Kryptographische Protokolle im GSM-Standard: Beschreibung und Kryptanalyse

Diplomarbeit

vorgelegt an der

Professur für Theoretische Informatik
Prof. Dr. M. Krause
Universität Mannheim

im

Juli 1999

von

Erik Zenner, BSc
aus Ludwigshafen am Rhein

Ehrenwörtliche Erklärung

Ich versichere, daß ich die beiliegende Diplomarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ich bin mir bewußt, daß eine falsche Erklärung rechtliche Folgen haben wird.

Mannheim, den 10. Juli 1999

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

In den letzten Jahren hat die Verbreitung von Mobiltelefonen massiv zugenommen. Vor wenigen Jahren noch erregte ein "Handy"-Benutzer Aufmerksamkeit, war ein solches Gerät doch Indiz dafür, daß sein Besitzer zu jenen gehörte, die sich diesen Luxus leisten konnten oder die ihrem Arbeitgeber derart wichtig waren, daß ihm die ständige Verfügbarkeit des Mitarbeiters die nicht unerheblichen Kosten eines "Firmenhandys" wert war.

Inzwischen haben Funktelefone, die erheblich kleiner, leichter und vor allem erschwinglicher geworden sind als ihre Vorläufer vor einigen Jahren, einen Siegeszug angetreten, den wohl selbst die optimistischsten Marketing-Strategen noch vor wenigen Jahren nicht für möglich gehalten hätten. Handys haben Verbreitung gefunden in allen sozialen Schichten und nahezu allen Altersgruppen.

Dementsprechend vielfältig ist die Art der übertragenen Daten. Bei der Mehrzahl handelt es sich um private Gespräche, die für Dritte meist uninteressant sind. Doch auch sensible Daten werden arglos dem Mobiltelefon anvertraut. So mußte der deutsche Hersteller von Windenergieanlagen Enercon Anfang des Jahres 1998 feststellen, daß vermeintlich geheimgehaltene Forschungsergebnisse von einem amerikanischen Konkurrenten zum Patent angemeldet worden waren (siehe z.B. [SH98]). Und der Inhalt der äußerst pikanten Telefonate zwischen dem englischen Thronfolger, Prince Charles, und seiner damaligen Geliebten, Camilla Parker-Bowles, zierten gar die Titelseite der *Yellow Press*.

Solche und andere Begebenheiten zeigen, daß die Benutzer mobiler Telefone häufig keinen Gedanken daran verschwenden, daß ein Handy nichts anderes ist als ein handlicher Radiosender und daß ein auf gleicher Frequenz operierender Radioempfänger alle gesendeten Signale mithören kann.

Ein weiteres Problem, das in Deutschland noch nicht mit der gleichen Häufigkeit auftritt wie beispielsweise in den Vereinigten Staaten, ist das illegale Telefonieren auf Kosten eines anderen. Auch durch deutsche Gazetten geistern gelegentlich Meldungen, nach denen Telefonkunden die Gebühren für Gespräche belastet wurden, die sie nach eigener Aussage nie geführt haben. Für die USA beziffert Briceno in [BGW98b] den jährlichen Schaden durch Telefonbetrug auf etwa 500 Millionen Dollar.

Natürlich ist ein Schutz vor derartigen Problemen, wenn er denn vom Kunden, vom Dienstanbieter und nicht zuletzt vom Staat gewünscht wird, mit Hilfe kryptographischer Methoden möglich. So verwendet auch der europaweit am weitesten verbreitete Standard für Mobiltelefonie, die GSM-Norm, digitale Authentifikations- und Verschlüsselungsmethoden, um die korrekte Zurechnung von Telefonkosten zum

Verursacher zu gewährleisten und das unbefugte Mithören von Telefongesprächen durch Dritte zu erschweren.

Dem Stolz der Entwickler in Reihen der GSM auf den bahnbrechenden Einsatz von Kryptographie auch im privaten Bereich stand jedoch seit jeher die Skepsis von Wissenschaftlern gegenüber, die die verwendeten kryptographischen Verfahren für zu schwach hielten. Eine eindeutige Antwort wurde vor allem dadurch erschwert, daß die verwendeten Verfahren von der GSM geheimgehalten wurden. Die Aussagen beider Seiten sind häufig eher pathetisch als informativ, und so bleibt selbst für jene Benutzer, die sich einer möglichen Gefahr des Belauschtwerdens bewußt sind, die Frage nach der technischen Machbarkeit unbeantwortet.

1.2 Zielsetzung und Gliederung der Arbeit

Ziel der vorliegenden Arbeit ist die wissenschaftliche Untersuchung der im GSM-Standard enthaltenen kryptographischen Protokolle. Dies umfaßt sowohl eine gründliche Beschreibung der Algorithmen als auch ihre kryptanalytische Untersuchung, d.h. die Beantwortung der Frage, ob und mit welchem Aufwand die Sicherheitsprotokolle des Standards gebrochen werden können.

Inhaltlich zerfällt die Arbeit daher grob in zwei Teile. In den Kapiteln 2 bis 4 wird der Leser zunächst in die Thematik eingeführt. **Kapitel 2** beschreibt die Geschichte des GSM-Standards, und zwar sowohl unter dem Blickwinkel der Entwickler als auch der Kryptanalytiker. Auch wird auch ein erster Überblick gegeben über die Rolle kryptographischer Protokolle im Rahmen der GSM-Norm. In **Kapitel 3** wird dann ein konkreter Algorithmus namens COMP128 beschrieben, der jahrelang von vielen Dienstanbietern im Mobilfunkbereich im Rahmen der Benutzerauthentifikation eingesetzt wurde. **Kapitel 4** beschreibt dagegen den Sprachverschlüsselungsalgorithmus A5/1 (im folgenden meist kurz als A5 bezeichnet), der unter Kryptographen schon seit Jahren diskutiert wurde und der erst im Mai 1999 von Briceno et al. in [BGW99] vollständig veröffentlicht wurde. Die Untersuchung des A5 macht den Schwerpunkt der vorliegenden Arbeit aus, weshalb ein Verständnis des Algorithmus unabdingbare Voraussetzung für das Verständnis der nachfolgenden Kapitel ist.

In den Kapiteln 5 bis 10 wenden wir uns dann der Kryptanalyse zu, also der Untersuchung der beschriebenen Algorithmen auf Sicherheitslücken. **Kapitel 5** führt dabei zunächst in die Problematik ein und erläutert die Schritte, die zur Kryptanalyse des Sprachverschlüsselungsalgorithmus A5 erforderlich sind. Wir werden zeigen, daß eine triviale Kryptanalyse bereits mit einem Aufwand von im Mittel 2^{53} Suchschritten möglich ist. Darüberhinaus wird gezeigt, daß eine noch effizientere Kryptanalyse in einem dreistufigen Inversionsverfahren möglich ist, das in den Kapiteln 7 bis 9 beschrieben wird.

Kapitel 6 legt zunächst einige stochastiche Grundlagen für die nachfolgenden Untersuchungen. In **Kapitel 7** wenden wir uns dann der ersten (und schwierigsten) Stufe des Inversionsangriffes zu. Wir werden dazu verschiedene Vorgehensweisen untersuchen und abschließend einen Angriff vorstellen, der zwischen $2^{41,49}$ und $2^{42,04}$ Suchschritten benötigt, um das benötigte Ergebnis zu liefern. Darüberhinaus werden wir einige ältere Ergebnisse von Anderson ([And94]) und Golić ([Gol97]) widerlegen.

In **Kapitel 8** wird die zweite Stufe der Kryptanalyse untersucht. Wir beschreiben dazu einen Inversionsalgorithmus, der mit maximal $2^{17,47}$ Suchschritten das erforderliche Resultat liefert. Der Grundidee des Angriffes stammt erneut von Golić, die Effizienz des Angriffes wurde jedoch um den Faktor 6 gesteigert und die mit dem Verfahren verbundene Komplexität erstmals exakt berechnet.

Kapitel 9 schließlich beschreibt den dritten und letzten Schritt des Inversionsangriffs auf die Chiffre A5 und zeigt, daß aus dem Ergebnis der ersten beiden Schritte durch einfaches Lösen eines linearen Gleichungssystems mit 64 Unbekann-

ten derjenige Schlüssel ermittelt werden kann, mit dem das untersuchte Telefonat verschlüsselt wurde.

In **Kapitel 10** wird ein Angriff auf den Hashing-Algorithmus COMP128 demonstriert, den Briceno et al. in [BGW98a] vorgestellt haben. Das beschriebene Verfahren ermöglicht es, mit lediglich etwa $2^{17,33}$ Suchschritten den geheimen Benutzerschlüssel eines Handys herauszufinden, mit dessen Hilfe ein Angreifer auf fremde Kosten telefonieren oder auch sämtliche Telefonate des betroffenen Handys abhören kann. Der Angriff setzt allerdings im Gegensatz zu dem in den vorangegangenen Kapiteln beschriebenenen Verfahren voraus, daß der Kryptanalytiker für den Zeitraum von einigen Stunden im physikalischen Besitz des zu untersuchenden Mobiltelefons ist.

1.3 Bemerkungen zu den verwendeten Quellen

Obschon es eine Vielzahl von Veröffentlichungen zum Thema GSM-Standard gibt, sind die in gedruckter Form vorliegenden Resultate häufig schon wieder veraltet. Eine aktuellere, wenn auch nicht immer verläßliche Informationsquelle ist dagegen das Internet, wo auf Webseiten und in Newsgroups aktuelle Entwicklungen diskutiert werden. Aus diesem Grunde finden sich unter den Literaturangaben auch einige Internet-Publikationen, deren URL sich leider im Laufe der Zeit ändern könnte. Die Adressen entsprechen dem Stand von Anfang Juli 1999.

Unter den Arbeiten, die in den letzten Jahren über die Sicherheit des GSM-Standards verfaßt wurden, besitzen die drei folgenden Abhandlungen besondere Bedeutung:

- In [BGW98a] stellen Briceno, Goldberg und Wagner den bis dato geheimgehaltenen Hashingalgorithmus COMP128 vor. Dieser Algorithmus war Teil des Authentifikationsprotokolls (genaueres siehe Abschnitt 2.2.2). Die Arbeit zeigt einen massiven Sicherheitsmangel des Algorithmus auf und erläutert, wie man daraus den streng geheimen Benutzerschlüssel ermitteln kann.

- In [BGW99] stellen die gleichen Autoren auch den ebenfalls geheimgehaltenen Sprachverschlüsselungsalgorithmus A5 vor, über dessen Aufbau bis zum Mai 1999 nur spekuliert worden war.

- Bereits 1997 hatte Golić in [Gol97] verschiedene Verfahren zur Kryptanalyse des A5 vorgestellt und evaluiert. Obwohl seine Arbeit zum Teil auf Annahmen beruhte, sind die grundlegenden Ideen noch immer gültig und bedurften nur weniger Modifikationen, um auf den tatsächlichen Algorithmus A5 angewendet zu werden.

Diese Veröffentlichungen besitzen auch in der vorliegenden Arbeit großes Gewicht, eine Kenntnis ihres Inhalts ist jedoch für das Verständnis der nachfolgenden Betrachtungen nicht erforderlich. Die von Briceno et al. rekonstruierten Algorithmen COMP128 und A5 sind allerdings ihrer Bedeutung für die vorliegende Arbeit entsprechend in den Anhängen A und B abgedruckt.

1.4 Voraussetzungen und Terminologie

Obwohl sich die vorliegende Arbeit ausschließlich mit kryptographischen Themen befaßt, sollte es möglich sein, sie weitestgehend ohne eigene kryptographische Vorkenntnisse zu verstehen. Vorausgesetzt werden lediglich Grundkenntnisse in Linearer Algebra, Wahrscheinlichkeitsrechnung und allgemeiner Informatik. Der Leser sollte

außerdem Grundkenntnisse in der Theorie der linear rückgekoppelten Schieberegister besitzen. Eine gute allgemeine Einführung hierzu liefert das Kapitel "Stromchiffren" in [FR88], eine mathematisch fundiertere Behandlung findet sich im Kapitel "Linear Recurring Sequences" in [LN94].

Den mit kryptographischer Literatur weniger vertrauten Leser wird vielleicht die häufige Verwendung der Ausdrücke **Angriff** und **Angreifer** stören. Dabei handelt es sich jedoch um durchaus gängige Begriffe. Ein Angriff ist der Versuch, die Informationssicherheit eines kryptographischen Protokolls zu umgehen. Ein Angreifer ist entsprechend eine Instanz, die einen solchen Angriff unternimmt (siehe hierzu z.B. [MOV96], S. 13).

Ein umstrittener Begriff ist der des **Pseudo-One-Time-Pad**. Es sei an dieser Stelle bereits betont, daß damit keinesfalls ein One-Time-Pad gemeint ist, dessen Schlüssel echt pseudozufällig im Sinne einer strengen Definition ist (für eine formale Definition von Pseudozufälligkeit siehe z.B. [Lub96], S. 50). Vielmehr bezeichnet der Begriff lediglich eine binäre additive Flußchiffre, deren Schlüssel nicht echt zufällig ist wie der eines echten One-Time-Pads. Für eine genauere Erklärung des Begriffes siehe Abschnitt 5.1.

Der Begriff der **Komplexität** wird im Rahmen dieser Arbeit nur informal verwendet. Er bezieht sich nicht auf konkrete Berechnungsmodelle (wie Turing-Maschinen, Random-Access-Maschinen oder Schaltkreisfamilien), sondern beschreibt lediglich, wie viele Wiederholungen einer elementaren, im jeweiligen Zusammenhang erläuterten Prozedur ein Algorithmus im Mittel umfaßt. Ein exakter Vergleich des Resourcenbedarfes verschiedener Algorithmen ist somit nicht möglich, wohl aber eine Aussage darüber, ob ein vorgestellter Algorithmus in realistischer Zeit berechenbar ist oder nicht.

Die folgenden gängigen **Abkürzungen** aus der Linearen Algebra bzw. Wahrscheinlichkeitstheorie werden in der Arbeit ohne weitere Erläuterung verwendet:

- Mit $\mathbf{N}$ wird die Menge der natürlichen Zahlen einschließlich der Null bezeichnet, mit $\mathbf{Z}$ die Menge der ganzen Zahlen. Mit $\mathbf{Z}_2$ sei der Restklassenring der ganzen Zahlen modulo 2 bezeichnet.

- Sei x ein Ereignis, dann bezeichnen wir mit $\mathrm{Prob}(x)$ die Wahrscheinlichkeit, mit der x eintritt. Ist weiterhin X eine Zufallsvariable, so bezeichnet $\mathrm{E}(X)$ den Erwartungswert, $\mathrm{var}(X)$ die Varianz und $\sigma(X)$ die Standardabweichung von X.

Kapitel 2

GSM-Standard und Kryptographie

GSM steht für "Global System for Mobile Communications" und stellt einen weit verbreiteten, internationalen Standard für Funktelefone zur Verfügung. Die Spezifikationen füllten laut [GSM98] im Jahre 1997 bereits 130 Bände mit rund 6.000 Seiten und sind größtenteils als "commercial and confidential" eingestuft, d.h.: ihr Inhalt ist Betriebsgeheimnis der beteiligten Netzbetreiber und somit der Öffentlichkeit nicht frei zugänglich.

Nichtsdestotrotz sind gerade diejenigen Teile der Spezifikationen, die sich mit der Sicherheit des Systems beschäftigen, im Laufe der letzten Jahre über verschiedene Kanäle ins Internet gelangt. Das folgende Kapitel soll zunächst einen Überblick geben über die Rolle der Kryptographie im Rahmen des GSM-Standards, bevor wir uns in den späteren Kapiteln den genauen Spezifikationen zuwenden wie auch der Frage, ob die von Herstellerseite gepriesene Sicherheit des Systems einer wissenschaftlichen Prüfung standhalten kann.

2.1 Geschichtlicher Überblick

2.1.1 Entwicklung des GSM-Standards

Die Entwicklung des GSM-Standards begann im Jahre 1982, als die CEPT (Conférence des Administrations Européennes des Postes et Télécommunications - der Zusammenschluß der Telekommunikationsanbieter Europas) die Groupe Spéciale Mobile, kurz GSM, ins Leben rief. Aufgabe der GSM sollte es sein, einen europaweiten Standard für Mobiltelefonie auszuarbeiten und durchzusetzen - ein für die damalige Zeit ungewöhnliches Unterfangen. Man muß sich vor Augen halten, daß in den frühen 80er Jahren noch nahezu alle Telekommunikationsanbieter staatliche Monopolisten waren, daß noch Schlagbäume und Zollkontrollen die Grenzen Europas markierten, daß jedes Land eigene Normen für die Mobiltelefonie besaß und daß von einem "Handy" noch niemand etwas gehört hatte. Mobiltelefone waren damals noch massige Geräte mit langen Funkantennen, die fest in die Fahrzeuge von Managern, Freiberuflern oder Rettungsdiensten eingebaut waren und die vom Rest der Bevölkerung spöttisch belächelt wurden.

So dauerte es denn auch nahezu zehn Jahre, bis im Juli 1991 erste Teile des GSM-Netzes im Rahmen eines Testbetriebes in verschiedenen europäischen Ländern in Betrieb gingen. Im Jahre 1993 wurden wichtige europäische Städte mit GSM-Netzen versorgt, und erst 1995 wurden diese Netze miteinander verbunden. Zur gleichen Zeit - im November 1995 - wurde auch der erste kommerzielle GSM-Dienst in den

Vereinigten Staaten von Amerika in Dienst gestellt. Heute (Stand Juni 1999) nutzen 170 Millionen Telefon-Kunden weltweit den GSM-Standard, wobei die meisten von ihnen das Wort "GSM" noch nie gehört haben dürften.

2.1.2 Veröffentlichungen zur Kryptanalyse

> *One of the most attractive features of GSM is that it is a very secure network. All communications, both speech and data, are encrypted to prevent eavesdropping. In fact, in the early stages of its development it was found that the encryption algorithm was too powerful for certain technology export regulators. This could have had serious implications for the global spread of GSM by limiting the number of countries to which it could be sold. Fortunately, the MoU*[1] *reacted promptly to this threat. Alternative algorithms were developed that enabled the free dissemination of the technology worldwide.*
>
> (entnommen den WWW-Seiten der GSM, siehe [GSM98])

Diejenigen Gruppen, die an der Entwicklung des GSM-Standards beteiligt waren, werden nicht müde, darauf hinzuweisen, wie bahnbrechend der Einsatz kryptographischer Techniken in der Mobiltelefonie ist (siehe z.B. [Ved98]). Kryptanalytiker dagegen waren von Anfang an eher skeptisch; man vermutete, daß die Sicherheit des GSM-Standards primär darauf beruhte, daß die Algorithmen unter Verschluß gehalten wurden ("security by obscurity"). Ein solches Vorgehen stellt einen eklatanten Verstoß gegen das Kerckhoff'sche Prinzip dar, nach dem die Sicherheit kryptographischer Systeme einzig von der Geheimhaltung des Schlüssels, nicht jedoch von der Geheimhaltung des Algorithmus abhängen darf. Gerüchte, daß die Regierungen insbesondere Frankreichs und Großbritanniens während der Entwicklungsphase "regulierend" eingegriffen hatten, um ihr Recht auf Überwachung auch mobiler Telefone zu wahren, taten ein übriges, um die kryptographische Gemeinde an der tatsächlichen Sicherheit des GSM-Standards zweifeln zu lassen.

Der Stein kam ins Rollen, als der Bradford University in Großbritannien Teile der Entwurfsunterlagen des Verschlüsselungsalgorithmus A5 in Form eines anonymen Briefes zugingen (eine Kopie kann unter [GSM88] im Internet eingesehen werden). A5 war ein bis dato geheimgehaltener Teil des GSM-Standards und diente der Sprachverschlüsselung. Es handelte sich dabei um eine taktgesteuerte Flußchiffre mit Schlüssellänge 64 Bit. Zwar waren die Unterlagen unvollständig, aber sie reichten aus, um erste Aussagen über die Sicherheit zu treffen.

