstig groß wurde, und dann eine Messung durchgeführt. Eine eventuelle Streuung wird also nicht gemessen.

A Experimentelle Ergebnisse

	Bivium	Trivium
most frequent	91	198
and	87	192
even distribution ending	84	195
less frequent	82	224
even distribution beginning	82	204
equations	76	206
beginning	101	199
every other	93	204
waltz	91	208
ending	88	199

Tabelle A.1: Minimale Anzahl der Variablen, um die Lösung des Gleichungssystems nur durch rekursives Einsetzen zu bestimmen.

		Bivium ($g = 44$)			Trivium ($g = 144$)		
		c	t	v	c	t	v
most frequent	kurz	**3 518**	**2 879**	**1 919**	**9 916**	**8 232**	**5 326**
	lang	3 901	3 247	2 063	62 335	50 841	31 374
and	kurz	4 211	3 551	2 264	10 662	8 979	5 685
	lang	4 650	4 042	2 448	63 861	54 449	32 691
e.d. ending	kurz	3 920	3 262	2 122	10 355	8 681	5 520
	lang	4 466	3 800	2 353	63 003	52 584	32 194
less frequent	kurz	3 889	3 174	2 127	11 281	9 494	6 005
	lang	8 826	7 018	4 642	218 090	162 855	10 7019
e.d. beginning	kurz	**3 776**	**3 070**	**2 068**	**9 470**	**7 731**	**5 111**
	lang	8 762	6 942	4 612	189 777	138 904	91 954
equations	kurz	3 922	3 192	2 150	**10 295**	**8 531**	**5 524**
	lang	6 714	5 426	3 516	85 663	67 878	42 674
beginning	kurz	3 906	3 188	2 136	11 015	9 170	5 901
	lang	8 642	6 904	4 538	178 972	134 903	87 580
every other	kurz	3 845	3 233	2 072	11 494	9 982	6 024
	lang	6 615	5 439	3 474	102 450	85 667	52 792
waltz	kurz	**3 630**	**3 101**	**1 954**	11 190	9 661	5 930
	lang	5 233	4 425	2 741	94 276	77 410	48 170
ending	kurz	4 066	3 342	2 218	11 738	9 902	6 246
	lang	5 574	4 595	2 924	110 787	88 760	55 142

Tabelle A.2: Anzahl der Klauseln c, Anzahl der mindestens 3-er Klauseln t und Anzahl der Variablen v für die verschiedenen Ratestrategien bei 44 bzw. 144 geratenen Variablen, Median aus 5 Instanzen

		$g = 60$	$g = 56$	$g = 52$	$g = 48$	$g = 44$	$g = 40$	$g = 36$	$e \cdot 2^g$
most freq.	kurz	–	–	–	1.4	22.2	221.1	–	2^{48}
	lang	–	–	–	2.2	29.3	248.0	–	2^{48}
and	kurz	–	–	15.2	122.0	314.3	–	–	2^{53}
	lang	–	–	2.6	34.8	147.3	–	–	2^{51}
e.d. end.	kurz	–	–	–	2.5	9.4	154.4	–	2^{47}
	lang	–	–	–	1.2	3.6	207.0	–	2^{46}
less freq.	kurz	–	–	1.7	24.7	609.4	–	–	2^{53}
	lang	12.9	23.7	438.9	–	–	–	–	2^{61}
e.d. beg.	kurz	–	–	–	2.2	19.9	357.6	–	2^{48}
	lang	2.9	9.8	71.8	–	–	–	–	2^{58}
equations	kurz	–	–	–	–	3.7	28.8	315.2	$\mathbf{2^{44}}$
	lang	–	–	–	–	8.5	63.0	777.2	2^{46}
beginning	kurz	–	–	1.7	6.0	171.6	–	–	2^{51}
	lang	–	11.2	79.3	–	–	–	–	2^{58}
ev. other	kurz	–	1.2	5.5	37.8	118.7	–	–	2^{51}
	lang	–	–	5.4	33.6	294.6	–	–	2^{52}
waltz	kurz	154.8	–	–	–	–	–	–	2^{67}
	lang	426.2	339.2	459.4	–	–	–	–	2^{61}
ending	kurz	–	–	–	–	5.3	44.2	258.2	$\mathbf{2^{44}}$
	lang	–	–	–	–	2.5	30.0	301.1	$\mathbf{2^{44}}$