Diese frühen Aussagen waren von sehr unterschiedlicher Qualität. Anderson [And94] schlug in einem Newsgroupbeitrag einen Angriff vor, der ein wichtiges Merkmal des A5 außer Acht ließ. Der Angriff dagegen, den Shepherd im Juni 1994 auf einer IEE-Konferenz vortragen wollte, beunruhigte das britische GCHQ[2] derart, daß der Vortrag untersagt und die zugehörige Veröffentlichung [Shep94] indiziert wurde. Bis heute ist über diesen Angriff nichts bekannt außer einigen Andeutungen, die Shepherd im Vorfeld gegenüber Bekannten gemacht hatte.

Im Mai 1997 stellte Golić auf der EUROCRYPT-Konferenz in Konstanz [Gol97] einen Angriff auf den A5 vor, der eine Komplexität von im Mittel $2^{40,16}$ Iterationen aufweisen sollte. Dieser Angriff wird bis heute von Kryptographen, die sich mit dem A5 befassen, als Referenz benutzt, zumal er allgemein genug gehalten war, um auch auf den vollständigen Algorithmus anwendbar zu sein, wenn die fehlenden Details bekannt würden.

[1] MoU = Mobile Unit

[2] GCHQ = Government Communications Headquarter, der britische Informationsgeheimdienst

Im April 1998 veröffentlichten Briceno, Goldberg und Wagner in [BGW98a] den Algorithmus COMP128, den sie durch das Auswerten durchgesickerter Entwurfsunterlagen und durch Reverse Engineering herausgefunden hatten. Bei diesem Algorithmus handelte es sich um einen Vorschlag der GSM-Kommission zur Umsetzung der Benutzerauthentifikation. Zusammen mit dem Code lieferten die Autoren einen Angriff, der aus etwa 150.000 Authentifikations-Anfragen den geheimen Masterschlüssel des Benutzers ermittelt. Der Angriff wurde im gleichen Jahr vom Chaos Computer Club Deutschland wiederholt und verifiziert (siehe hierzu [CCC98]). Die Pressestelle der GSM sprach von "keiner ernstzunehmenden Gefahr", dennoch beeilte sich die Gesellschaft, an ihre Mitglieder einen modifizierten Algorithmus namens COMP128-2 herauszugeben, über dessen Sicherheit zur Zeit noch nichts bekannt ist.

Im Mai 1999 veröffentlichten ebenfalls Briceno, Goldberg und Wager in [BGW99] auch den vollständigen Algorithmus A5. Dieser unterschied sich in einigen Punkten von der bisher vermuteten Version und machte einen Teil des Angriffes nach Golić obsolet. Obwohl eine rege Diskussion in Newsgroups und Mailinglisten folgte, sind noch keine auf dieser Fassung des A5 basierenden effizienteren Angriffe aufgetaucht. Auch hat die Veröffentlichung des A5 (ganz im Gegensatz zum Angriff auf den COMP128) in der Presse bisher keinen Niederschlag gefunden. Nichtsdestotrotz arbeiten vermutlich zur Zeit zahlreiche Kryptographen unabhängig voneinander daran, dem A5 eine Schwäche nachzuweisen. Die Zeit wird zeigen, ob sie dabei Erfolg haben und mit einem Angriff aufwarten können, der die Ergebnisse von Golić noch verbessert.

2.2 Kryptographischer Überblick

Der folgende Abschnitt soll zunächst einen Überblick geben darüber, zu welchen Zwecken Kryptographie im Rahmen des GSM-Standards genutzt wird. Dabei kann man sich hier noch auf "bekannte", d.h. von der GSM selbst bzw. von an der Entwicklung beteiligten Wissenschaftlern veröffentlichte Quellen wie z.B. [RE96] und [Ved98] stützen.

2.2.1 Einführung

In der Terminologie des GSM-Standards wird ein Funktelefon als *Mobile Station*, kurz MS, bezeichnet. Eine solche Mobilstation besteht aus zwei separaten Komponenten: Dem *Mobile Equipment* (ME), das im wesentlichen den Sende- und Empfangsteil umfaßt, und dem *Subscriber Identity Module* (SIM), einer manipulationsgeschützten Smartcard, die alle benutzer- bzw. anbieterspezifischen Informationen enthält.

Auf der SIM-Karte sind insbesondere die sicherheitsrelevanten Daten gespeichert, so auch der geheime, 128 Bit lange Benutzerschlüssel Ki, eine weltweit eindeutige, 64 Bit lange Identifikationsnummer namens IMSI (international mobile subscriber identity) und eine einmalige, ebenfalls 64 Bit lange Identifikationsnummer namens TMSI (temporary mobile subscriber identity). IMSI bzw. TMSI werden von der Mobilstation an den Betreiber übertragen und erlauben es diesem, den Benutzerschlüssel Ki abzuleiten, ohne daß dieser übertragen werden mußte. Dabei gibt es zwei mögliche Verfahren, die je nach Präferenzen des Netzbetreibers verwendet werden können:

- Der Benutzerschlüssel Ki wird aus der IMSI und einem streng geheimen Masterschlüssel Mk ermittelt (z.B. mit Hilfe des DES, siehe hierzu auch [Ved98]).
- Der Benutzerschlüssel wird anhand der IMSI aus einer Datenbank ausgelesen.

Kryptographie kommt im Verlaufe eines Funktelefonats vor allem in zwei Bereichen zum Einsatz: Bei der Authentifikation des Benutzers gegenüber dem Netzbetreiber und bei der Verschlüsselung der übertragenen Sprachdaten. Abbildung 2.1 auf Seite 9 zeigt den prinzipiellen Verlauf beider Vorgänge aus kryptographischer Sicht.

2.2.2 Authentifikation

Sinn der Authentifikation ist es, den Benutzer gegenüber dem Dienstanbieter eindeutig zu identifizieren. Schließlich müssen die Telefongebühren korrekt zugerechnet werden. Gäbe es keine kryptographische Authentifikation, so könnte ein betrügerischer Telefonnutzer einfach eine beliebige IMSI an den Betreiber schicken und darauf hoffen, daß es einen Nutzer mit dieser Identifikationsnummer gibt. Wenn ja, würden diesem die Telefongebühren belastet. Um einen solchen Betrug zu verhindern, wird zu Beginn eines Telefonates eine Authentifikation durchgeführt, bei der geprüft wird, ob dem Benutzer (bzw. seinem SIM) sowohl die IMSI als auch der Benutzerschlüssel Ki bekannt sind.

Soll also eine Verbindung zwischen Mobil- und Basisstation hergestellt werden, so wird zunächst die IMSI/TMSI an die Basisstation gefunkt. Diese ermittelt aus dem Hintergrundsystem den zugehörigen Benutzerschlüssel Ki. Außerdem sendet sie eine 128-Bit-Zufallszahl RAND an die Mobilstation. Aus dieser Zufallszahl RAND ermitteln nun sowohl die SIM-Karte des Funktelefons als auch das Hintergrundsystem der Basisstation mit Hilfe des kryptographischen Algorithmus A3 und des (beiden bekannten) Benutzerschlüssels Ki eine 4 Byte lange *signed response* SRES bzw. SRES'. Stimmen diese beiden Werte überein, so gilt die Mobilstation bei der Basisstation als autorisiert, andernfalls wird die Verbindung abgebrochen.

A3 ist dabei zunächst nichts anderes als die Signatur eines nicht näher definierten deterministischen Algorithmus, der als Eingabe Ki und RAND erwartet und als Ausgabe eine Signed Response SRES liefert. Es steht jedem Netzbetreiber frei, einen eigenen Algorithmus zu diesem Zweck zu entwickeln. Die GSM hat ihren Mitgliedern ein "Beispiel" für einen solchen Algorithmus geliefert, und zwar den bereits in Abschnitt 2.1 erwähnten COMP128. Nach Aussage der SmartCard Developer Association haben jedoch fast alle Netzanbieter diesen Algorithmus übernommen, eine erfreuliche Ausnahme bilden lediglich die meisten der deutschen Provider. Nichtsdestotrotz ist der COMP128 verbreitet genug, um eine genauere Untersuchung zu rechtfertigen. Aus diesem Grunde wird er in Kapitel 3 im Detail vorgestellt.

2.2.3 Kommunikation

Die zweite wichtige kryptographische Anwendung besteht natürlich in der Verschlüsselung der Sprachdaten auf der Luftschnittstelle, d.h. jenen Daten, die über Radiowellen übertragen werden und die ein Dritter mit Hilfe eines geeigneten Empfangsgerätes problemlos mithören könnte.

Zur Verschlüsselung der Kommunikation wird nicht der geheime Benutzerschlüssel Ki selbst eingesetzt, sondern ein sogenannter Sitzungsschlüssel, der mit Kc bezeichnet wird. Dieser wird zu Beginn eines Telefonates mit Hilfe des Algorithmus A8 aus der Zufallszahl RAND und dem Benutzerschlüssel Ki generiert und behält längstenfalls für die Dauer dieses Gespräches Gültigkeit. A8 ist dabei erneut nur die Signatur eines entsprechend zu wählenden Algorithmus, und in der Praxis handelt es sich dabei ebenfalls häufig um den erwähnten COMP128.

Im GSM-Standard werden Sprachdaten in einzelnen Rahmen fester Größe übertragen. Dies geschieht im Voll-Duplex-Modus, d.h.: jedem gesendeten Datenpaket steht ein empfangenes Datenpaket gegenüber. Jedes solche Datenpaket enthält neben den üblichen administrativen Einträgen zwei Elemente, die im folgenden von

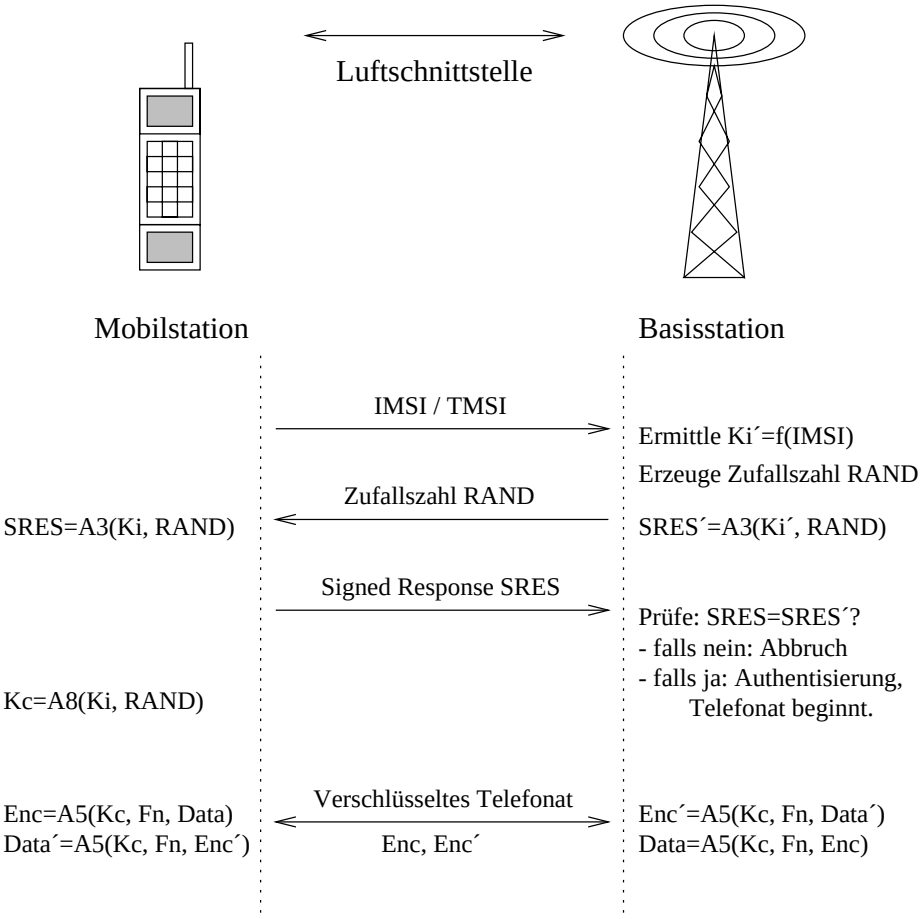

Abbildung 2.1: Authentifikation und Kommunikation unter GSM

besonderer Bedeutung sind:

- eine Rahmennummer Nf der Länge 22 Bit sowie
- digitalisierte und verschlüsselte Sprachdaten der Länge 114 Bit.

Für jeden solchen Rahmen existiert nun ein eindeutiger Rahmenschlüssel, der sich durch Kombination des (geheimen) Sitzungsschlüssels Kc mit der (öffentlichen) Rahmennummer Nf ergibt und der sowohl von der Mobil- als auch von der Basisstation ermittelt werden kann. Dieser wird von der Chiffre A5 verwendet, um die eigentliche Ver- und Entschlüsselung der digitalisierten Sprachdaten zu ermöglichen.

Im Gegensatz zu den Algorithmen A3 und A8 ist das A5-Protokoll im GSM-Standard eindeutig festgelegt. Es wird aus diesem Grunde im Kapitel 4 im Detail beschrieben.

Kapitel 3

Beschreibung des Algorithmus COMP128

Bereits in Kapitel 2 wurde darauf hingewiesen, daß im GSM-Standard eigentlich von den kryptographischen Algorithmen A3 und A8 die Rede ist, die nicht näher spezifiziert werden. Der COMP128, den das nachfolgende Kapitel genauer beschreibt, war von der GSM-Kommission lediglich als Anregung zur Entwicklung eigener Algorithmen gemeint, wurde jedoch (vorhersehbarerweise) von vielen Dienstanbietern der Einfachheit halber unverändert übernommen. Nachdem er 1998 von Briceno et al. gebrochen wurde, wurde er in den technischen Unterlagen der GSM eiligst durch den Nachfolger COMP128-2 ersetzt, in vielen (insbesondere älteren) Mobilstationen versieht jedoch noch immer der COMP128 seinen Dienst. Aufgrund der weiten Verbreitung soll dieser Algorithmus hier im Detail vorgestellt werden. Der von Briceno, Goldberg und Wagner veröffentlichte Code, der der folgenden Beschreibung zugrunde liegt, ist im Anhang A abgedruckt.

3.1 Konzeption und Rundenstruktur

COMP128 ist ein Hashing-Algorithmus, der als Eingabe die 16-Byte-Zufallszahl RAND und den 16-Byte-Benutzerschlüssel Ki erhält und als Ausgabe einen 16 Byte langen Hashwert produziert. Das Grundprinzip des Algorithmus wird von Abbildung 3.1 veranschaulicht.

Der Algorithmus wird initialisiert, indem RAND und Ki zu einem 32-Byte-Wert x konkateniert werden. Dabei füllt Ki die 16 niederstwertigen und RAND die 16 höchstwertigen Bytes von x aus.

Kern des Algorithmus ist eine Hash-Funktion, die diesen 32-Byte-Wert x in einen 16-Byte-Wert x' überführt. Diese Funktion ist surjektiv, jedoch offensichtlich nicht injektiv, da auf jeden möglichen Ausgabewert x' im Mittel immerhin 2^{16} Eingabewerte x kommen, die ihn erzeugt haben können.

Die Hash-Funktion wird insgesamt neunmal durchlaufen. Zwischen zwei Durchläufen werden die Bits von x' permutiert; das Ergebnis x" wird anschließend durch Konkatenation mit Ki erneut auf die Länge 32 Byte gebracht.

3.1.1 Die Hash-Funktion

In der obigen Terminologie erhält die Hash-Funktion als Eingabe einen 32-Byte-Wert x. Um die Arbeitsweise der Funktion besser zu verstehen, sprechen wir im folgenden allerdings nicht mehr von "Bytes", sondern sagen stattdessen: Die Hash-Funktion erhält als Eingabe ein Feld aus 32 Werten, wobei jeder Wert die Länge 8

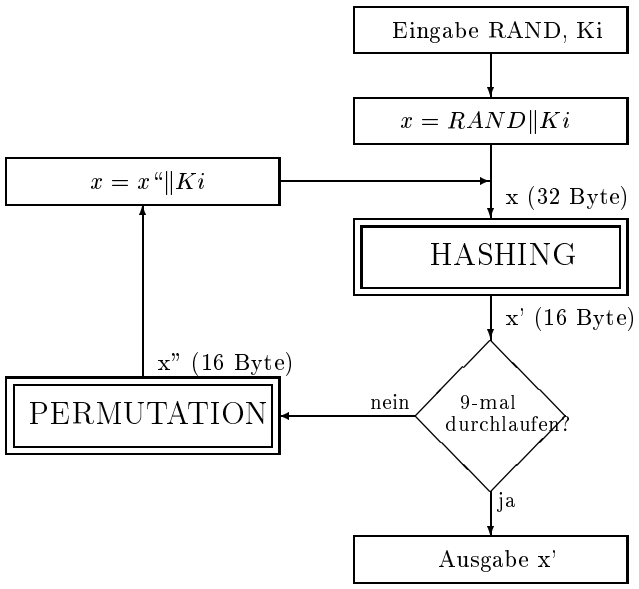

Abbildung 3.1: Schematischer Aufbau des Algorithmus COMP128

Bit besitzt. Im folgenden bezeichnen wir dieses Eingabefeld mit x_0. Der i-te Wert im Feld wird mit $x_0[i]$ bezeichnet.

Die Hash-Funktion umfaßt fünf "Runden". In jeder solchen Runde werden die 32 Werte zu 16 Paaren geordnet, und zwar wie in Tabelle 3.1 angegeben. So beeinflussen sich beispielsweise in Runde 1 die Werte mit den Indizes 0 und 16, 1 und 17 usw.

Jede der fünf Runden entspricht nun einer Abbildung H_r, die ein solches Wertepaar als Eingabe erhält und ein anderes Wertepaar aus Ausgabe erzeugt:

$$H_r : (x_{r-1}[i], x_{r-1}[j]) \rightarrow (x_r[i], x_r[j])$$

Dabei erzeugt Runde 1 aus zwei 8-Bit-Werten zwei neue 8-Bit-Werte, alle übrigen Runden jedoch reduzieren die Größe der Eingabewerte um eins (d.h.: Runde 2 bildet 7-Bit-Werte, Runde 3 bildet 6-Bit-Werte usw.). Zu diesem Zweck werden Substitutionstabellen, sogenannte S-Boxen, verwendet. Jede Runde besitzt ihre eigene Substitutionstabelle. Die Ersetzung erfolgt dabei wie folgt:

Runde	Indizes der Wertepaare															
#1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
#2	0	1	2	3	4	5	6	7	16	17	18	19	20	21	22	23
	8	9	10	11	12	13	14	15	24	25	26	27	28	29	30	31
#3	0	1	2	3	8	9	10	11	16	17	18	19	24	25	26	27
	4	5	6	7	12	13	14	15	20	21	22	23	28	29	30	31
#4	0	1	4	5	8	9	12	13	16	17	20	21	24	25	28	29
	2	3	6	7	10	11	14	15	18	19	22	23	26	27	30	31
#5	0	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31

Tabelle 3.1: Bildung von Wertepaaren in den Substitutionsrunden

- Bilde zwei Hilfswerte y, z wie folgt:

$$\begin{aligned} y &= (x_{r-1}[i] + 2 \cdot x_{r-1}[j]) \bmod 2^{10-r} \\ z &= (2 \cdot x_{r-1}[i] + x_{r-1}[j]) \bmod 2^{10-r} \end{aligned}$$

- Bilde neue Werte wie folgt:

$$\begin{aligned} x_r[i] &= \text{S-Box}_r[y] \\ x_r[j] &= \text{S-Box}_r[z] \end{aligned}$$

Abbildung 3.2 auf Seite 13 gibt einen Überblick über die Rundenstruktur der Hash-Funktion. Die S-Boxen sind dem Anhang A zu entnehmen.

3.1.2 Die Permutation

Die 16 Byte lange Ausgabe x' der Hash-Funktion ist zugleich die Eingabe der Permutationsfunktion. Diese faßt den Wert als Bitstring der Länge 128 auf. Sie erzeugt daraus durch Permutation der einzelnen Bits den Ausgabewert x", der die gleichen Bits enthält, diese jedoch in veränderter Reihenfolge.

Zunächst werden die Bits des Eingabestrings umgeordnet. An die Position k des neu entstehenden Strings wird dasjenige Bit geschrieben, das an Position $(k \cdot 17) \bmod 128$ des Eingabestrings stand. Da 17 ein primitves Erzeugendes von $\mathbf{Z}_{128}$ ist, wird auf diese Weise für $k = 0, \ldots, 127$ tatsächlich jedes Bit genau einmal verwendet.

Aus dem umgeordneten String werden nun die Bits des Ausgabewertes x" erzeugt. Dabei entsprechen die Bits $(i, i+1, \ldots, i+7)$ des permutierten Strings dem i-ten Byte des Ausgabewertes x", allerdings werden sie in umgekehrter Reihenfolge eingelesen.

3.2 Ermittlung von SRES und Kc

Wie oben beschrieben, erzeugt der COMP128 einen 16 Byte (bzw. 128 Bit) langen Ausgabestring. Einige dieser Bytes werden nun ausgewählt, um die 4 Byte lange *signed response* SRES und den 8 Byte langen Sitzungsschlüssel Kc zu erhalten. Auf diese Weise erfüllt der COMP128 sowohl die Funktion des A3 also auch des A8.

Es ist nicht bekannt, ob aus den technischen Unterlagen der GSM tatsächlich hervorgeht, welche Bytes zu diesem Zweck ausgewählt werden. Die Autoren von [BGW98a] beziehen sich bei den von ihnen gemachten Angaben auf die SIM-Karte eines konkreten Providers, da sie die entsprechenden Informationen durch Reverse Engineering herausgefunden haben. Demnach werden die Bytes wie folgt verwendet:

- SRES besteht aus den Bits 0 bis 31 des Ausgabestrings x'.
- Kc besteht aus den Bits 74 bis 127, gefolgt von 10 Nullen.

Die letztere Information ist natürlich sensationell und verdient besondere Beachtung. Wenn die letzten zehn Bits von Kc automatisch auf 0 gesetzt werden, bedeutet dies, daß die Entropie des Sitzungsschlüssels von 64 auf 54 Bit reduziert wurde. Somit läge eine vorsätzliche Schwächung des Sitzungsschlüssels um den Faktor 2^{10} vor. In der Tat behauptet Briceno in [BGW98b], daß diese vorsätzliche Schwächung selbst bei denjenigen Dienstanbietern erfolgt, die anstelle des COMP128 einen eigenen Algorithmus verwenden. Es ist somit zumindest nicht auszuschließen, daß diese Schwächung integraler Bestandteil der A8-Spezifikation ist.

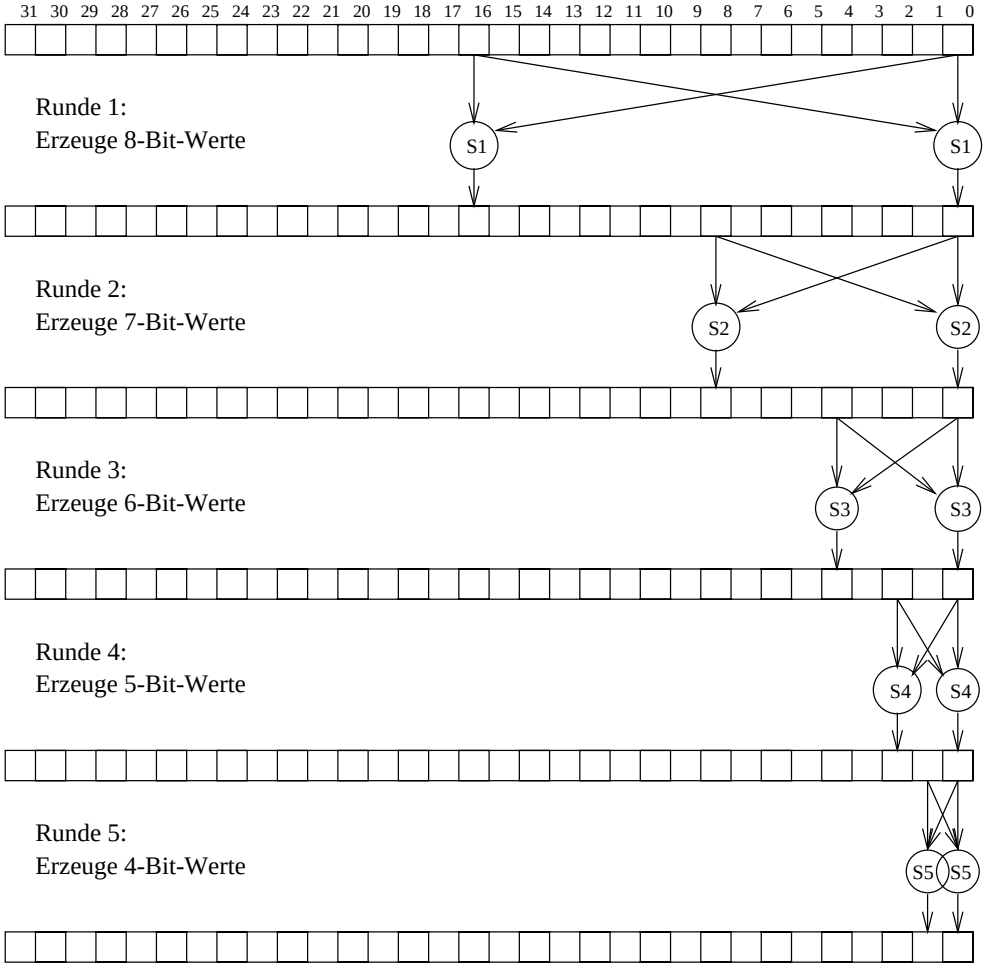

Abbildung 3.2: “Schmetterlinge” und S-Boxen in einer Hashing-Runde

Kapitel 4

Beschreibung des Algorithmus A5

Wie bereits in Kapitel 2 erwähnt, wird der Algorithmus A5 schon seit Ende der achtziger Jahre in der Fachliteratur und im Internet diskutiert, und zwar stets auf der Grundlage des Teiles der Entwurfsunterlagen, der zum jeweiligen Zeitpunkt bereits bekannt war. Seit Mai 1999 nun liegt eine Implementierung des A5 in der Programmiersprache C vor, die von Briceno, Goldberg und Wagner erstellt wurde (siehe [BGW99]). Im Gegensatz zu der bis dahin kursierenden Implementierung nach Roe (die z.B. in [Shn96] abgedruckt ist) enthält dieser C-Code keine erkennbaren Fehler und ist zudem nach Aussage der Autoren gegen Testvektoren getestet. Die folgende Beschreibung geht daher davon aus, daß es den Autoren tatsächlich gelungen ist, den "echten" A5 zu finden. Da der Quellcode bisher nur im Internet vorliegt, wurde er in Anhang B abgedruckt.

4.1 Konzeption und Terminologie

Die Chiffre A5 zählt zur Gruppe der synchronen, nichtlinearen Flußchiffren. Ihre wesentlichen Grundelemente sind der Schlüsselstromgenerator und die Ver-/Entschlüsselungseinheit. Abbildung 4.1 zeigt den grundlegenden Aufbau der Chiffre. Die im folgenden eingeführte Terminologie orientiert sich im wesentlichen an derjenigen in [Rue92].

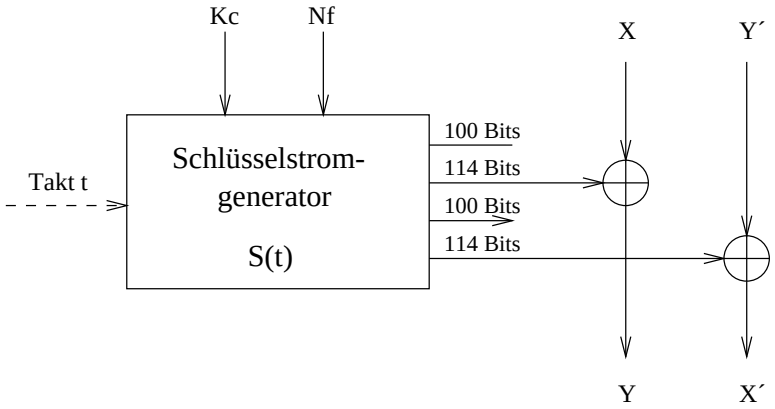

Abbildung 4.1: Schematischer Aufbau der Chiffre A5

- Sei $S(t)$ der innere Zustand des Schlüsselstromgenerators zum Zeitpunkt t, d.h. der Zustand nach dem t-ten Mastertakt. Mit $\mathcal{S}$ sei die Menge der möglichen inneren Zustände bezeichnet. Da der Schlüsselstromgenerator aus 64 Speicherzellen zu je 1 Bit besteht, ist $\mathcal{S} = \{0,1\}^{64}$.

- Der Anfangszustand $S(0)$ des Schlüsselstromgenerators wird durch das Tupel (Kc, Nf) bestimmt, wobei Kc $\in \{0,1\}^{64}$ als "geheimer" und Nf $\in \{0,1\}^{22}$ als "öffentlicher" Schlüssel bezeichnet werden kann. Beide zusammen bilden den Schlüssel K. Für den Schlüsselraum $\mathcal{K}$ gilt also: $|\mathcal{K}| = 2^{64+22} = 2^{86}$.

- Aus jedem Anfangszustand $S(0)$ erzeugt der Generator auf eindeutige Weise einen Schlüsselstrom Z der Länge 428 Bit. Es kann folglich nicht mehr verschiedene Schlüsselströme als Anfangszustände geben. Für das Schlüsselstromalphabet $\mathcal{Z}$ muß demnach gelten: $|\mathcal{Z}| \leq |\mathcal{S}| = 2^{64} - 1 \ll 2^{428}$.

  Für Elemente $Z \in \mathcal{Z}$ bezeichnet z_t das t-te Bit der Binärdarstellung, d.h. dasjenige Bit, das zum Zeitpunkt t vom Schlüsselstromgenerator erzeugt wird, $Z = (z_t)_{t=1}^{428}$.

- Sei schließlich $\mathcal{X}$ die Menge der möglichen Klartexte X und $\mathcal{Y}$ die Menge der möglichen Chiffretexte Y, $\mathcal{X} = \mathcal{Y} = \{0,1\}^{114}$. Dabei bezeichne x_i bzw. y_i das i-te Bit der Binärdarstellung von X bzw. Y, d.h. $X = (x_i)_{i=1}^{114}$ und $Y = (y_i)_{i=1}^{114}$.

Die eigentliche Verschlüsselung erfolgt durch ein sogenanntes Pseudo-One-Time-Pad, d.h. ein Klartext X wird chiffriert durch bitweise binäre Addition des vom Generator erzeugten Schlüsselstroms. Analog wird der sich ergebende Geheimtext Y dechiffriert, indem der gleiche Schlüsselstrom erneut binär bitweise addiert wird.

Eine Besonderheit der A5-Chiffre besteht darin, daß nicht alle 428 Bits, die der Schlüsselstromgenerator pro Übertragungsrahmen erzeugt, verwendet werden. Die Verwendung erfolgt vielmehr nach dem folgenden Schema:

Bit-Nr.	*Verwendung*
1-100	verfallen ungenutzt
101-214	Verschlüsselung der Senderichtung durch bitweise Addition zu den digitalisierten Sprachdaten
215-314	verfallen ungenutzt
315-428	Entschlüsselung der Empfangsrichtung durch bitweise Addition zu den verschlüsselten Sprachdaten.

Diese Aufstellung erfolgt aus Sicht des Funktelefons, die Basisstation nutzt analog die Bits 101-214 zur Entschlüsselung und die Bits 315-428 zur Verschlüsselung. Ebenfalls aus der Sicht des Funktelefons gilt somit für die Verschlüsselung der 114 Bits des Klartextes

$$y_i = x_i \oplus z_{i+100}$$

und für die Entschlüsselung der 114 Bits des Chiffretextes

$$x_i = y_i \oplus z_{i+314}$$

4.2 Aufbau des Schlüsselstromgenerators

Der Schlüsselstromgenerator erzeugt einen pseudozufälligen Schlüsselstrom, wobei er die Nichtlinearität der erzeugenden Funktion durch eine Taktsteuerung erzwingt. Er besteht im wesentlichen aus den folgenden Bausteinen:

- Drei linear rückgekoppelten Schieberegistern (engl.: linear feedback shift register, im folgenden gelegentlich mit LFSR abgekürzt),
- einer Taktkontrolle, die die Taktung dieser Schieberegister bestimmt, sowie
- einem XOR-Gatter, das die Ausgabeströme der drei Schieberegister zum Ausgabestrom der gesamten Flußchiffre verknüpft.

Abbildung 4.2 zeigt den Aufbau des Schlüsselstromgenerators.

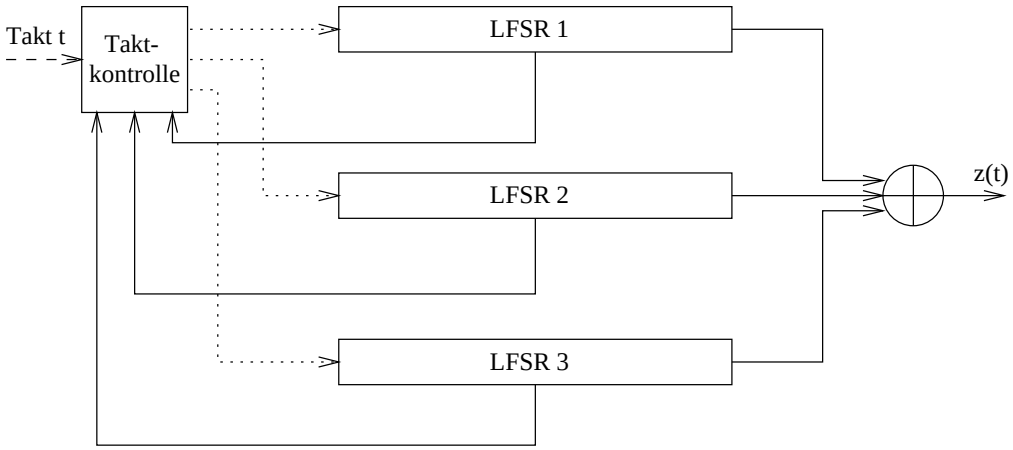

Abbildung 4.2: Aufbau des Schlüsselstromgenerators

4.2.1 Die Schieberegister

Primärer Grundbaustein des Generators sind drei linear rückgekoppelte Schieberegister $\mathrm{LFSR}_r, r = 1, 2, 3$. Die Längen der einzelnen Register seien im folgenden bezeichnet mit L_r, sie betragen $L_1 = 19$, $L_2 = 22$ und $L_3 = 23$.

Der gesamte innere Zustand $S(t)$ des Generators zu einem Zeitpunkt t setzt sich zusammen aus den inneren Zuständen $S_r(t)$ der drei Schieberegister zum Zeitpunkt t:

$$S(t) = (S_1(t), S_2(t), S_3(t))$$

Bezeichne weiterhin $s_{r,i}(t)$ den inneren Zustand des Registers r an der Position i zum Zeitpunkt t, so gilt:

$$S_r(t) = (s_{r,i}(t))_{i=1}^{L_r}$$

Jedes dieser LFSR_r wird mathematisch beschrieben durch eine lineare Rekurrenzgleichung der folgenden Form[1]:

$$s_i(t) = \begin{cases} a_1 s_1(t-1) + a_2 s_2(t-1) + \ldots + a_L s_L(t-1) & \text{falls} \quad i = L_r \\ s_{i+1}(t-1) & \text{sonst} \end{cases}$$

wobei $a_i = 1$ das Vorhandensein eines Rückkopplungsausganges an der Position i bezeichnet und $a_i = 0$ das Fehlen eines solchen Ausganges.

Abbildung 4.3 zeigt den Aufbau eines solchen Schieberegisters am Beispiel von LFSR_1.

[1] Der Index r wurde hier der besseren Lesbarkeit halber ausgelassen.

Alternativ kann man die linear rückgekoppelten Schieberegister durch das charakteristische Polynom beschreiben, das der obigen linearen Abbildung zugeordnet ist. Dieses besitzt die folgende Gestalt:

$$F(x) = x^L + \sum_{i=1}^{L} a_i x^{i-1}$$

Bis vor kurzem war von den drei charakteristischen Polynomen nur $F_1(x)$ definitiv bekannt, für $F_2(x)$ und $F_3(x)$ hatte man schwach besetzte, primitive Polynome angenommen (so z.B. in der Implementierung nach Roe, siehe hierzu [Shn96]). In [BGW99] werden nun jedoch charakteristische Polynome angegeben, die vermutlich korrekt sind (andernfalls wäre ein Test der Chiffre wegen fehlerhafter Schlüsselstromerzeugung gescheitert). Diese Polynome sind definiert wie folgt:

$$F_1(x) = x^{19} + x^5 + x^2 + x + 1 \quad (4.1)$$

$$F_2(x) = x^{22} + x + 1 \quad (4.2)$$

$$F_2(x) = x^{23} + x^{15} + x^2 + x + 1 \quad (4.3)$$

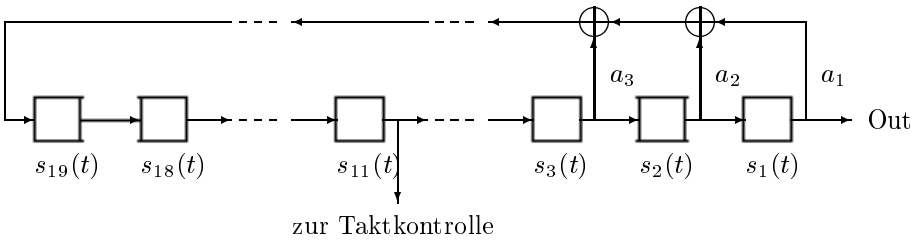

Abbildung 4.3: Schematischer Aufbau von Schieberegister LFSR_1

4.2.2 Die Taktkontrolle

Verantwortlich für die Nicht-Linearität des vom Generator erzeugten Schlüsselstromes ist die Taktkontrolle. Würde man nämlich stets alle drei Schieberegister gleichzeitig weitertakten und ihre Ausgabeströme am Ende mit Hilfe des XOR-Gatters zu einem Schlüsselstrom verknüpfen, so wäre dieser Schlüsselstrom linear abhängig vom Anfangszustand $S(0)$ des Generators. In diesem Fall könnte der Anfangszustand $S(0)$ durch Lösen eines linearen Gleichungssystems effizient aus dem Schlüsselstrom rekonstruiert werden.

Um diese Linearität zu vermeiden, wurde eine Taktkontrollfunktion eingebaut. Diese erhält als Eingabe zum Zeitpunkt t von jedem der drei Schieberegister das "mittlere" Zustandsbit s_{r,τ_r}. Im folgenden bezeichnen wir τ_r als Taktausgang des Registers r. In [BGW99] werden für die Taktausgänge die konkreten Werte $\tau_1 = 11$, $\tau_2 = 12$ sowie $\tau_3 = 13$ angegeben.

Als Ausgabe legt die Taktkontrollfunktion für jedes LFSR_r fest, ob es mit dem nächsten Takt t weitergetaktet werden soll oder nicht. Dabei kann die Entscheidungsregel umgangssprachlich wie folgt formuliert werden:

- Falls 0 oder 1 Taktkontrollbits auf '1' stehen, so takte alle Register, deren Taktkontrollbits auf '0' stehen.
- Falls 2 oder 3 Taktkontrollbits auf '1' stehen, so takte alle Register, deren Taktkontrollbits auf '1' stehen.

Mathematisch formaler kann die Taktkontrollfunktion $C(t)$ wie folgt definiert werden:

$$C(t) = (c_1(t), c_2(t), c_3(t))$$

mit

$$c_r(t) = \begin{cases} 0 & falls \quad s_{r,\tau_r}(t-1) \neq s_{j,\tau_j}(t-1) = s_{k,\tau_k}(t-1) \\ 1 & sonst \end{cases}$$

und $r \neq j \neq k$.

Wichtig ist vor allem die Feststellung, daß auf diese Weise in jedem Fall mindestens zwei Schieberegister getaktet werden.

4.2.3 Der Ausgabeschlüsselstrom

Aufgabe des Generators ist es, einen (möglichst pseudozufälligen) Ausgabeschlüsselstrom Z zu erzeugen. Nach jedem Takt t greift der Generator an jedem Register r das erste Bit $s_{r,1}$ ab und liefert es an das XOR-Gatter, wo durch additive Verknüpfung der drei Bits ein neues Schlüsselbit z_t erzeugt wird. Es gilt also:

$$Z = (z_t)_{t=1}^{428}$$

mit

$$z_t = s_{1,1}(t) \oplus s_{2,1}(t) \oplus s_{3,1}(t)$$

4.3 Initialisierung des Schlüsselstromgenerators

Bevor die Schlüsselstromerzeugung durch den Generator gestartet werden kann, muß zunächst der Anfangszustand $S(0)$ geladen werden. Dabei wird wie folgt vorgegangen:

- Alle drei Register werden zunächst mit dem Nullvektor initialisiert. Das heißt: $s_{r,i} = 0$ für alle $r = 1,2,3$ und $i = 1,\ldots,L_r$. Das Ergebnis ist der innere Zustand $S(-86)$.