Tabelle A.3: Bivium: Laufzeiten e der Ratestrategien in Sekunden für g korrekt geratene Variablen (SAT)

		$g = 60$	$g = 56$	$g = 52$	$g = 48$	$g = 44$	$g = 40$	$e \cdot 2^g$
most frequent	kurz	–	1.1	1.4	5.1	44.5	551.2	2^{49}
	lang	–	–	–	4.2	49.2	–	2^{50}
and	kurz	–	1.1	19.6	238.8	–	–	2^{56}
	lang	–	–	12.7	127.9	–	–	2^{55}
e.d. ending	kurz	–	–	–	4.2	31.4	358.7	2^{48}
	lang	–	–	–	2.5	17.4	315.5	2^{48}
less frequent	kurz	–	–	3.1	41.1	1054.1	–	2^{53}
	lang	22.9	52.0	1505.6	–	–	–	2^{62}
e.d. beginning	kurz	–	–	–	5.5	70.3	–	2^{50}
	lang	5.6	28.5	322.1	–	–	–	2^{60}
equations	kurz	–	–	–	–	8.3	105.1	$\mathbf{2^{47}}$
	lang	–	–	–	–	18.2	137.6	$\mathbf{2^{47}}$
beginning	kurz	–	–	2.4	24.1	352.9	–	2^{52}
	lang	–	8.8	144.8	–	–	–	2^{59}
every other	kurz	–	4.0	7.2	68.6	–	–	2^{54}
	lang	–	4.0	8.5	115.3	–	–	2^{55}
waltz	kurz	286.6	–	–	–	–	–	2^{68}
	lang	393.0	–	–	–	–	–	2^{68}
ending	kurz	–	–	–	–	10.6	104.8	$\mathbf{2^{47}}$
	lang	–	–	–	–	4.6	108.4	$\mathbf{2^{46}}$

Tabelle A.4: Bivium: Laufzeiten e der Ratestrategien in Sekunden für g zufällig geratene Variablen (UNSAT)

		$g = 168$	$g = 162$	$g = 156$	$g = 150$	$g = 144$	$e \cdot 2^g$
most frequent	kurz	–	–	1.5	13.5	94.6	$\mathbf{2^{150}}$
	lang	–	1.5	6.1	56.1	–	2^{155}
and	kurz	–	–	13.7	–	–	2^{159}
	lang	–	5.4	56.4	–	–	2^{161}
e.d. ending	kurz	–	–	–	10.3	65.3	$\mathbf{2^{150}}$
	lang	–	–	5.5	33.7	404.8	2^{152}
less frequent	kurz	–	32.7	325.1	–	–	2^{164}
	lang	381.8	–	–	–	–	2^{176}
e.d. beginning	kurz	–	–	–	8.2	54.5	$\mathbf{2^{149}}$
	lang	–	307.7	–	–	–	2^{170}
equations	kurz	–	–	–	1.6	37.9	$\mathbf{2^{149}}$
	lang	–	–	3.0	14.6	200.9	2^{151}
beginning	kurz	–	–	3.6	12.7	–	2^{153}
	lang	–	17.5	16.6	109.2	–	2^{156}
every other	kurz	–	38.1	317.1	–	–	2^{164}
	lang	–	33.6	–	–	–	2^{167}
waltz	kurz	–	–	9.5	–	–	2^{159}
	lang	–	1.2	63.7	–	–	2^{161}
ending	kurz	–	–	5.6	168.9	–	2^{157}
	lang	–	2.6	39.4	516.3	–	2^{159}

Tabelle A.5: Trivium: Laufzeiten e der Ratestrategien in Sekunden für g korrekt geratene Variablen (SAT)