- Dann wird der geheime Sitzungsschlüssel Kc der Länge 64 Bit in die Register eingespeist. Dazu wird wird jedes Register 64 mal gleichförmig (d.h. ohne die in Abschnitt 4.2.2 beschriebene Taktkontrolle) getaktet. Mit jedem Takt $t = 1,\ldots,64$ wird das t-te Bit des Sitzungsschlüssels in den Feedback-Pfad aller drei Register addiert. Das Ergebnis ist der innere Zustand $S(-22)$.

  Das Überraschendste an dieser Initialisierung ist, daß in diesem Fall kein Zusammenhang bestünde zwischen der Schlüssellänge von 64 Bit und der Gesamtlänge der drei Schieberegister, die ebenfalls 64 Bit beträgt. Bis zur Veröffentlichung des A5-Codes durch Briceno et al. ging man davon aus, daß jedes der Sitzungsschlüsselbits genau einem Bit des inneren Zustandes $S(-22)$ entspräche.

- Auf die gleiche Weise wird nun auch die Rahmennummer eingespeist. Jedes Register wird 22 mal gleichförmig getaktet, und mit jedem Takt $t = 1,\ldots,22$ wird das t-te Bit der Rahmennummer in den Feedback-Pfad aller drei Register addiert. Das Ergebnis ist der innere Zustand $S(0)$.

  Auch hier tritt ein kryptographisch entscheidender Unterschied zum bisher vermuteten Aufbau des A5 auf. Man war nämlich zuvor davon ausgegangen, daß die Rahmennummern unter Verwendung der Taktkontrolle (also nichtlinear) eingespeist würden, was die Kryptanalyse erheblich verkompliziert hätte.

4.4 Bemerkungen zur Implementierung nach Roe

Wie bereits erwähnt, wurde vor der Veröffentlichung von Briceno et al. eine ebenfalls in C geschriebene Implementierung von Mike Roe von der Cambridge University als Referenz für den A5 verwendet. Roe gibt in seinem Code selbst an, daß er einige Annahmen treffen mußte. Allerdings enthält seine Version des A5 auch zwei von den Annahmen unabhängige, offensichtliche Fehler, so daß es eigentlich erstaunlich ist, daß der Quellcode von Autoren wie Schneier (in [Shn96]) und Wobst (in [Wob98]) unverändert übernommen wurde.

Zum einen enthält das Programm gleich vier fehlerhafte Funktionsaufrufe, die vermutlich durch einen Tippfehler und anschließendes Cut-and-Paste entstanden sind. Man beachte dazu die Zeile:

```
clock_ctl = treshold(r1, r2, r2)
```

die natürlich korrekt lauten müßte:

```
clock_ctl = treshold(r1, r2, r3)
```

Somit arbeitet die Funktion `treshold` (sie ermittelt, ob die Register mit Inhalt 0 oder diejenigen mit Inhalt 1 getaktet werden sollen) mit dem Inhalt von Register 1 und zweimal Register 2, was zu regelmäßigen Fehlern in der Taktkontrolle führt.

Ebenfalls offensichtlich falsch sind die von Roe angenommenen Feedback-Polynome $F_2(x)$ und $F_3(x)$. Im einleitenden Kommentar schreibt der Autor nämlich völlig korrekt: *"we do not know the feedback taps of registers 2 and 3, but we do know from the chip's gate count that they have at most 6 feedback taps between them."* Als Feedback-Polynome nimmt Roe dann aber

$$\begin{aligned} F_2(x) &= x^{22} + x^9 + x^5 + x + 1 \qquad \text{und} \\ F_3(x) &= x^{23} + x^5 + x^4 + x + 1 \end{aligned}$$

an, die jeweils 4, insgesamt also 8 Feedback Taps aufweisen.

Kapitel 5

Vorgehensweise bei der Kryptanalyse

Die vorliegende Arbeit befaßt sich schwerpunktmäßig mit der Fragestellung, ob und mit welchem Rechenaufwand der Sitzungsschlüssel eines mit dem Algorithmus A5 verschlüsselten Telefonates ermittelt werden kann. Gelingt dies, so kann der Angreifer das gesamte Telefonat entschlüsseln unter der Voraussetzung, daß nicht zwischenzeitlich der Schlüssel gewechselt wurde. Das folgende Kapitel soll zunächst eine grobe Übersicht über denkbare Strategien zur Kryptanalyse geben, bevor wir uns in den Kapiteln 7 bis 9 den Details der Kryptanalyse zuwenden.

5.1 Vorbemerkung

Der Algorithmus A5 ist eine Flußchiffre nach dem Prinzip des sogenannten Pseudo-One-Time-Pads - ein etwas umstrittener Begriff, der hier kurz erläutert werden soll.

Unter einem echten One-Time-Pad (nach seinem Erfinder[1] auch als Vernam-Chiffre bezeichnet) versteht man einen Verschlüsselungsalgorithmus, der jedes Klartextbit mit genau einem echt zufällig ermittelten Schlüsselbit codiert wie folgt:

$$y_i = x_i \oplus z_i$$

Der Schlüssel Z muß dazu mindestens die gleiche Länge besitzen wie der Klartext X. Wird jeder Schlüssel nur genau einmal verwendet, so besitzt das echte One-Time-Pad die Besonderheit, daß es beweisbar sicher ist gegen Ciphertext-Only-Angriffe und daß selbst Known-Plaintext-Angriffe dem Angreifer nicht mehr Informationen liefern, als er vorher schon besaß.

Das echte One-Time-Pad ist für eine Anwendung wie z.B. ein Mobiltelefonat nicht praktikabel, da die Gesprächsteilnehmer nicht vor Gesprächsbeginn einen geheimen, echt zufälligen Schlüssel austauschen können, der lang genug ist, um das gesamte Telefonat ohne Wiederholungen verschlüsseln zu können. Aus diesem Grunde behilft man sich mit dem Pseudo-One-Time-Pad, bei dem ein Schlüsselstromgenerator aus einem kurzen geheimen Schlüssel eine lange Folge sogenannter Pseudozufallsbits erzeugt, die für den Betrachter zwar kaum von echten Zufallsbits zu unterscheiden sind, die aber nichtsdestotrotz das Ergebnis eines effizient berechenbaren, deterministischen Prozesses sind. Im Gegensatz zum echten One-Time-Pad ist das Pseudo-One-Time-Pad theoretisch unsicher sowohl gegen Ciphertext-Only- als auch gegen Known-Plaintext-Angriffe. Daraus folgt jedoch nicht automatisch,

[1] Gilbert S. Vernam, 1890 bis 1960

daß eine Kryptanalyse praktisch möglich ist. Praktisch unsicher ist ein Pseudo-One-Time-Pad nur dann, wenn es einen effizienten Algorithmus zur Kryptanalyse gibt. Die folgenden Untersuchungen haben zum Ziel, solche effiziente Algorithmen zu finden.

5.2 Der Ciphertext-Only-Angriff

Ein Ciphertext-Only-Angriff gegen den A5 setzt voraus, daß der Angreifer eine Möglichkeit besitzt, einen korrekten Klartext zu erkennen, wenn er ihn gefunden hat. Dazu gibt es mehrere Möglichkeiten:

- Bei dem vom A5 verschlüsselten Klartext handelt es sich um komprimierte Sprachdaten. Ist dem Angreifer sowohl das Digitalisierungsverfahren für die (ursprünglich analoge) Sprache als auch der Kompressionsalgorithmus bekannt, so kann er (zumindest mit einer gewissen Wahrscheinlichkeit) entscheiden, ob es sich bei einem rekonstruierten Klartext um Sprachdaten oder um zufällige Daten ("Rauschen") handelt.
- Wenn das Kompressionsverfahren mit einer Prüfsumme arbeitet, so kann diese ebenfalls dazu genutzt werden, die Korrektheit einer Entschlüsselung zu verifizieren.

Im Falle des A5 sollte es bei Kenntnis der Spezifikation möglich sein, einen gefundenen Klartext-Kandidaten mit einer Wahrscheinlichkeit größer Null von zufälligem Rauschen zu unterscheiden.

Der trivialste Ciphertext-Only-Angriff besteht dann darin, den vollständigen Schlüsselraum zu durchsuchen. Wie in Abschnitt 3.2 bereits erwähnt wurde, wird die theoretisch mögliche Anzahl von 2^{64} Schlüsseln anscheinend nicht ausgenutzt. Vielmehr verwenden zumindest die bekannten Implementierungen nur 2^{54} verschiedene Sitzungsschlüssel Kc. Ein Brute-Force-Angriff auf den A5 erfordert also im Durchschnitt das Austesten von lediglich 2^{53} Sitzungsschlüsseln. An dieser Schranke werden sich alle übrigen Angriffe messen lassen müssen, die wir im folgenden betrachten.

5.3 Der Known-Plaintext-Angriff

> *During about the first 1/10th of a call the vocoder will encode silence. A very rough estimate is 13000 bps * 0.1 s = 1300 bits of known plaintext. Clearly, the cryptanalyst has a lot to work with here.*
>
> (Marc Briceno, in [Bri99])

5.3.1 Das Problem der Datenbeschaffung

Ein Known-Plaintext-Angriff setzt voraus, daß der Angreifer über eine Anzahl korrespondierender Klartext-/Chiffretextpaare verfügt. Im Falle des A5 sind mehrere Methoden denkbar, solche Paare zu erhalten:

- Golić denkt in [Gol97] zunächst daran, einen Teil des unverschlüsselten Gesprächs über die Basisstation oder das Festnetz abzuhören. Obwohl dieses Szenario natürlich nicht undenkbar ist, stellt sich dabei die Frage, warum ein Angreifer, der diese Möglichkeit besitzt, überhaupt auf die verschlüsselte Luftschnittstelle zurückzugreifen sollte.

- Der Gedanke, im Chiffrat nach besonders häufigen Wörtern oder Gesprächspausen zu suchen, ist eher unpraktikabel. Zum einen weiß der Kryptanalytiker nicht genau, ob und an welcher Stelle des Chiffretextes der von ihm geratene Klartext beginnt. Zum anderen werden analoge Sprachdaten aufgrund wechselnder Stimmhöhe, Betonung, Hintergrundgeräuschen etc. nur äußerst selten zweimal in den gleichen Bitstring umgesetzt.

- Eine elegantere Möglichkeit besteht dagegen darin, den Abgehörten dazu zu bringen, einen bekannten und in digitaler Form vorliegenden Text verschlüsselt zu übertragen. Dies könnte z.B. gelingen, wenn dieser einen digitalen Anrufbeantworter oder eine T-Box anruft bzw. verwendet. Es ist für den Angreifer möglich, sich diesen Klartext durch einen eigenen Anruf zu besorgen. Da einmal digitalisierte Daten immer wieder auf gleiche Weise wiedergegeben werden, besitzt der Angreifer nun den benötigten Klartext. Er muß allerdings noch herausfinden, an welcher Stelle des Chiffrats der ihm bekannte Text beginnt, was ebenfalls keine leichte Aufgabe ist.

- Eine weitere Möglichkeit ist ein sogenannter *Replay-Angriff*. Dabei zeichnet der Angreifer zunächst das gesamte chiffrierte Telefonat auf. Danach ruft er den Belauschten selbst an, wobei er sich zugleich als Basisstation ausgibt und dem Handy die gleiche RAND zusendet, die für das aufgezeichnete Gespräch verwendet wurde. Somit wird das Gespräch, das der Angreifer mit dem Belauschten führt, mit dem gleichen Sitzungsschlüssel Kc codiert wie das Gespräch, das der Angreifer eigentlich entschlüsseln will. Der Angreifer kann nun die Klartext-Chiffretext-Paare aus dem eigenen Gespräch benutzen, um den Schlüssel Kc zu rekonstruieren und anschließend das belauschte Telefonat zu dechiffrieren.

- Die wohl einfachste Möglichkeit zur Beschaffung von Klartext-/Chiffretextpaaren bietet der von Briceno erwähnte "lautlose" Gesprächsauftakt. Hier kennt der Kryptanalytiker tatsächlich Klartext und korrespondierenden Chiffretext, und zwar sogar in ausreichendem Umfang.

Das Problem der Datenbeschaffung soll im folgenden nicht weiter vertieft werden. Es ist realistisch anzunehmen, daß einem entschlossenen Angreifer die für einen Known-Plaintext-Angriff benötigte Menge an Klartext-/Chiffretext-Paaren zur Verfügung steht. Anders ausgedrückt: Wir nehmen im folgenden an, daß der Kryptanalytiker für einen oder mehrere Übertragungsrahmen die Ausgabe des Schlüsselstromgenerators kennt.

5.3.2 Die Phasen eines Angriffes

Es gibt im Grunde genommen zwei Arten von Angriffen auf den A5: Den Vorwärtsangriff und den Inversionsangriff. Vorwärtsangriffe treffen dabei bestimmte Annahmen über den Schlüssel Kc, durchlaufen den Algorithmus A5 mit seinen Phasen *Initialisierung, Erzeugung von Verwirrungsbits* und *Erzeugung von Verschlüsselungsbits* und prüfen schließlich nach, ob das Ergebnis mit dem bekannten Output des Schlüsselstromgenerators konsistent ist. Die vollständige Suche, die oben als Beispiel für einen Ciphertext-Only-Angriff genannt wurde, ist ein Spezialfall eines solchen Vorwärtsangriffs.

Inversionsangriffe gehen den umgekehrten Weg: Sie beginnen mit dem Output des Schlüsselstromgenerators, arbeiten sich rückwärts durch die Phasen des A5 und liefern als Ausgabe Informationen über den Sitzungsschlüssel Kc. Da die folgenden Kapitel sich mit einem solchen Inversionsangriff beschäftigen, sollen hier die drei Phasen eines solchen Vorgehens detailliert dargelegt werden:

- Die erste Phase (Kapitel 7) des Angriffs geht aus von einer dem Kryptanalytiker bekannten Schlüsselstromsequenz. Ziel dieser ersten Phase ist es, einen oder mehrere innere Zustände $S(t)$ des Schlüsselstromgenerators zu ermitteln, aus denen diese Schlüsselstromsequenz erzeugt worden sein könnte.
- Die zweite Phase (Kapitel 8) des Angriffs verwendet die so ermittelten Zwischenzustände $S(t)$ und ermittelt aus diesen diejenigen Anfangszustände $S(0)$, aus denen sie hervorgegangen sein könnten.
- Die dritte Phase (Kapitel 9) des Angriffs schließlich geht von den so ermittelten Anfangszuständen $S(0)$ aus, um die bekannte Rahmennummer Nf zu eliminieren und schließlich eine Reihe von Kandidaten für den Sitzungsschlüssel Kc zu erhalten.

Abbildung 5.1 veranschaulicht noch einmal die Vorgehensweise bei der Erzeugung des Schlüsselstroms und die dazu inverse Vorgehensweise bei der Kryptanalyse.

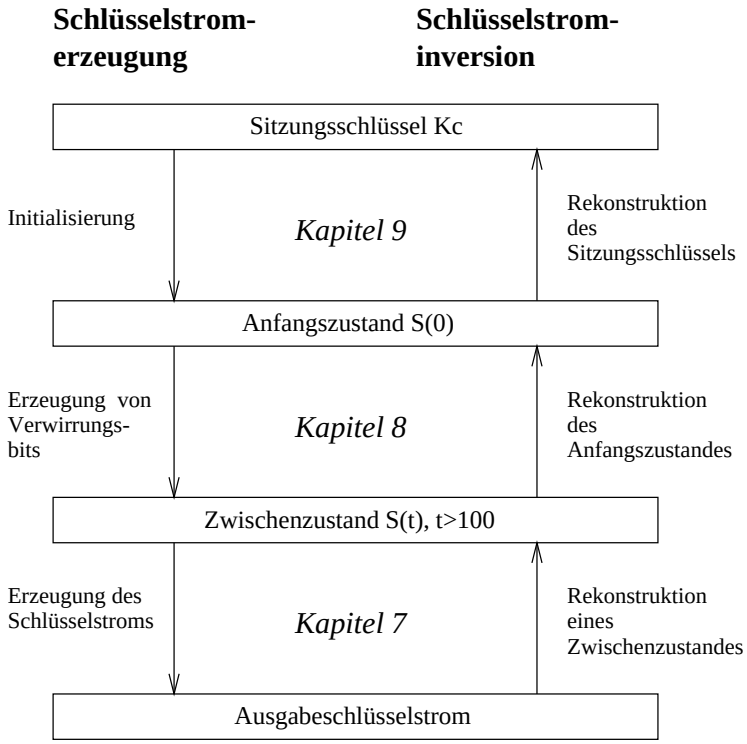

Abbildung 5.1: Schlüsselstromerzeugung und Inversionsangriff

Für das Durchlaufen dieser drei Phasen genügt die Schlüsselstromsequenz eines einzelnen Übertragungsrahmens. Hat man auf diese Weise mehrere Kandidaten für den Sitzungsschlüssel erhalten, so muß man diesen mit Hilfe weiterer Übertragungsrahmen (deren Schlüsselstromsequenz man im Normalfall nicht mehr kennen muß) verfizieren.

Kapitel 6

Statistische Vorbemerkungen

Im folgenden sollen einige triviale statistische Besonderheiten der Chiffre A5 beschrieben werden, die für die Kryptanalyse in den nachfolgenden Kapiteln von Bedeutung sind.

6.1 Das Verhalten der Schieberegister

Die kryptographische Qualität eines linear rückgekoppelten Schieberegisters ist abhängig von der Periode der erzeugten Sequenz. Diese Periode ist maximal (nämlich gleich $2^n - 1$ für ein Schieberegister der Länge n), wenn das dem Schieberegister zugeordnete charakteristische Polynom primitiv ist.

In der Tat ist diese Eigenschaft für die drei vom Schlüsselstromgenerator verwendeten Polynome erfüllt. Dies kann man beispielsweise nachprüfen, indem man die in Abschnitt 4.2.1 gegebenen Polynome in einer entsprechenden Tabelle nachschlägt (siehe z.B. [Cha98]).

Eine wichtige Eigenschaft primitiver Polynome, die in den nachfolgenden Kapiteln mehrfach verwendet wird, ist die folgende:

Lemma 6.1 *Für primitive charakteristische Polynome ist der niederstwertige Koeffizient $a_{r,1}$ immer gleich 1.*

Bew.:

Angenommen der Koeffizient $a_{r,1}$ sei gleich 0 in einem linear rückgekoppelten Schieberegister der Länge n. Dann würde das niederstwertigste Bit des Registers die Rückkopplung nicht beeinflussen. Dieses letzte Register würde somit stets nur das Outputbit eines kürzeren, aus den höherwertigen $n-1$ Zellen bestehenden Schieberegisters enthalten. Die Periode der erzeugten Bitfolge betrüge somit maximal 2^{n-1} mit einer Vorperiode der Länge 1 Bit. Dies steht im Widerspruch dazu, daß Schieberegister mit primitiven charakteristischen Polynomen stets maximale Folgen mit Periode $2^n - 1$ erzeugen. ■

Es ist bekannt, daß Folgen, die von linear rückgekoppelten Schieberegistern mit primitiven charakteristischen Polynomen erzeugt wurden, die Golomb'schen Kriterien für pseudozufällige Folgen erfüllen (siehe z.B. [FR88]). Insbesondere treten alle Teilfolgen der Länge m gleich häufig auf (mit Ausnahme derjenigen Teilfolgen, die nur aus Nullen bestehen). Wenn wir also im folgenden bei der Betrachtung des inneren Zustand eines Schieberegisters keine weiteren Informationen über dessen Inhalt besitzen, so ist es gerechtfertigt, die Bits eines Registers als Zufallsvariablen zu betrachten, die mit Gleichverteilung aus $\{0, 1\}$ gezogen wurden.

Es sei an dieser Stelle noch angemerkt, daß aufgrund des in Abschnitt 4.3 beschriebenen Initialisierungsalgorithmus die Möglichkeit besteht, daß ein Register

mit dem Nullvektor initialisiert wird. In diesem besonderen Fall würde ein Register für die Dauer eines Rahmens nur Nullen erzeugen. Die Wahrscheinlichkeit, daß dies für wenigstens eines der drei Register der Fall ist, liegt allerdings nur bei $2^{-18,75}$. Ein kryptographischer Angriff, der auf dieser Schwäche aufbaut, würde somit nur bei sehr langen Telefonaten Sinn machen (bei 13.000 bps bzw. 114 Rahmen pro Sekunde würde ein solcher Fall im statistsichen Mittel einmal in 64,45 Minuten auftreten).

6.2 Das Verhalten der Taktkontrolle

Bei der Analyse der Taktkontrolle ist es hilfreich, zunächst eine Wertetabelle der Taktkontrollfunktion zu betrachten. In Tabelle 6.1 bezeichnen die s_{r,τ_r} die Belegungen der Taktkontrollbits und c_r die daraus resultierenden Taktungen für das Register r ($c_r = 1$ bedeutet, daß das Register mit dem nächsten Mastertakt weitergetaktet wird).

s_{1,τ_1}	s_{2,τ_2}	s_{3,τ_3}	c_1	c_2	c_3	$\sum_{i=1}^{3} c_i$
0	0	0	1	1	1	3
0	0	1	1	1	0	2
0	1	0	1	0	1	2
0	1	1	0	1	1	2
1	0	0	0	1	1	2
1	0	1	1	0	1	2
1	1	0	1	1	0	2
1	1	1	1	1	1	3

Tabelle 6.1: Verhalten der Taktkontrolle

Für den Kryptanalysten ist der Inhalt der Schieberegister zunächst unbekannt. Man betrachtet die $s_{r,i}$ daher als Zufallsvariablen. Unter der vereinfachenden Annahme, daß die $s_{r,i}$ voneinander unabhängig sind und mit Gleichverteilung aus $\{0,1\}$ gezogen wurden, lassen sich einige Aussagen über die statistischen Eigenheiten der Taktkontrolle treffen. So sind die beiden folgenden Korollare sofort ersichtlich:

Lemma 6.2 *Die Wahrscheinlichkeit, daß ein fest gewähltes Register r mit einem gegebenen Mastertakt getaktet wird, beträgt $\frac{3}{4}$.*

Bew.:

Aus der Annahme der Gleichverteilung für die Belegungen der s_{r,τ_r} folgt, daß die 8 Elementarereignisse aus Tabelle 6.1 gleich wahrscheinlich sind und somit jeweils die Wahrscheinlichkeit $\frac{1}{8}$ besitzen.
Wie aus Tabelle 6.1 ersichtlich, wird jedes Register in 6 der 8 Fälle weitergetaktet. Somit beträgt die Wahrscheinlichkeit, daß ein bestimmtes Register mit einem Mastertakt t getaktet wird, genau $\frac{1}{8} \cdot 6 = \frac{3}{4}$. ■

Lemma 6.3 *Der Erwartungswert für die Summe der Registertakte $\sum_{i=1}^{3} c_i$ mit einem gegebenen Mastertakt beträgt $\frac{9}{4}$.*

Bew.:

Gemäß der Annahme der Gleichverteilung tritt jedes Elementarereignis mit Wahrscheinlichkeit $\frac{1}{8}$ auf. Der Erwartungswert für die Summe der Taktungen c_r beträgt dann:

$$\frac{3+2+2+\ldots+2+3}{8} = \frac{18}{8} = \frac{9}{4}$$

■

6.3 Die Anzahl möglicher Vorgängerzustände

In [Gol97] demonstriert Golić[1] die folgende Aussage:

Satz 6.4 *Der Schlüsselstromgenerator kann für $t \geq 1$ maximal $2^{63,32}$ verschiedene innere Zustände $S(t)$ annehmen.*

Bew.:

Wir nehmen zunächst an, daß alle 2^{64} inneren Zustände $S(t)$ angenommen werden können und interessieren uns für die Frage, aus welchen zugehörigen Vorgängerzuständen $S(t-1)$ diese inneren Zustände hervorgegangen sein könnten.
Wir betrachten daher für alle drei Register die Zellen $s_{r,\tau}$ und $s_{r,\tau-1}$ (siehe Abbildung 6.1), denn in drei dieser sechs Register müssen zwangsläufig diejenigen drei Bits stehen, die die letzte Taktkontrolle bestimmt und somit zum aktuellen Zustand geführt haben.

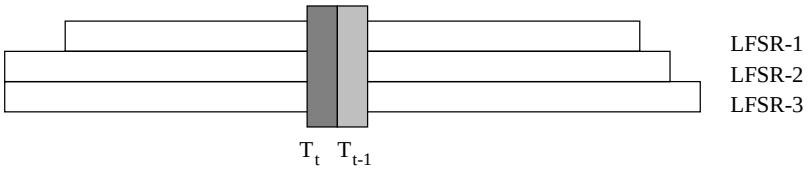

Abbildung 6.1: Betrachtung von 6 Registerzellen

Wenn man sich nun fragt, wie der mögliche Vorgängerzustand zu einem gegebenen inneren Zustand aussieht, so stellt man fest, daß es keinen Vorgängerzustand gibt, der die folgende Belegung der 6 betrachteten Registerzellen erzeugt:

$$s_{i,\tau-1} = s_{j,\tau-1} = s_{k,\tau} \neq s_{k,\tau-1}$$

Abbildung 6.2 veranschaulicht diesen Fall, wobei eine Belegung mit '?' bedeutet, daß der Zustand nicht erzeugt werden kann, gleichgültig, ob eine '0' oder eine '1' anstelle des '?' auftritt.

Abbildung 6.2: Unmögliche innere Zustände

Von den $2^6 = 64$ möglichen Kombinationen für die Belegung der betrachteten 6 Registerzellen besitzen immerhin $4 \cdot 3 \cdot 2 = 24$ das obige Profil (Es gibt 4 Möglichkeiten, die Zellen $s_{i,\tau}$ und $s_{j,\tau}$ zu belegen und 3 Möglichkeiten, dem Index k einen der drei Werte 1,2 oder 3 zuzuweisen, und das für jede der 2 in Abbildung 6.2 dargestellten Kombinationen).
Somit sind $\frac{24}{64} = \frac{3}{8}$ der denkbaren inneren Zustände $S(t), t \geq 1$, gar nicht erzeugbar. Es verbleiben also lediglich $\frac{5}{8} \cdot 2^{64} \approx 2^{63,32}$ mögliche innere Zustände $S(t)$ für $t \geq 1$. ■

[1] Die Aussage ist Teil einer umfangreichen Untersuchung über die Anzahl möglicher Vorgängerzustände zu einem gegebenen inneren Zustand $S(t), t \geq 1$. Die übrigen Ergebnisse sind jedoch für die konkrete Implementierung des A5 nicht mehr von Bedeutung.

Es sei an dieser Stelle noch angemerkt, daß aus dieser Aussage keinesfalls der Umkehrschluß gezogen werden darf, daß aus n inneren Zuständen zum Zeitpunkt $S(t-1)$ lediglich $\frac{5}{8} \cdot n$ Nachfolgezustände $S(t)$ erzeugt werden können. Eine solche Aussage ist schon deshalb unsinnig, weil dann nach 95 Takten die Zahl der möglichen inneren Zustände $S(95)$ gleich $2^{64} \cdot \left(\frac{5}{8}\right)^{95} \approx 0,75$ und somit echt kleiner 1 wäre.

Wir werden in späteren Kapiteln stets davon ausgehen, daß es für jeden Zeitpunkt $t \geq 1$ höchstens $2^{63,32}$ mögliche innere Zustände $S(t)$ gibt und daß die unmöglichen Zustände durch einfache lineare Gleichungen beschrieben werden können.

Kapitel 7

Rekonstruktion eines Zwischenzustandes

In den folgenden Kapiteln sollen nun verschiedene Techniken zur Kryptanalyse der Chiffre A5 diskutiert werden. Wir beginnen dazu mit dem ersten der in Abschnitt 5.3.2 beschriebenen Schritte: Aus einer Folge von Schlüsselstrombits soll ein innerer Zustand $S(t), t \geq 0$, ermittelt werden, der diese Bitfolge erzeugt haben könnte.

7.1 Vorbemerkungen

7.1.1 Verifikation eines gefundenen Zwischenzustandes

Da es möglich ist, verschiedene innere Zustände $S(t)$ zu finden, die alle das gleiche Outputbit z_t und den gleichen Folgezustand $S(t+1)$ erzeugen, ist die Schlüsselstromfunktion nicht injektiv. Das bedeutet insbesondere, daß es zu einem gegebenen Schlüsselstrom mehrere innere Zustände $S(t)$ geben kann, aus denen er hervorgegangen sein könnte. Obwohl ihre Zahl nicht besonders groß sein sollte, ist sie doch im Mittel größer als Eins.

Da es Ziel der Kryptanalyse ist, den korrekten Sitzungsschlüssel Kc zu finden, genügt es nicht, nur einen beliebigen inneren Zustand zu finden, der mit dem beobachteten Schlüsselstrom konsistent ist. Der Kryptanalytiker muß vielmehr genau den gleichen inneren Zustand finden, der auch bei der Verschlüsselung verwendet wurde.

Die in den nachfolgenden Abschnitten aufgeführten Angriffsmethoden können meist auf zwei Arten eingesetzt werden:

1. Man kann den Suchalgorithmus so lange laufen lassen, bis ein konsistenter innerer Zustand $S(t)$ gefunden ist. Ein solcher Zustand muß aber nicht notwendigerweise identisch sein mit dem inneren Zustand, aus dem der Ausgabeschlüsselstrom erzeugt wurde. Daher muß der Kryptanalytiker für diesen inneren Zustand die in Kapitel 8 und 9 beschriebenen Schritte auszuführen und den resultierenden Sitzungsschlüssel Kc' gegen weitere Rahmen zu verifizieren.

2. Man kann aber auch gleich den gesamten Suchraum durchlaufen und alle konsistenten inneren Zustände abspeichern. Dann erst nimmt man die Verifikation der Zustände vor.

Lösung 1 ist zu empfehlen, wenn genügend Speicherplatz vorhanden ist, um den Suchalgorithmus für die Dauer der Verifikation "einzufrieren" (d.h. den aktuellen

Zustand des Kryptanalysealgorithmus so abzuspeichern, daß bei einem negativen Ergebnis der Verifikation an der Stelle fortgefahren werden kann, wo man aufgehört hat). Lösung 2 dagegen verbraucht im Mittel doppelt so viel Rechenzeit, wie notwendig wäre, um die korrekte Lösung zu finden, und ist nur dann sinnvoll, wenn dem Angreifer mehr Zeit als Speicherplatz zur Verfügung steht.

Zum Zwecke der Verifikation benötigt man für gewöhnlich keine weiteren Klartext-/Chiffretextpaare mehr. War nämlich der gefundene Schlüssel Kc' korrekt, so muß sich das vollständige abgehörte Chiffrat zu einem sinnvollen Bitstring entschlüsseln lassen. Ist dies der Fall, so ist der Kc' gleich dem korrekten Sitzungsschlüssel Kc.

7.1.2 Anzahl der benötigten Schlüsselstrombits

Es stellt sich nun noch die Frage, wie viele zusammenhängende Schlüsselstrombits tatsächlich für die Kryptanalyse benötigt werden. Anders formuliert: Seien zwei Schlüsselstromfolgen echt verschieden, d.h. sie unterscheiden sich um mindestens 1 Bit. Wie lang müssen die beobachtbaren Teilsequenzen sein, damit sich bereits diese Teilsequenzen mit hoher Wahrscheinlichkeit unterscheiden?

Golić argumentiert in [Gol97], daß zum einen die einzelnen LFSR-Sequenzen maximal seien und somit die bekannt guten Autokorrelations-Eigenschaften aufwiesen. Zum anderen sei die Kombinationsfunktion (bitweise Addition modulo 2) korrelationsimmun. Bedenkt man, daß der verwendete Teil der Schlüsselstromsequenz nur aus maximal $2^{63,32}$ verschiedenen inneren Zuständen $S(t)$ erzeugt wird, so ist es sehr wahrscheinlich, daß sich echt verschiedene Schlüsselstromsequenzen bereits auf den ersten 64 Bit unterscheiden.

In Anbetracht der Tatsache, daß dem Kryptanalytiker Schlüsselstrombits normalerweise gleich "rahmenweise", d.h. in Blöcken zu 114 oder 228 Bits, zufallen, sollte diese Anforderung jedoch ohnehin kein Problem darstellen. Wir nehmen im folgenden also stets an, daß dem Kryptanalytiker 64 zusammenhängende Bits des Schüsselstroms, beginnend mit dem Bit $\tilde{t}$, bekannt sind, also die Bits $(z_{\tilde{t}}, \ldots, z_{\tilde{t}+63})$.

7.2 Brute-Force-Angriffe

7.2.1 Vollständige Suche

Die trivialste Möglichkeit, um einen gesuchten inneren Zustand $S(\tilde{t})$ zu finden, besteht darin, den kompletten Zustandsraum abzusuchen. Das bedeutet, daß man jeden möglichen inneren Zustand einmal in die Register lädt, den Schlüsselstromgenerator vorwärts laufen läßt und dann überprüft, ob der resultierende mit dem beobachteten Schlüsselstrom übereinstimmt.

Im vorliegenden Fall gibt es zunächst 2^{64} mögliche innere Zustände $\mathrm{S}(\tilde{t})$, doch läßt sich der Suchaufwand mit einigen Vorüberlegungen noch reduzieren:

- In Abschnitt 6.3 wurde bereits gezeigt, daß es zwar 2^{64} mögliche Anfangszustände $S(0)$ gibt, aber nur $\frac{5}{8} \cdot 2^{64} \approx 2^{63.32}$ mögliche Zwischenzustände $S(\tilde{t})$ für $\tilde{t} \geq 1$.

- Nach der obigen Annahme sind dem Kryptanalytiker $(z_{\tilde{t}}, \ldots, z_{\tilde{t}+63})$ und somit insbesondere $z_{\tilde{t}}$ bekannt. Es können also alle Zustände $\mathrm{S}(\tilde{t})$ ausgeschlossen werden, die die Konsistenzbedingung

$$z_{\tilde{t}} = s_{1,1}(\tilde{t}) \oplus s_{2,1}(\tilde{t}) \oplus s_{3,1}(\tilde{t})$$

  verletzen. Da dies unter der Annahme der Gleichverteilung der Schlüsselstrombits genau für die Hälfte aller Zustände gilt, verbleiben noch etwa $2^{62.32}$ mögliche Zwischenzustände $S(\tilde{t})$.

Wie bereits erwähnt, gibt es zwar möglicherweise mehrere konsistente innere Zustände $S(\tilde{t})$, jedoch nur einen korrekten Sitzungsschlüssel Kc. Nimmt man die Verifikation eines gefundenen inneren Zustandes sofort vor, so findet man nach im Mittel $\frac{1}{2} \cdot 2^{61,32}$ Ratedurchgängen den "korrekten" inneren Zustand.

Hält man sich nun vor Augen, daß eine vollständige Suche nach dem korrekten Sitzungsschlüssel (wie in Abschnitt 5.2 beschrieben) im Mittel lediglich 2^{53} Simulationsdurchläufe des etwa doppelten Umfangs erfordert, so ist die vollständige Suche an dieser Stelle keine sinnvolle Vorgehensweise.

7.2.2 Vollständige Speicherung

Es liegt nun der Gedanke nahe, diese vollständige Suche einmalig auszuführen und alle dabei auftretenden Ergebnisse (d.h. die Kombinationen aus den Schlüsselstromsequenzen $(z_{\tilde{t}}, \ldots, z_{\tilde{t}+63})$ und den zugehörigen $S(\tilde{t})$) in einer Tabelle abzuspeichern. Bei einem konkreten Angriff wäre dann die Schlüsselstromsequenz bekannt. Es müßten also nur noch die zugehörigen $S(\tilde{t})$ aus der Tabelle ausgelesen werden.

In diesem Falle kann die Zahl der zu berechnenden Zustände $S(\tilde{t})$ nicht von $2^{63.32}$ auf im Mittel $2^{62.32}$ reduziert werden, da ja die korrekte Schlüsselstromsequenz zum Zeitpunkt, an dem die Tabelle berechet wird, noch nicht bekannt ist. Auch können nicht unmittelbar die 2^{54} möglichen Sitzungsschlüssel Kc in einer solchen Tabelle gespeichert werden, da eine solche Tabelle auch noch die 2^{22} möglichen Rahmennummern berücksichtigen müßte und somit eine Größe von 2^{76} Einträgen erreichen würde.

Naiverweise besteht jeder der $2^{63.32}$ Tabelleneinträge aus einer Schlüsselstromsequenz (64 Bit) und einem Zwischenzustand (64 Bit). Bei geschickter Speicherorganisation kann man jedoch einen Teil des Speicherplatzbedarfs für die Schlüsselstromsequenz einsparen (Hashing). Obwohl eine vollständige Einsparung nicht möglich ist (weil zu einzelnen Schlüsselstromsequenzen ja auch mehrere Zwischenzustände gehören können), soll hier dennoch zur Vereinfachung die Annahme einer idealen Speicherorganisation getroffen werden, bei der jeder Eintrag nur 64 Bit bzw. 8 Byte benötigt.

Unter dieser Annahme beträgt dann der Speicherplatzbedarf $2^{63.32} \cdot 8 = 2^{66.32}$ Byte. Veranschlagt man die Speicherkapazität einer CD-ROM mit 600 MByte, so wären für die Speicherung dieser Tabelle in etwa $14,64 \cdot 10^{10}$ CD-ROMs erforderlich. Zur Veranschaulichung: Würde man diese CDs ohne Hüllen oder Zugriffsmechanismus zu einer Pyramide stapeln, so besäße diese das achtfache Volumen der Cheops-Pyramide! Somit dürfte dieser Angriff selbst bei verbesserten Speichermedien kaum praktikabel sein. Der Suchaufwand von maximal $\lceil log_2(2^{63.32}) \rceil = 64$ Zugriffen dagegen wäre natürlich ausgesprochen vorteilhaft.

7.3 Time-Memory-Tradeoff nach Golić

Ein Time-Memory-Tradeoff wird dann sinnvoll, wenn es dem Kryptanalysten gelingt, gleich mehrere Schlüsselstromsequenzen der Länge 64 Bit herauszufinden. Der Grundgedanke besteht darin, nur einen Teil aller möglichen Tupel $(S(\tilde{t}), (z_{\tilde{t}}, \ldots, z_{\tilde{t}+63}))$ abzuspeichern und darauf zu hoffen, daß mindestens eine der gefundenen Schlüsselstromsequenzen mit einer der gespeicherten Folgen $(z_{\tilde{t}}, \ldots, z_{\tilde{t}+63})$ übereinstimmt. Der folgende Angriff wurde von Golić in [Gol97] vorgestellt.

7.3.1 Allgemeine Vorgehensweise

Sei M die Zahl der gespeicherten Schlüsselstromsequenzen, wobei jede Sequenz in der Tabelle genau einmal vorkommt (Ziehen ohne Zurücklegen). Sei weiterhin F die Zahl der gefundenen Schlüsselstromsequenzen, wobei hier Sequenzen doppelt vorkommen können (Ziehen mit Zurücklegen). Dann beträgt die Wahrscheinlichkeit für mindestens eine Übereinstimmung nach den Gleichungen zum Geburtstagsproblem (siehe z.B. [MOV96], Formel 2.30):

$$\text{Prob} = 1 - \left(1 - \frac{M}{2^{63.32}}\right)^F \tag{7.1}$$

Da der Kryptanalyst eine Erfolgswahrscheinlichkeit erzielen will, die sehr nahe bei 1 liegt, setzt er Prob = 1 in Gleichung 7.1 ein und schätzt ab wie folgt:

$$\begin{aligned} 1 &= 1 - \left(1 - \frac{M}{2^{63.32}}\right)^F \\ 0 &= \left(1 - \frac{M}{2^{63.32}}\right)^F \\ 0 &\geq 1 - \frac{F \cdot M}{2^{63.32}} \\ F \cdot M &\geq 2^{63.32} \end{aligned}$$

Aufgrund der Ungenauigkeit der getroffenen Abschätzung ist die Wahrscheinlichkeit für eine Kollision ist zwar groß, aber nicht gleich 1. Sie zeigt aber, daß die Zahl M der Schlüsselstromsequenzen, die der Kryptanalyst a priori berechnen und abspeichern muß, umgekehrt proportional ist zur Zahl F der Zahl der Schlüsselstromsequenzen, die er zu finden hofft. Zu beachten ist dabei, daß sich ein solcher Angriff nur dann lohnt, wenn F noch immer signifikant kleiner ist als die Zahl der Rahmen, die mit dem Sitzungsschlüssel verschlüsselt wurden.