	$g =$	165	162	159	156	153	150	147	144	$e \cdot 2^g$
most freq.	kurz	–	–	–	3.3	6.6	26.0	62.4	216.5	$\mathbf{2^{152}}$
	lang	2.9	2.6	5.3	16.0	19.9	76.1	–	–	2^{156}
and	kurz	–	2.2	5.1	26.9	145.0	–	–	–	2^{160}
	lang	3.6	9.6	28.4	111.6	–	–	–	–	2^{163}
e.d. ending	kurz	–	–	–	2.0	7.1	11.2	49.3	196.7	$\mathbf{2^{152}}$
	lang	–	1.5	3.4	11.5	36.3	75.0	–	–	2^{156}
less freq.	kurz	–	63.2	168.8	–	–	–	–	–	2^{166}
	lang	1192.3	–	–	–	–	–	–	–	2^{175}
e.d. beg.	kurz	–	–	–	1.8	3.7	11.5	37.6	241.6	$\mathbf{2^{152}}$
	lang	54.1	234.0	–	–	–	–	–	–	2^{170}
equations	kurz	–	–	–	–	–	1.6	9.1	57.3	$\mathbf{2^{150}}$
	lang	–	1.0	2.3	7.1	13.8	22.8	–	–	2^{155}
beginning	kurz	–	–	1.5	3.4	9.3	8.1	377.7	–	$\mathbf{2^{153}}$
	lang	6.0	30.2	29.7	36.0	57.8	154.8	–	–	2^{157}
ev. other	kurz	–	46.4	91.1	–	–	–	–	–	2^{166}
	lang	45.2	214.3	–	–	–	–	–	–	2^{170}
waltz	kurz	–	–	–	7.6	67.8	–	–	–	2^{159}
	lang	–	–	7.1	25.0	785.9	–	–	–	2^{161}
ending	kurz	–	–	2.3	11.0	72.9	–	–	–	2^{159}
	lang	1.1	4.4	20.3	80.5	–	–	–	–	2^{162}

Tabelle A.6: Trivium: Laufzeiten e der Ratestrategien in Sekunden für g zufällig geratene Variablen (UNSAT)

	g	e	$2^g \cdot e$
even distribution ending	50	368.7 s	$\mathbf{2^{56}}$
equations	56	9.6 s	2^{60}
and	56	109.8 s	2^{63}
every other	57	636.0 s	2^{66}
ending	57	1428.1 s	2^{68}
most frequent	60	828.9 s	2^{70}
waltz	64	168.5 s	2^{71}
beginning	68	192.8 s	2^{76}
even distribution beginning	80	60.4 s	2^{86}
less frequent	80	91.3 s	2^{87}
even distribution beginning	78	1480.4 s	2^{89}

Tabelle A.7: Messwerte zur Berechnung der Gröbnerbasis bei Bivium für verschiedene Ratestrategien

Literaturverzeichnis

[1] Steve Babbage, Christophe De Cannière, Anne Canteaut, Carlis Cid, Henri Gilbert, Thomas Johansson, Matthew Parker, Bart Preneel, Vincent Rijmen, and Matthew Robshaw. The eStream Portfolio, 2008.
http://www.ecrypt.eu.org/stream/portfolio.pdf.

[2] Gregory V. Bard. *Algorithms for Solving Linear and Polynomial Systems of Equations over Finite Fields with Applications to Cryptanalysis*. PhD thesis, University of Maryland, 2007. http://www.cs.umd.edu/~jkatz/THESES/bard_thesis.pdf.

[3] Gregory V. Bard, Nicolas T. Courtois und Chris Jefferson. Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers. *Cryptology ePrint Archive, Report 2007/024*, 2007.
http://eprint.iacr.org/2007/024.

[4] Michael Brickenstein und Alexander Dreyer. PolyBoRi: A Gröbner Basis Framework for Boolean Polynomials. *Reports of Fraunhofer ITWM, No. 122*, 2007.
http://www.itwm.fhg.de/zentral/download/berichte/bericht122.pdf.

[5] Christophe De Cannière und Bart Preneel. TRIVIUM – A Stream Cipher Construction Inspired by Block Cipher Design Principles. *eStream*, 2005.
http://www.ecrypt.eu.org/stream/papersdir/2006/021.pdf.

[6] Christophe De Cannière und Bart Preneel. TRIVIUM – Specifications. *eStream*, 2005. http://www.ecrypt.eu.org/stream/p3ciphers/trivium/trivium_p3.pdf.

[7] Christophe De Cannière und Bart Preneel. TRIVIUM. In Matthew Robshaw und Olivier Billet, Herausgeber, *New Stream Cipher Designs – The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 244–266. Springer, 2008.

[8] Satisfiability Suggested Format, 1993.
http://www.satlib.org/Benchmarks/SAT/satformat.ps.