Das Erstellen der Wertetabelle erfordert einen einmaligen Aufwand von M Berechnungen und einen Speicherplatz von $M \cdot 8$ Byte. Die eigentliche Suche nach Koinzidenzen benötigt noch einmal $F \cdot log_2(M)$ Zugriffe auf die Tabelle.

7.3.2 Angriff gegen einen Sitzungsschlüssel

Es stellt sich nun die Frage, ab welcher Größenordnung für F ein solcher Time-Memory-Tradeoff praktikabel wird. Dazu überlegt man sich zunächst, daß der Kryptanalytiker aufgrund der relativ kurzen Rahmenlänge meist entweder alle 114 Schlüsselstrombits einer Übertragungsrichtung kennt oder aber gar keine.[1] In diesen 114 Schlüsselstrombits sind dann 51 Sequenzen zu 64 Bit enthalten. Nimmt man nun an, daß der Kryptanalyst in den Besitz von K Rahmen mit zugehörigem Klartext gelangt ist, so kennt er $F = K \cdot 51$ Schlüsselstromsequenzen. Also muß gelten:

$$51 \cdot K \cdot M \geq 2^{63.32}$$

Erwartet der Kryptanalyst beispielsweise, $K = 2^{15}$ Rahmen mit je 51 Schlüsselstromsequenzen herauszufinden, so muß seine Tabelle noch immer $M = 2^{42.65}$ Einträge enthalten. Für die Suche sind dann etwa $2^{20.67} \cdot \log_2(42.65) \approx 2^{23.01}$ Zugriffe erforderlich. Die Wahrscheinlichkeit, dabei wenigstens eine der belauschten Sequenzen in der Tabelle wiederzufinden, beträgt dann nach Gleichung 7.1 immerhin 0.632 bzw. 63.2%.

[1] Golić geht in [Gol97] davon aus, daß der Kryptanalyst sogar die 228 Schlüsselstrombits beider Übertragungsrichtungen kennt. Die Annahme, daß der Kryptanalyst Klar- und Chiffretext sowohl des Senders als auch des Empfängers kennt, ist jedoch eher unrealistisch, wenn kein Insider bei einem der Netzbetreiber involviert ist.

7.3.3 Angriff gegen mehrere Sitzungsschlüssel

Golić denkt in [Gol97] auch über die Möglichkeit nach, daß der beschriebene Time-Memory-Tradeoff gleichzeitig gegen mehrere GSM-Benutzer eingesetzt werden könnte, deren Verkehr der Kryptanalytiker abhört. Besteht diese Möglichkeit bei L Benutzern, so muß M so gewählt werden, daß die Gleichung

$$51 \cdot K \cdot L \cdot M \geq 2^{63.32}$$

erfüllt ist.

Es ist allerdings äußerst fraglich, wie der Kryptanalyst in den Besitz einer Vielzahl von Klartext-/Chiffretextpaaren von einer Vielzahl von Benutzern gelangen will. Außerdem kann er a priori nicht wissen, aus welchem Gespräch die Schlüsselstromsequenz, die er in seiner Tabelle wiederfindet, letztlich stammt. Zudem kann er nur einen geringen Teil der für ihn interessanten Gespräche tatsächlich entschlüsseln. Es bleiben also Zweifel, ob dieser Angriff tatsächlich in der Praxis sinnvoll einzusetzen ist.

7.4 Der Angriff nach Anderson

Einige ältere Quellen (z.B. [Shn96], [Wob98]) zitieren noch immer einen Angriff, den Anderson in [And94] vorgeschlagen hatte. Dieser Angriff wurde wie folgt beschrieben:

- Der Inhalt von Register 1 und 2 wird "geraten".
- Der Inhalt von Register 3 wird mit Hilfe des Schlüsselstroms und der Gleichung

  $$z_{\tilde{t}} = s_{1,1}(\tilde{t}) \oplus s_{2,1}(\tilde{t}) \oplus s_{3,1}(\tilde{t})$$

  berechnet.
- Der so erhaltene innere Zustand wird überprüft, indem man ihn einsetzt und das Register 64 Takte weit vorwärts laufen läßt. Stimmt der so erzeugte Schlüsselstrom mit dem beobachteten Schlüsselstrom überein, so wurde ein korrekter innerer Zustand gefunden.

Dieser Angriff ist jedoch so nicht durchführbar, da er die Taktkontrolle des Generators vernachlässigt. Da die Ausgabe der Taktkontrolle nämlich auch von LFSR_3 abhängt, müßte auch für jeden Takt t das Taktkontrollbit $s_{3,\tau_3}(t)$ geraten werden. Da man zur Rekonstruktion mindestens 23 Takte benötigt, müßte man somit auch den gesamten Inhalt von Register 3 raten und hätte nichts anderes als eine vollständige Suche durchgeführt.

Der Angriff wird allerdings praktikabel, wenn man zusätzlich zu den Belegungen von Register 1 und 2 nur die höherwertigen Bits aus Register 3 rät, so daß man die Taktkontrolle für die entsprechende Zahl von Takten kennt. Auf diese Weise müßte man $(23 - 13 + 1) = 11$ weitere Bits raten und könnte dann die 11 niederwertigen Bits für Register 3 mit Hilfe der obigen Gleichung errechnen. Das letzte, fehlende Bit müßte dann erneut erraten werden.

Mit diesem verbesserten Angriff würde man somit 11 Bit gegenüber der vollständigen Suche einsparen. Die Komplexität liegt also im Worst Case bei etwa $2^{63,32-11} = 2^{52,32}$ Rateversuchen, der gesuchte korrekte innere Zustand würde im Mittel schon nach $2^{51,32}$ Versuchen gefunden. Somit ist der verbesserte Angriff nach Anderson schon etwas effizienter als die in Abschnitt 5.2 vorgestellte vollständige Suche über alle Sitzungsschlüssel Kc.

7.5 Divide-and-Conquer-Angriff nach Golić

Es liegt also nahe, als Grundlage für einen noch effizienteren Angriff auf den Generator A5 nicht komplette Registerbelegungen, sondern einzelne Taktkontrollbits zu raten. Der im folgenden vorgestellte Angriff basiert auf dem von Golić in [Gol97] gemachten Vorschlag, enthält aber geringfügige Modifikationen sowohl im Algorithmus als auch in der Berechnung der Komplexität.

7.5.1 Aufstellen der Gleichungen

Gesucht sei ein innerer Zustand $S(\tilde{t})$, der einen Schlüsselstrom $(z_{\tilde{t}}, z_{\tilde{t}+1}, \ldots)$ erzeugt haben könnte. Um diesen inneren Zustand zu finden, raten wir zunächst einige Taktkontrollbits.

Wenn Taktkontrollbit s_{r,τ_r} bekannt ist, erhält man eine (häufig sehr simple) lineare Gleichung. Wir betrachten dazu zunächst ein einfaches Beispiel: Für den Takt $\tilde{t}+1$ sei das Taktkontrolltripel $(0,0,1)$ gegeben (z.B. durch Raten), wie in Abbildung 7.1 dargestellt. Diese drei Bits liefern uns die folgenden linearen Gleichungen:

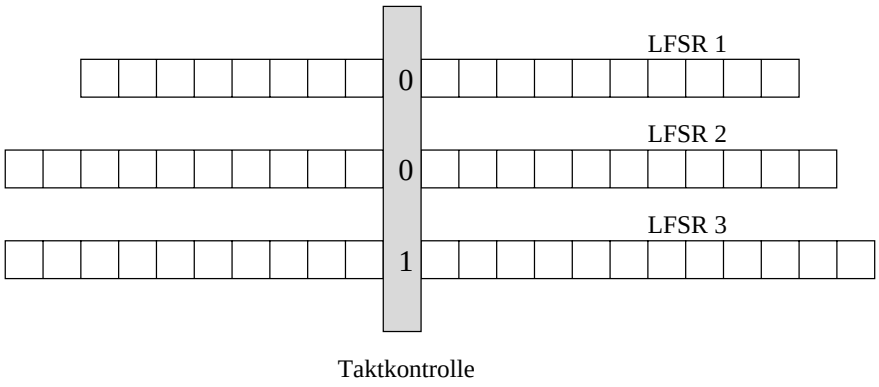

Abbildung 7.1: Geratene Kontrollbits für den ersten Takt

$$\begin{aligned} s_{1,11}(\tilde{t}) &= 0 \\ s_{2,12}(\tilde{t}) &= 0 \\ s_{3,13}(\tilde{t}) &= 1 \end{aligned}$$

Wir bezeichnen diese Art linearer Gleichung, die einem Bit direkt einen Wert zuweist, im folgenden als Gleichung vom Typ A.

Aus den drei bekannten Taktkontrollbits können wir folgern, daß mit Takt $\tilde{t}+1$ die Register 1 und 2 weitergetaktet werden. Für den nächsten Takt benötigen wir also erneut zwei Taktkontrollbits, nämlich je eines für Register 1 und Register 2. Angenommen, der zweite Takt wird durch die Taktkontrolle $(1,0,1)$ bestimmt (der Wert '1' für Register 3 stammt noch aus der letzten Durchgang). Dann können wir diese Bits wie in Abbildung 7.2 gezeigt in unser Schema eintragen: Gleichungen vom Typ A werden analog zu oben aufgestellt, also:

$$\begin{aligned} s_{1,12}(\tilde{t}) &= 1 \\ s_{2,13}(\tilde{t}) &= 0 \end{aligned}$$

Wir wissen somit, daß mit dem Takt $\tilde{t}+2$ die Register 1 und 3 weitergetaktet werden. Fahren wir nun fort, Werte für die Taktkontrolle einzutragen, so erreichen

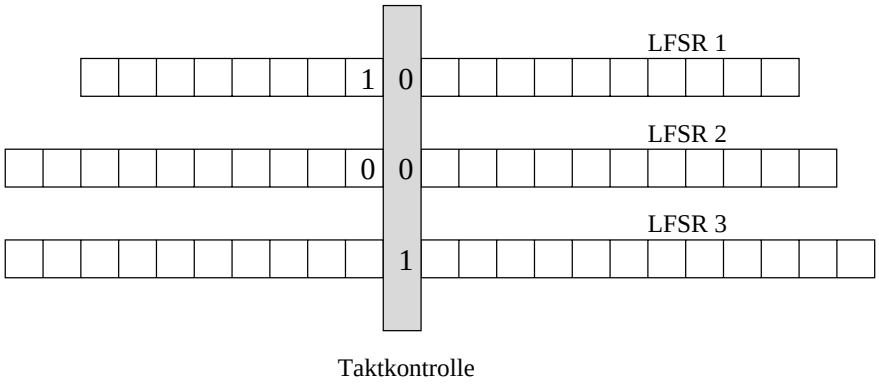

Abbildung 7.2: Geratene Kontrollbits für die ersten beiden Takte

wir einen Punkt, an dem die Bits nicht mehr direkt in das Schema eingetragen werden können. Ein solcher Fall wird in Abbildung 7.3 dargestellt. In diesem Fall

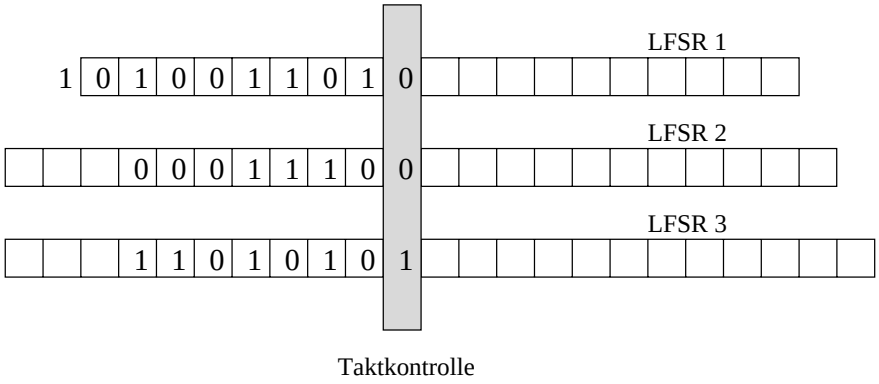

Abbildung 7.3: Geratene Kontrollbits für die ersten 11 Takte

können wir für Register 1 keine Gleichung vom Typ A mehr aufstellen. Stattdessen beschreiben wir das neue Bit mit Hilfe der Rekursionsgleichung des Registers, im vorliegenden Fall also als:

$$s_{1,1}(\tilde{t}) + s_{1,2}(\tilde{t}) + s_{1,3}(\tilde{t}) + s_{1,6}(\tilde{t}) = 1$$

Eine solche lineare Gleichung, die durch die Rückkopplungsfunktion des Schieberegisters bestimmt wird, bezeichnen wir als Gleichung vom Typ B.

Ein Gleichungssystem mit Gleichungen vom Typ A und B allein wäre nun allerdings wenig hilfreich. Immerhin gilt es, 64 Unbekannte zu ermitteln, wir benötigen dazu also auch 64 Gleichungen. Da wir mit jedem bekannten Bit genau eine Gleichung vom Typ A oder B erhalten, müßten wir also alle 64 Bit kennen (bzw. raten).

Aus diesem Grunde führen wir noch einen dritten Typ Gleichung ein, der sich aus dem Ausgabestrom ergibt. Ist beispielsweise das Schlüsselstrombit $z_{\tilde{t}}$ gleich 1, so wissen wir:

$$s_{1,1}(\tilde{t}) + s_{2,1}(\tilde{t}) + s_{3,1}(\tilde{t}) = 1$$

Eine solche Gleichung, die durch den Ausgabestrom bestimmt wird, nennen wir eine Gleichung vom Typ C.

Mit jedem bekannten Taktsteuerungstripel erhält man eine weitere Gleichung vom Typ C. Ist beispielsweise das Schlüsselstrombit $z_{\tilde{t}+1}$ gleich 0 und die Takt-

steuerung geraten wie oben, so lautet die nächste Gleichung:

$$s_{1,2}(\tilde{t}) + s_{2,2}(\tilde{t}) + s_{3,1}(\tilde{t}) = 0$$

da die Register 1 und 2 weitergetaktet wurden.

7.5.2 Der Backtracking-Algorithmus

Das Grundprinzip des resultierenden Angriffes läßt sich wie folgt beschreiben: Rate so viele Taktkontrollbits, bis sich daraus insgesamt 64 linear unabhängige Gleichungen ergeben haben. Löst man dann das resultierende Gleichungssystem auf, so erhält man einen Kandidaten für den gesuchten inneren Zustand $S(\tilde{t})$, der den bekannten Schlüsselstrom erzeugt haben kann. Da man für gewöhnlich bei dem Angriff nicht alle 64 bekannten Schlüsselstrombits verwendet hat, verifiziert man den Zustand zunächst gegen die nicht berücksichtigten Bits, indem man den Generator vorwärts laufen läßt und nachprüft, ob der resultierende mit dem bekannten Schlüsselstrom konsistent ist.

Formal besteht der Backtracking-Angriff aus den folgenden Schritten:

- Wir kennen von vornherein die erste Gleichung vom Typ C, da das erste Schlüsselstrombit noch von keiner Taktung beeinflußt wurde.
- Wir raten die erste Belegung der Taktkontrollbits. Auf diese Weise erhalten wir 3 Gleichungen vom Typ A und 1 Gleichung vom Typ C.
- Wir spannen nun einen Suchbaum auf, in dessen Knoten die geratenen Taktkontrollbits stehen. Im Mittel müssen wir dabei pro Knoten $\frac{9}{4}$ Bits raten und erhalten daraus $\frac{9}{4}$ Gleichungen vom Typ A oder B und 1 Gleichung vom Typ C. Abbildung 7.4 zeigt einen Ausschnitt eines solchen Suchbaumes.
- Für jede neue Gleichung wird geprüft, ob diese
  - linear unabhängig von den bisherigen Gleichungen
  - linear abhängig und konsistent mit den bisherigen Gleichungen oder
  - linear abhängig und im Widerspruch zu den bisherigen Gleichungen

  ist. Falls die Gleichung im Widerspruch zu den bisherigen steht, kann der aktuelle Teilbaum als falsch verworfen werden, in den beiden anderen Fällen wird die Suche fortgesetzt. Abbildung 7.5 zeigt das Vorgehen noch einmal schematisch
- Die Suche wird beendet, wenn 64 linear unabhängige Gleichungen gefunden sind.

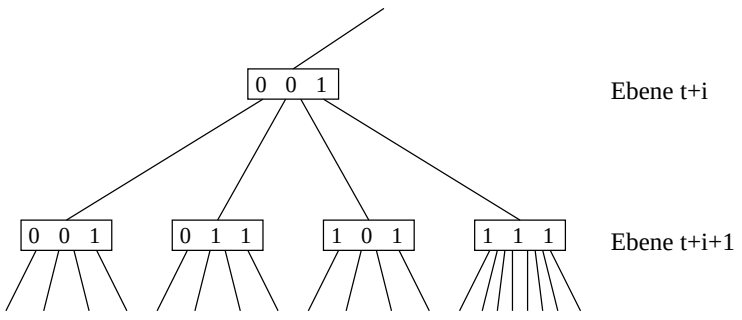

Abbildung 7.4: Ausschnitt aus dem Suchbaum

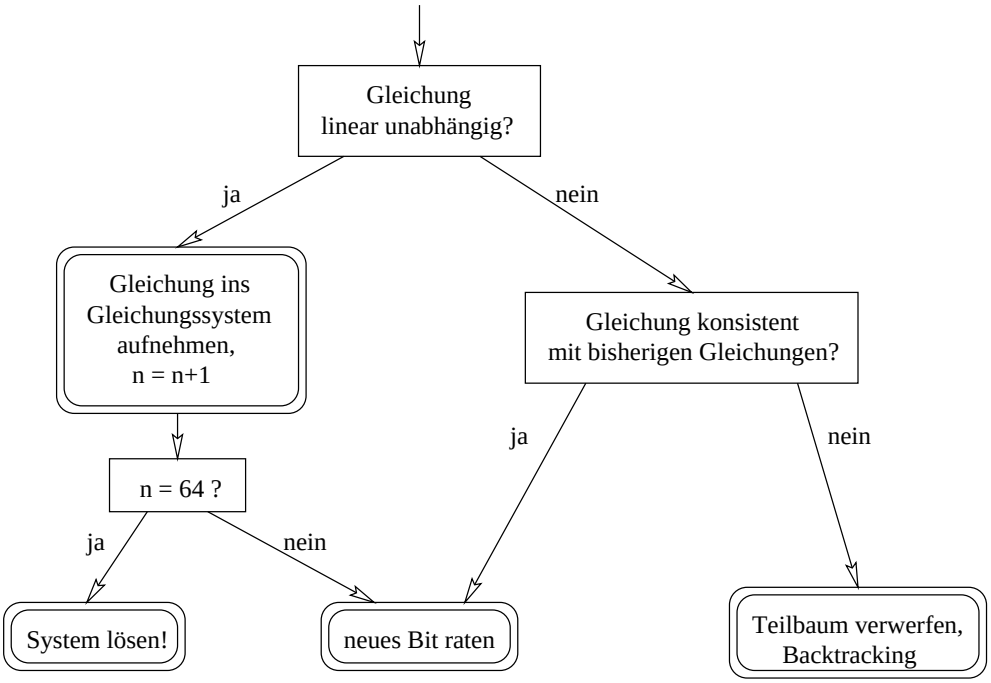

Abbildung 7.5: Vorgehensweise beim Backtracking

7.5.3 Lineare Unabhängigkeit der Gleichungen

Möchte man die Komplexität des Backtracking-Algorithmus untersuchen, so muß man sich die Frage stellen, unter welchen Bedingungen eine neu hinzugekommene Gleichung linear abhängig vom bereits existierenden Gleichungssystem ist und mit welcher Wahrscheinlichkeit eine linear abhängige Gleichung im Widerspruch steht zu den bisherigen. Es ist daher notwendig, zunächst einige Aussagen zur linearen Unabhängigkeit zu beweisen, bevor man die Komplexität des Angriffes berechnen kann.

Definition 7.1 *Gegeben sei ein Suchbaum wie in Abschnitt 7.5.2 beschrieben. Dann bezeichne T die aktuelle Tiefe des Suchbaumes, d.h. die Zahl der bisher geratenen Takte.*
Weiterhin bezeichne n_r die Zahl der Taktkontrollbits, die für Register r geraten wurden. Es gilt trivialerweise $n_r \leq T$ für $r = 1, 2, 3$.
Schließlich bezeichne $N = \sum_{r=1}^{3} n_r$ die Summe der geratenen Taktkontrollbits. Dann ist zu jedem Zeitpunkt die Anzahl der Gleichungen vom Typ A und B gleich N und die Anzahl der Gleichungen vom Typ C gleich $T + 1$.

Die lineare Unabhängigkeit einer neu hinzugekommenen Gleichung hängt zunächst davon ab, ob es sich um eine Gleichung vom Typ A bzw. B oder eine Gleichung vom Typ C handelt. Wir betrachten daher zunächst Gleichungen vom Typ A und B. Dabei fällt auf, daß das entstehende Gleichungssystem mit seinen N Gleichungen in drei voneinander unabhängige Systeme mit n_1, n_2 bzw. n_3 Gleichungen zerfällt. Zur Betrachtung der linearen Unabhängigkeit genügt es also, die einzelnen Schieberegister und die durch sie erzeugten Gleichungssysteme separat zu betrachten.

Satz 7.2 *Die durch ein linear rückgekoppeltes Schieberegister mit primitivem charakteristischen Polynom erzeugten Gleichungen vom Typ A und B sind immer linear unabhängig, falls die Zahl n_r der Gleichungen kleiner oder gleich der Länge L_r des Schieberegisters ist.*

Bew.:

Wir betrachten ein von einem linear rückgekoppelten Schieberegister der Länge L_r erzeugtes Gleichungssystem aus $n_r = L_r$ Gleichungen. Ist dieses Gleichungssystem linear unabhängig, so ist auch jedes Teilgleichungssystem aus $n_r \leq L_r$ Gleichungen linear unabhängig und die Aussage somit gezeigt.
Ein lineares Gleichungssystem mit L_r Gleichungen und L_r Unbekannten ist linear unabhängig genau dann, wenn sich die zugehörige Koeffizientenmatrix in obere oder untere Dreiecksform bringen läßt und alle Einträge auf der Hauptdiagonalen ungleich 0 sind. Wir wollen nun zeigen, daß dies für jedes Gleichungssystem aus Gleichungen vom Typ A und B gilt.
Sei τ_r der Taktausgang des Schieberegisters und $x^L + \sum_{i=1}^{L} a_i x^{i-1}$ das charakteristische Polynom. Sei $\hat{\imath}$ der größte Index, für den $a_i = 1$ gilt. Dann kann man zwei Fälle unterscheiden:
Fall 1: $\hat{\imath} \leq L_r - \tau_r + 2$
In diesem Fall entsteht kein "Überlauf", d.h. jede Formel vom Typ B entsteht einfach durch Linksverschieben der Koeffizienten der vorangegangenen Formel. In diesem Fall erhält man ein recht einfaches Gleichungssystem, das sich durch Zeilenvertauschung in untere Dreiecksform bringen läßt. Dabei sind tatsächlich alle Einträge auf der Hauptdiagonalen gleich 1, da die erste Gleichung vom Typ B in jedem Fall den Koeffizienten $s_{r,1}$ enthält (aufgrund der Primitivität des charakteristischen Polynoms, siehe auch Lemma 6.1 auf Seite 24), die zweite Gleichung vom Typ B den Koeffizienten $s_{r,2}$ usw. Abbildung 7.6 zeigt das Ergebnis der Zeilenvertauschung am Beispiel von LFSR_1 aus der konkreten Implementierung des A5.
Fall 2: $\hat{\imath} > L_r - \tau_r + 2$
In diesem Fall entsteht bei der Bildung von mindestens einer Gleichung vom Typ B ein Überlauf, d.h. es können nicht alle Gleichungen durch reines Linksverschieben der Koeffizienten gebildet werden. Wir betrachten zur Veranschaulichung ein Beispiel:
Nach 6 Schritten habe eine Koeffizientenmatrix die folgende Gestalt:

$$M = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Wie sieht nun die nächste Zeile dieser Matrix aus? Der intuitive Ansatz (zyklischer Links-Shift aller Einträge) ist natürlich falsch, stattdessen entsteht die neue Zeile aus der Bitsumme der linksverschobenen sechsten Zeile und den Koeffizienten der Rekursionsgleichung (also der fünften Zeile):

$$\begin{array}{rcccccccccc} & (& 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 &) \\ + & (& 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 &) \\ \hline & (& 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 &) \end{array}$$

Obwohl die resultierende Gleichung nun erheblich komplizierter aussieht als die Gleichungen im Falle $\hat{\imath} \leq L_r - \tau_r + 2$, kann das resultierende Gleichungssystem mit Leichtigkeit in die gleiche untere Dreiecksform gebracht werden. Zu diesem Zwecke zieht man einfach die fünfte Zeile wieder von der neu entstandenen Zeile ab. Der Rang der Matrix bleibt dabei unverändert. Wiederholt man dieses Vorgehen für alle Gleichungen, die durch Überlauf entstanden sind, so erhält man die gleiche Matrix, die man erhalten hätte, wenn man die Überlaufgleichungen von vornherein durch nicht-zyklischen Links-Shift erzeugt hätte. Durch Vertauschung der Zeilen kann man diese Matrix nun ebenso einfach in untere Dreiecksform bringen wie im ersten Fall, und erneut sind alle Einträge auf der Hauptdiagonalen gleich 1 aus dem gleichen Grunde wie oben. Abbildung 7.7 zeigt dies am Beispiel von LFSR_3. ■

```
(01) - - - - - - - - 1 - - - - - - - - - -      (09) 1 - - - - - - - - - - - - - - - - - -
(02) - - - - - - - 1 - - - - - - - - - - -      (08) - 1 - - - - - - - - - - - - - - - - -
(03) - - - - - - 1 - - - - - - - - - - - -      (07) - - 1 - - - - - - - - - - - - - - - -
(04) - - - - - 1 - - - - - - - - - - - - -      (06) - - - 1 - - - - - - - - - - - - - - -
(05) - - - - 1 - - - - - - - - - - - - - -      (05) - - - - 1 - - - - - - - - - - - - - -
(06) - - - 1 - - - - - - - - - - - - - - -      (04) - - - - - 1 - - - - - - - - - - - - -
(07) - - 1 - - - - - - - - - - - - - - - -      (03) - - - - - - 1 - - - - - - - - - - - -
(08) - 1 - - - - - - - - - - - - - - - - -      (02) - - - - - - - 1 - - - - - - - - - - -
(09) 1 - - - - - - - - - - - - - - - - - -      (01) - - - - - - - - 1 - - - - - - - - - -
(10) - - - - - - - - - - - - - 1 - - 1 1 1      (19) - - - - 1 - - 1 1 1 - - - - - - - - -
(11) - - - - - - - - - - - - 1 - - 1 1 1 -      (18) - - - - - 1 - - 1 1 1 - - - - - - - -
(12) - - - - - - - - - - - 1 - - 1 1 1 - -      (17) - - - - - - 1 - - 1 1 1 - - - - - - -
(13) - - - - - - - - - - 1 - - 1 1 1 - - -      (16) - - - - - - - 1 - - 1 1 1 - - - - - -
(14) - - - - - - - - - 1 - - 1 1 1 - - - -      (15) - - - - - - - - 1 - - 1 1 1 - - - - -
(15) - - - - - - - - 1 - - 1 1 1 - - - - -      (14) - - - - - - - - - 1 - - 1 1 1 - - - -
(16) - - - - - - - 1 - - 1 1 1 - - - - - -      (13) - - - - - - - - - - 1 - - 1 1 1 - - -
(17) - - - - - - 1 - - 1 1 1 - - - - - - -      (12) - - - - - - - - - - - 1 - - 1 1 1 - -
(18) - - - - - 1 - - 1 1 1 - - - - - - - -      (11) - - - - - - - - - - - - 1 - - 1 1 1 -
(19) - - - - 1 - - 1 1 1 - - - - - - - - -      (10) - - - - - - - - - - - - - 1 - - 1 1 1
```

Abbildung 7.6: Koeffizientenmatrix für $LFSR_1$ (rechts in Dreiecksform)

```
(01) - - - - - - - - - - 1 - - - - - - - - - - - -   1 - - - - - - - - - - - - - - - - - - - - - -
(02) - - - - - - - - - 1 - - - - - - - - - - - - -   - 1 - - - - - - - - - - - - - - - - - - - - -
(03) - - - - - - - - 1 - - - - - - - - - - - - - -   - - 1 - - - - - - - - - - - - - - - - - - - -
(04) - - - - - - - 1 - - - - - - - - - - - - - - -   - - - 1 - - - - - - - - - - - - - - - - - - -
(05) - - - - - - 1 - - - - - - - - - - - - - - - -   - - - - 1 - - - - - - - - - - - - - - - - - -
(06) - - - - - 1 - - - - - - - - - - - - - - - - -   - - - - - 1 - - - - - - - - - - - - - - - - -
(07) - - - - 1 - - - - - - - - - - - - - - - - - -   - - - - - - 1 - - - - - - - - - - - - - - - -
(08) - - - 1 - - - - - - - - - - - - - - - - - - -   - - - - - - - 1 - - - - - - - - - - - - - - -
(09) - - 1 - - - - - - - - - - - - - - - - - - - -   - - - - - - - - 1 - - - - - - - - - - - - - -
(10) - 1 - - - - - - - - - - - - - - - - - - - - -   - - - - - - - - - 1 - - - - - - - - - - - - -
(11) 1 - - - - - - - - - - - - - - - - - - - - - -   - - - - - - - - - - 1 - - - - - - - - - - - -
(12) - - - - - - - 1 - - - - - - - - - - - - 1 1 1   - - - - - - - - - 1 1 1 - - - - - - - - - - -
(13) - - - - - - 1 - - - - - - - - - - - - 1 1 1 -   - - - - - - - - - - 1 1 1 - - - - - - - - - -
(14) - - - - - 1 - - - - - - - - - - - - 1 1 1 - -   - - - - - - - - - - - 1 1 1 - - - - - - - - -
(15) - - - - 1 - - - - - - - - - - - - 1 1 1 - - -   - - - - - - - - - - - - 1 1 1 - - - - - - - -
(16) - - - 1 - - - - - - - - - - - - 1 1 1 - - - -   1 - - - - - - - - - - - - 1 1 1 - - - - - - -
(17) - - 1 - - - - - - - - - - - - 1 1 1 - - - - -   - 1 - - - - - - - - - - - - 1 1 1 - - - - - -
(18) - 1 - - - - - - - - - - - - 1 1 1 - - - - - -   - - 1 - - - - - - - - - - - - 1 1 1 - - - - -
(19) 1 - - - - - - - - - - - - 1 1 1 - - - - - - -   - - - 1 - - - - - - - - - - - - 1 1 1 - - - -
(20) - - - - - - - 1 - - - - 1 1 1 - - - - - 1 1 1   - - - - 1 - - - - - - - - - - - - 1 1 1 - - -
(21) - - - - - - 1 - - - - 1 1 1 - - - - - 1 1 1 -   - - - - - 1 - - - - - - - - - - - - 1 1 1 - -
(22) - - - - - 1 - - - - 1 1 1 - - - - - 1 1 1 - -   - - - - - - 1 - - - - - - - - - - - - 1 1 1 -
(23) - - - - 1 - - - - 1 1 1 - - - - - 1 1 1 - - -   - - - - - - - 1 - - - - - - - - - - - - 1 1 1
```

Abbildung 7.7: Koeffizientenmatrix für $LFSR_3$ (rechts in Dreiecksform)

Wir gehen im folgenden davon aus, daß $\max(n_r) \leq 19$ ist und daß somit die Gleichungen vom Typ A und B untereinander linear unabhängig sind. Bezieht man nun Gleichungen vom Typ C mit in die Betrachtung ein, so wird das resultierende Gleichungssystem erheblich unberechenbarer. Wir führen zunächst einen neuen Begriff ein:

Definition 7.3 *Gegeben sei ein linear unabhängiges Gleichungssystem G, bestehend aus Gleichungen vom Typ A, B und C.*
Ein Bit $s_{r,i}$ heißt ***bestimmt*** *durch das Gleichungssystem, wenn sich durch Linearkombination eine eindeutige Lösung für $s_{r,i}$ ermitteln läßt.*

Mit Hilfe dieser Definition gelangen wir nun zu folgender Aussage:

Satz 7.4 *Sei G ein lineares Gleichungssystem, bestehend aus Gleichungen vom Typ A, B und C. Sei weiterhin* $\max(n_r) \leq 19$.
Dann ist das Gleichungssystem G linear abhängig genau dann, wenn es eine Gleichung vom Typ C enthält, deren drei Summandenbits jeweils einzeln durch die übrigen Gleichungen aus G bestimmt sind.

Bew.:

Da $\max(n_r) \leq 19$ ist, sind die Gleichungen vom Typ A und B untereinander linear unabhängig. Das Gleichungssystem kann also dann und nur dann linear abhängig

sein, wenn sich eine Gleichung vom Typ C als Linearkombination der übrigen Gleichungen schreiben läßt.
Existiert eine solche Linearkombination, so kann sie immer in drei getrennte Teilgleichungssysteme zerlegt werden, die jeweils ein einzelnes Bit der gesuchten Gleichung beschreiben. Dies hat den Grund, daß

1. andere Gleichungen vom Typ C jeweils nur 1 Bit mit der gesuchten Gleichung gemeinsam haben können (schließlich werden stets mindestens 2 Register weitergetaktet) und
2. Gleichungen vom Typ A und B jeweils nur ein Register beschreiben und somit auch nur ein Bit der gesuchten Gleichung enthalten können.

Eine Gleichung vom Typ C ist also linear abhängig von den übrigen Gleichungen in G genau dann, wenn ihre Bits einzeln bestimmt sind. ■

Man kann nun für eine konkrete Implementierung des A5 untersuchen, wie groß die Zahl n_r der geratenen Taktkontrollbits sein muß, damit einzelne Bits $s_{r,i}$ bestimmbar sind. Dazu betrachtet man jeweils die ersten n_r Gleichungen, lößt sie soweit wie möglich auf und prüft nach, welche Bits dadurch bestimmt wurden. Für die von Briceno et al. gegebene Implementierung des A5 kommt man zu den folgenden Ergebnissen[2]:

Register 1: Trivialerweise liefert jede der 9 Gleichungen vom Typ A genau ein bestimmbares Bit, nämlich das Bit $s_{1,10+i}$. Komplizierter wird die Betrachtung erst, wenn Gleichungen vom Typ B mit in das Gleichungssystem aufgenommen werden. Es stellt sich heraus, daß für $n_1 = 17$ das Bit $s_{1,2}$ bestimmt ist und für $n_1 = 18$ die Bits $s_{1,3}$ und $s_{1,8}$. Für $n_1 = 19$ ist natürlich das gesamte Gleichungssystem lösbar.

Register 2: Register 2 liefert zunächst für jede der 11 Gleichungen vom Typ A ein bestimmbares Bit $s_{2,11+i}$. Die Gleichungen vom Typ B helfen erst dann bei der Bestimmung weiterer Bits, wenn $n_2 = 22$, also wenn das System vollständig lösbar ist.

Register 3: Register 3 liefert zunächst ebenfalls für die 11 Gleichungen vom Typ A je ein bestimmbares Bit $s_{3,12+i}$. Für $n_3 = 22$ sind die Bits $s_{3,1}, s_{3,4}, s_{3,7}$ und $s_{3,10}$ bestimmbar. Für $n_3 = 23$ ist das Gleichungssystem natürlich vollständig lösbar.

Ergebnis: Wichtig ist dabei vor allem die Beobachtung, daß für $\max(n_r) \leq 16$ einzig diejenigen Taktkontrollbits bestimmt sind, die durch die Gleichungen vom Typ A vorgegeben werden. Dies führt uns zu der folgenden naheliegenden Aussage:

Satz 7.5 *Gegeben sei ein lineares Gleichungssystem G, bestehend aus Gleichungen vom Typ A, B und C. Für die Zahl der Gleichungen vom Typ A und B gelte* $\max(n_r) \leq 16$ *für* $r = 1, 2, 3$. *Die "neueste" Gleichung vom Typ C sei beschrieben durch die Gleichung*

$$s_{1,\nu_1} + s_{2,\nu_2} + s_{3,\nu_3} = \alpha$$

Dann ist das lineare Gleichungssystem G in jedem Falle linear unabhängig, falls für höchstens ein ν_r *gilt:*

$$\nu_r \geq \tau_r \tag{7.2}$$

wobei τ_r *der Index des Taktkontrollbits ist wie in Kapitel 4 beschrieben.*
Analog ist das lineare Gleichungssystem G in jedem Falle linear abhängig, falls die Bedingung 7.2 für alle drei ν_r *erfüllt ist.*

[2] Auf eine Auflistung der Rechenwege wird verzichtet, da diese selbst im Anhang zu viel Platz in Anspruch nehmen würde und zudem der Informationswert nicht besonders hoch wäre.

Bew.:

Wegen $\max(n_r) \leq 16$ kann ein Bit nur dann bestimmt sein durch Gleichungen vom Typ A oder B, wenn es unmittelbar durch eine Gleichung vom Typ A bestimmt wird.
zu Aussage 1: Solange für höchstens ein ν_r die Bedingung 7.2 erfüllt ist, kann in jeder Gleichung vom Typ C jeweils höchstens ein Bit bestimmt sein. Somit kann keine dieser Gleichungen durch Linearkombination erzeugt werden.
zu Aussage 2: Ist für alle drei ν_r die Bedingung 7.2 erfüllt, so ist die Gleichung

$$s_{1,\nu_1} + s_{2,\nu_2} + s_{3,\nu_3} = \alpha$$

linear abhängig von den Gleichungen vom Typ A. ■

Schwierigkeiten bereitet uns also lediglich der Fall, wenn für genau zwei ν_r die Bedingung 7.2 erfüllt ist. Für diesen Fall läßt sich leider keine eindeutige Aussage treffen, so daß wir im nächsten Abschnitt Annahmen darüber treffen müssen.

7.5.4 Komplexität des Angriffes

Wir wollen nun basierend auf den obigen Ergebnissen die Komplexität des Angriffes für den Average Case bestimmen. Mit der "Komplexität" sei hier die durchschnittliche Breite des Suchbaumes gemeint, den man aufspannen muß, um einen Lösungskandidaten für $S(\tilde{t})$ zu finden. Um den Ergebnissen des letzten Abschnittes Rechnung zu tragen, erstellen wir ein "optimistisches" und ein "pessimistisches" Modell. Während das optimische Modell die tatsächliche Komplexität des Angriffes unterschätzt, gilt für das pessimistische Modell das Gegenteil. Die tatsächliche Komplexität liegt irgendwo zwischen den Werten, die für die beiden Modelle ermittelt werden.

Optimistisches Modell: Je früher die Gleichungen vom Typ C linear abhängig von den bisherigen Gleichungen sind, desto vorteilhafter ist dies für die Komplexität des Angriffes. Aus diesem Grunde gehen wir bei der optimistischen Abschätzung davon aus, daß ab dem Zeitpunkt, zu dem Bedingung 7.2 für zwei von drei Bits der jeweils "neuesten" Gleichung vom Typ C erfüllt ist, diese und alle nachfolgenden Gleichungen vom Typ C linear abhängig sind von den bisherigen Gleichungen.