[9] ecrypt forum, 2007. http://www.ecrypt.eu.org/stream/phorum/read.php?1,448.

[10] Niklas Eén und Armin Biere. Effective Preprocessing in SAT through Variable and Clause Elimination. In Fahiem Bacchus und Toby Walsh, Herausgeber, *Theory and Applications of Satisfiability Testing – SAT 2005*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2005. http://minisat.se/SatELite.html.

[11] Niklas Eén und Niklas Sörensson. An extensible SAT-solver. In *Proceedings of SAT '03*, pages 502–518, 2003.
http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/.

[12] Tobias Eibach und Enrico Pilz. Ein Angriff auf Bivium mittels SAT Solver. 7. Kryptotag, GI technical report, 2007.

[13] Tobias Eibach, Enrico Pilz und Sebastian Steck. Comparing and Optimising Two Generic Attacks on Bivium. In *Proceedings of SASC'08 – The State of the Art of Stream Ciphers*, pages 57–68, 2008.

[14] Tobias Eibach, Enrico Pilz und Gunnar Völkel. Attacking Bivium Using SAT Solvers. In H. Kleine Büning und X. Zhao, Herausgeber, *Theory and Applications of Satisfiability Testing – SAT 2008*, volume 4996 of *Lecture Notes in Computer Science*, pages 63–76. Springer, 2008.

[15] Tobias Eibach und Gunnar Völkel. Optimising Gröbner Bases on Bivium, 2008.

[16] Kris Gaj, Gabriel Southern und Ramakrishna Bachimanchi. Comparison of hardware performance of selected Phase II eStream candidates. *eStream*, 2007.
http://www.ecrypt.eu.org/stream/papersdir/2007/026.pdf.

[17] Carla P. Gomes, Henry Kautz, Ashish Sabharwal und Bart Selman. *Satisfiability Solvers*, chapter 2, pages 89–134. Handbook of Knowledge Representation. Elsevier, 2008.
http://www.cs.rochester.edu/u/kautz/papers/SATSolvers-KR-Handbook.pdf.

[18] Alexander Maximov und Alex Biryukov. Two Trivial Attacks on TRIVIUM. *eStream*, 2007. http://www.ecrypt.eu.org/stream/papersdir/2007/003.pdf.

[19] Cameron McDonald, Chris Charnes und Josef Pieprzyk. An Algebraic Analysis of Trivium Ciphers based on the Boolean Satisfiability Problem. *Cryptology ePrint Archive, Report 2007/129*, 2007. http://eprint.iacr.org/2007/129.

[20] Cameron McDonald, Chris Charnes und Josef Pierprzyk. Attacking Bivium with MiniSat. *eStream*, 2007.
http://www.ecrypt.eu.org/stream/papersdir/2007/040.pdf.

[21] Enrico Pilz. Implementierung zu dieser Arbeit, 2008.
http://www.enricopilz.de/studium.html.

[22] Knot Pipatsrisawat und Adnan Darwiche. RSat 2.0: SAT Solver Description. Technical Report D–153, Automated Reasoning Group, Computer Science Department, UCLA, 2007. http://reasoning.cs.ucla.edu/rsat/.

[23] PolyBoRi – Polynomials over Boolean Rings, 2007.
http://polybori.sourceforge.net/.

[24] *Python Programming Language*, 2008. http://www.python.org.

[25] Håvard Raddum. Cryptanalytic Results on Trivium. *eStream*, 2006. http://www.ecrypt.eu.org/stream/papersdir/2006/039.ps.

[26] *SAGE Mathematical Software, Version 3.0.1*, 2008. http://www.sagemath.org.

[27] Christopher Wolf. *Multivariate Quadratic Polynomials in Public Key Cryptography.* PhD thesis, Katholieke Universiteit Leuven – Faculteit Ingenieurswetenschappen, 2005. *Cryptology ePrint Archive, Report 2005/393*, http://eprint.iacr.org/2005/393.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Weiterhin habe ich wörtlich oder inhaltlich übernommene Stellen als solche kenntlich gemacht und die Satzung der Universität Ulm zur Sicherung guter wissenschaftlicher Praxis in der Fassung vom 27. September 2003 beachtet.

Ulm, den 27. August 2008 Enrico Pilz