- Wie in Abschnitt 7.5.2 beschrieben, erhalten wir zunächst eine Gleichung vom Typ C ohne weiteren Aufwand.
- Sodann raten wir die ersten drei Taktkontrollbits und erhalten vier Gleichungen: drei von Typ A und eine von Typ B.
- Nun wird der Suchbaum aufgespannt, wobei wir zunächst auf eine Verifikation der resultierenden Gleichungen verzichten können. Pro Ebene erhalten wir im Mittel $\frac{3}{4} \cdot 2 + \frac{1}{4} \cdot 3 = \frac{9}{4}$ Gleichungen vom Typ A und B und eine Gleichung vom Typ C. Im Mittel gehen dabei $\frac{3}{4} \cdot 4 + \frac{1}{4} \cdot 8 = 5$ Äste von jedem Knoten aus.
  Dieses Vorgehen wird so lange fortgesetzt, wie Bedingung 7.2 für höchstens ein Bit der neuesten Gleichung vom Typ C erfüllt ist. Da im Mittel jedes Register $\frac{3}{4}$ mal pro Ebene des Suchbaumes getaktet wird, ist im Mittel auf Ebene T des Baumes das Bit $\frac{3}{4}T+1$ berücksichtigt. Nimmt man vereinfachend an, daß alle drei Register sich genau gemäß ihren Erwartungswerten verhalten, so erreicht zunächst Register 1, dann Register 2 und als letztes Register 3 den "kritischen

Punkt". Somit ist Register 2 für unsere Analyse ausschlaggebend, und es muß gelten:

$$\begin{aligned} \frac{3}{4} \cdot T + 1 &< \tau_2 \\ \frac{3}{4} \cdot T + 1 &< 12 \\ T &< \frac{44}{3} \\ &\approx 14,667 \end{aligned}$$

Nach $T - 1 = \frac{41}{3} \approx 13,667$ Verzweigungsebenen ist also der kritische Punkt erreicht. Da wir eine Ebene weiter oben im Baum bereits anhalten müssen, haben wir $\frac{38}{3}$ Ebenen zu je $\frac{13}{4}$ Gleichungen durchlaufen und kennen somit im Mittel weitere 41,167 linear unabhängige Gleichungen.
Bem.: Da $13,667 \leq 16$ ist, ist im Normalfall auch die Bedingung $\max(n_r) \leq 16$ erfüllt, die Voraussetzung dafür ist, daß nur die "geratenen" Taktkontrollbits bestimmt sind.

- Wenn wir das Ergebnis aus Abschnitt 6.3 berücksichtigen, benötigen wir im Mittel nicht 64, sondern lediglich 63,32 linear unabhängige Gleichungen. Im Mittel fehlen also noch $63,32 - 1 - 4 - 41,167 = 17,153$ Gleichungen. Diese erhalten wir, indem wir den Suchbaum erweitern. Allerdings erhalten wir nun pro Ebene nur noch $\frac{9}{4}$ Gleichungen vom Typ A und B (die Gleichungen vom Typ C sind ja nun gemäß Annahme linear abhängig). Da die linear abhängigen Gleichungen aber genutzt werden können, um das bisherige Ergebnis zu verifizieren (eine abhängige Gleichung über $\mathbf{Z}_2$ führt unter der realistischen Annahme der Gleichverteilung der Schlüsselstrombits mit einer Wahrscheinlichkeit von 50% zu einem Widerspruch), kann im Mittel die Hälfte der neuen Blätter gleich wieder verworfen werden: Im Durchschnitt gehen folglich nur 2,5 Äste von jedem Blatt aus.
  Um die benötigten 17,153 linearen Gleichungen zu finden, müssen also im Mittel noch $17,153 \cdot \frac{4}{9} \approx 7,624$ Taktkontrollen geraten werden.

Die Breite des Suchbaumes im optimistischen Modell betägt also:

$$\begin{aligned} & 2^3 \cdot 5^{12,667} \cdot 2,5^{7,624} \\ \approx\ & 2^3 \cdot 2^{29,411} \cdot 2^{10,078} \\ \approx\ & 2^{42,49} \end{aligned}$$

Pessimistisches Modell: Wenn wir eine pessimistische Abschätzung erreichen wollen, so können wir annehmen, daß Gleichungen vom Typ C erst dann linear abhängig von den übrigen Gleichungen sind, wenn Bedingung 7.2 für alle drei Bits der "neuesten" Gleichung vom Typ C erfüllt ist. Das Vorgehen ist dabei natürlich identisch zu dem im optimistischen Modell beschriebenen, bei der Komplexitätsrechnung gibt es jedoch die folgenden Unterschiede:

- Beim Aufspannen des Suchbaumes ist nun Register 3 als "kritischer Punkt" ausschlaggebend, und es muß gelten:

$$\begin{aligned} \frac{3}{4} \cdot T + 1 &< \tau_3 \\ \frac{3}{4} \cdot T + 1 &< 13 \\ T &< 16 \end{aligned}$$

Der kritische Punkt wird also nach 15 Verzweigungsebenen erreicht, nach 14 Ebenen müssen wir anhalten, so daß wir diesmal im Mittel $14 \cdot \frac{13}{4} = 45,5$ linear unabhängige Gleichungen kennen.

- Somit fehlen uns diesmal nur noch $63,32 - 1 - 4 - 45,5 = 12,82$ Gleichungen. Diese erhalten wir im Mittel durch das Durchlaufen weiterer $12,82 \cdot \frac{4}{9} = 5,7$ Ebenen des Suchbaumes.

Die Komplexität des pessimistischen Modelles beträgt also

$$\begin{aligned} & 2^3 \cdot 5^{14} \cdot 2,5^{5,7} \\ \approx\ & 2^3 \cdot 2^{32,507} \cdot 2^{7,535} \\ \approx\ & 2^{43,04} \end{aligned}$$

Ergebnis: Der aufgespannte Suchbaum besitzt im Mittel also eine Breite zwischen $2^{42,49}$ und $2^{43,04}$. Es ist anzunehmen, daß die tatsächliche Breite etwas stärker zum pessimistischen Modell hin tendiert: Der Unterschied ist aber insgesamt nicht besonders groß, da die Breite des Suchbaumes nach dem pessimistischen Modell nur um den Faktor $2^{0,55} = 1,464$ größer ist als nach dem optimistischen Modell.

Auch hier gilt wieder, daß der korrekte innere Zustand $S(\tilde{t})$ bereits nach etwa der Hälfte der Schritte gefunden wird. Die mittlere Komplexität liegt somit zwischen $2^{41,49}$ und $2^{42,04}$, einer Größenordnung also, die für einen entschlossenen Angreifer mit moderner Hard- und Software kein wirkliches Hindernis mehr darstellt.

7.5.5 Kritik an der Komplexitätsrechnung nach Golić

Wie bereits erwähnt, basiert der oben beschriebene Angriff auf einer von Golić in [Gol97] vorgestellten Idee. Als dieses Paper im Jahre 1997 vorgestellt wurde, war die korrekte Implementierung des A5 noch nicht bekannt, der Autor versuchte daher, den Angriff so allgemein wie möglich zu halten. Die Breite des Suchbaumes wurde damals mit $2^{41,16}$ abgeschätzt, eine Anpassung an die mittlerweile bekannte Implementierung des A5 liefert eine Breite von $2^{41,61}$. Dennoch wurde Golićs Vorschlag hier nicht übernommen, sondern durch einen modifizierten Angriff mit einer vermeintlich höheren Komplexität ersetzt.

Der Grund dafür liegt darin, daß die Komplexitätsabschätzung nach Golić derart deutliche Schwächen aufweist, daß sie sich im Rahmen dieser Diplomarbeit als nicht haltbar erwies. Dabei sind insbesondere die folgenden Kritikpunkte zu nennen:

- Es werden unklare Begriffe verwendet, die nicht definiert werden.
- Es werden Aussagen ohne Beweis getroffen, die für den Leser nicht nachzuvollziehen sind.
- Es werden unsaubere Abschätzungen ("mit hoher Wahrscheinlichkeit") ohne Begründung getroffen.
- Es wird nicht unterschieden, welche Aussagen im allgemeinen Fall gelten und welche nur für die konkret betrachtete Implementierung des A5 korrekt sind.[3]

Im folgenden sollen die besagten Aussagen kurz untersucht und, soweit möglich, richtiggestellt oder widerlegt werden.

[3] Man beachte hierbei, daß die korrekte Implementierung zum Zeitpunkt der Veröffentlichung von [Gol97] gar nicht bekannt war.

Die Argumentation nach Golić: Der Angriff nach Golić beginnt damit, daß für jedes Register die ersten n Bits, die die Taktkontrolle beeinflussen, *en bloc* geraten werden. Zu diesem Zwecke ist folglich noch keine Baumstruktur notwendig, und man erhält $3n$ Gleichungen vom Typ A und B sowie im Mittel $\frac{4}{3}n + 1$ Gleichungen vom Typ C. Der entscheidende Punkt ist, daß es nach Golić einen a priori berechenbaren Wert

$$\hat{n} = \max(\tau_1, \tau_2, \tau_3) - 1$$

gibt, so daß gilt:

- Ist $n < \hat{n}$, so sind alle daraus resultierenden Gleichungen mit hoher Wahrscheinlichkeit linear unabhängig.
- Ist $n \geq \hat{n}$, so sind die daraus resultierenden Gleichungen mit hoher Wahrscheinlichkeit linear abhängig.

Dabei argumentiert er wie folgt:

1. Gleichungen vom Typ A und B sind untereinander linear unabhängig, falls $n \leq 19$.
2. Gleichungen vom Typ C sind untereinander linear unabhängig, falls $n \leq 18$.
3. Die Gleichungen vom Typ C sind linear unabhängig von den Gleichungen vom Typ A und B genau dann, wenn jede von ihnen mindestens ein "neues" Bit enthält, das noch nicht "geraten" wurde.
4. Dies ist "mit hoher Wahrscheinlichkeit" der Fall, wenn $n < \max(\tau_1, \tau_2, \tau_3) - 1$.
5. Wenn nicht, enthält die letzte Gleichung vom Typ C "zwingend" einige der bereits "geratenen" Bits und ist "mit hoher Wahrscheinlichkeit" linear abhängig von den vorangegangenen Gleichungen.

Im folgenden soll zu den einzelnen Schritten dieser Argumentation Stellung bezogen werden.

zu 1) Diese Aussage ist korrekt, wie in Lemma 7.2 gezeigt wurde.

zu 2) Diese Aussage ist zwar nicht falsch, sie geht jedoch nicht weit genug. Jede neue Gleichung vom Typ C enthält mindestens 2 Bit, die noch in keiner anderen Gleichung vom Typ C vorgekommen sind. Abhängig sein kann sie aber erst, wenn alle drei Bits einer Gleichung schon in anderen Gleichungen vorgekommen sind. Dies ist nicht etwa wie behauptet schon ab $n = 18$ möglich, sondern frühestens ab $n = 21$. Allerdings tut diese Ungenauigkeit der Korrektheit der Argumentation nach Golić an dieser Stelle noch keinen Abbruch.

zu 3) Diese Aussage krankt zunächst daran, daß die Begriffe "neues Bit" und "geratenes Bit" nicht definiert sind:

- Mit einem "neuen Bit" könnte entweder ein Bit gemeint sein, das zum ersten Mal in einer Gleichung vom Typ C vorkommt, oder aber ein Bit, das "geraten" wurde.
- Unter einem "geratenen Bit" wiederum kann man ein Bit verstehen, das durch eine Gleichung vom Typ A bestimmt wurde, oder eines, das durch Gleichungen vom Typ A und B bestimmt ist im Sinne von Definition 7.3.

Wenn mit einem "geratenen" Bit ein Bit gemeint ist, das durch eine Gleichung vom Typ A bestimmt wurde, dann ist die getroffene Aussage leicht zu widerlegen. Natürlich kann eine Gleichung vom Typ C auch dann linear abhängig sein, wenn all ihre Bits durch beliebige Gleichungen bestimmt sind im Sinne von Definition 7.3. Wir nehmen also an, daß mit "geratenen" Bits diejenigen gemeint sind, die im Sinne von Definition 7.3 durch Gleichungen vom Typ A und B bestimmt sind. Die getroffene Aussage kann dann wie folgt umformuliert werden:
Die Gleichungen vom Typ C sind linear unabhängig von den Gleichungen vom Typ A und B genau dann, wenn jede von ihnen mindestens ein "neues" Bit enthält, das noch nicht durch Gleichungen vom Typ A und B bestimmt ist.
Es bleibt noch die Unklarheit darüber, was mit einem "neuen" Bit gemeint ist. Die obige Aussage kann jedoch für beide denkbaren Bedeutungen widerlegt werden.

Nehmen wir zunächst an, daß ein "neues" Bit ein Bit bezeichnet, das noch nicht in einer Gleichung vom Typ C vorgekommen ist. Dann widerlegt das folgende Lemma die Aussage von Golić:

Lemma 7.6 *Ein Gleichungssystem mit Gleichungen vom Typ A, B und C sei linear unabhängig. Dann folgt daraus nicht notwendigerweise, daß jede Gleichung vom Typ C mindestens ein Bit enthält, das noch in keiner anderen Gleichung vom Typ C vorgekommen ist und nicht durch Gleichungen vom Typ A und B bestimmt ist.*

Bew.:

Annahme: Aus der linearen Unabhängigkeit des Gleichungssystems folgt, daß jede Gleichung vom Typ C mindestens ein Bit enthält, das noch in keiner anderen Gleichung vom Typ C vorgekommen ist und das unbestimmt ist.
Wir betrachten ein konkretes Beispiel ein Gleichungssystem wie folgt: Seien alle Gleichungen vom Typ A und B und Gleichungen C_1 bis C_{T-1} linear unabhängig. Die Gleichungen C_T bzw. C_{T+1} lauten wie folgt:

$$\begin{aligned} \underbrace{s_{1,10} + s_{2,11} + s_{3,12}}_{\text{alle unbest.}} &= 1 \\ s_{1,10} + \underbrace{s_{2,12}}_{\text{best.}} + \underbrace{s_{3,13}}_{\text{best.}} &= 0 \end{aligned}$$

Gleichung C_{T+1} ist dabei durch Takten der beiden Register 2 und 3 entstanden. Das resultierende Gleichungssystem ist linear unabhängig, obwohl Gleichung C_{T+1} kein Bit enthält, das sowohl neu als auch unbestimmt wäre.
→ Widerspruch zur Annahme. ■

Nehmen wir nun stattdessen an, daß ein "neues" Bit ein Bit bezeichnet, das unbestimmt ist. In diesem Fall kann die Aussage durch das folgende Lemma widerlegt werden:

Lemma 7.7 *Wenn jede Gleichung vom Typ C mindestens ein Bit enthält, das nicht durch Gleichungen vom Typ A und B bestimmt ist, dann folgt daraus nicht notwendigerweise, daß das Gleichungssystem linear unabhängig ist.*

Bew.:

Annahme: Aus der Tatsache, daß jede Gleichung vom Typ C mindestens ein Bit enthält, das nicht durch Gleichungen vom Typ A und B bestimmt ist, folgt, daß das Gleichungssytem linear unabhängig ist.
Wir betrachten als Beispiel ein Gleichungssystem wie folgt: Seien alle Gleichungen vom Typ A und B und Gleichungen C_1 bis C_{T-1} linear unabhängig. Jede der Gleichungen vom Typ C enthalte ein Bit, das nicht durch Gleichungen vom Typ A und

B bestimmt wird. Die Gleichungen C_T bzw. C_{T+1} lauten wie folgt:

$$\begin{aligned} \underbrace{s_{1,11} + s_{2,12}}_{\text{best. durch A+B}} + s_{3,11} &= 1 \\ \underbrace{s_{1,12} + s_{2,13}}_{\text{best. durch A+B}} + s_{3,11} &= 0 \end{aligned}$$

Gleichung C_{T+1} ist dabei durch Takten der beiden Register 1 und 2 entstanden. Gleichung C_{T+1} ist dabei linear abhängig von den bisherigen Gleichungen. Folglich ist auch das resultierende Gleichungssystem linear abhängig, obwohl alle Gleichungen vom Typ C ein Bit enthalten, das nicht durch Gleichungen vom Typ A und B bestimmt ist.
$\rightarrow$ Widerspruch zur Annahme. ■

Die einzige Möglichkeit, die Aussage von Golić als korrekt zu deuten, besteht darin, ein "neues" Bit als eines zu interpretieren, das unbestimmt ist durch alle bisherigen Gleichungen vom Typ A, B und C. Dies führt zu der in Lemma 7.4 bewiesenen Aussage.

zu 4) Da die Argumentationskette in allen anderen Fällen bereits durchbrochen ist, gehen wir im folgenden davon aus, daß Aussage 3 formuliert werden kann als:
Eine neue Gleichung vom Typ C ist linear unabhängig von den bisherigen Gleichungen, wenn sie mindestens ein Bit enthält, das noch nicht durch die bisherigen Gleichungen bestimmt wurde.
Nehmen wir nun an, daß die Bedingung $n < \max(\tau_1, \tau_2, \tau_3) - 1$ erfüllt ist. In diesem Falle ist der höchste Index in der letzten Gleichung vom Typ C genau gleich $\max(\tau_1, \tau_2, \tau_3) - 1$. Im Falle der konkreten Implementierung des A5 folgt daraus, daß $\max(n_r) = n < 16$ gilt und daß das erzeugte Gleichungssystem immer linear unabhängig ist (d.h. mit einer Wahrscheinlichkeit von 1). Es sei aber betont, daß sich eine solche Aussage nur für die konkrete Implementierung treffen läßt und nicht im allgemeinen Fall. Es sind Fälle denkbar, in denen die Taktausgänge weit auseinander liegen oder in denen die ersten Bits bereits für weitaus kleinere n_r bestimmt sind. In diesen Fällen ist selbst eine ungenaue Aussage wie "mit hoher Wahrscheinlichkeit" nicht mehr haltbar.

zu 5) Da für $n \geq \max(\tau_1, \tau_2, \tau_3) - 1$ der höchste Index in der letzten Gleichung vom Typ C genau gleich $\max(\tau_1, \tau_2, \tau_3)$ ist, ist tatsächlich mindestens ein bestimmtes Bit in der Gleichung enthalten. Die Folgerung, daß die letzte Gleichung deshalb mit hoher Wahrscheinlichkeit linear abhängig von den bisherigen Gleichungen sei, ist jedoch ohne eingehende statistische Analyse nicht haltbar, und wenn doch, dann erneut nur für eine konkrete Implementierung.

Um die Aussage für die tatsächliche Implementierung des A5 zu verifizieren, wurde die Entwicklung von Gleichungen vom Typ C am Rechner simuliert. Dabei wurde wie folgt vorgegangen:

- Die Taktkontrolle wurde simuliert, so daß jede der vier möglichen Taktkombinationen mit Wahrscheinlichkeit von je $\frac{1}{4}$ vorkam.
- Mit jedem Mastertakt wurde jedes Register entsprechend der Taktkontrolle weitergetaktet.
- Wenn eines der Register 12-mal getaktet wurde ($n = \max(\tau_1, \tau_2, \tau_3) - 1$), wurde die Simulation beendet.

Am Ende der Simulation wurde die letzte Gleichung vom Typ C betrachtet. Dabei wurden drei Resultate unterschieden:

- Wenn nur ein Bit der Gleichung bestimmt war, konnte die Gleichung als **mit Sicherheit unabhängig** betrachtet werden (siehe dazu Lemma 7.5) Dies war in immerhin 41,9% der Simulationen der Fall.
- Wenn zwei Bit der Gleichung bestimmt waren, mußte auf eine Aussage zur linearen Unabhängigkeit verzichtet werden. Dies war in 48,4% der Simulationen der Fall.
- Wenn alle drei Bit der Gleichung bestimmt waren, konnte die Gleichung als **mit Sicherheit abhängig** betrachtet werden (siehe ebenfalls Lemma 7.5). Dies war in lediglich 9,7% der Simulationen der Fall.

Ausgeführt wurden insgesamt 1.000.000 Simulationen, so daß die Stichprobe als groß genug betrachtet werden kann. Als Ergebnis kann festgehalten werden, daß für $n = \max(\tau_1, \tau_2, \tau_3) - 1$ keine Rede davon sein kann, daß die resultierenden Gleichungen "mit hoher Wahrscheinlichkeit" linear abhängig sind.

Folgerung: Der Splitpunkt $\hat{n} = \max(\tau_1, \tau_2, \tau_3) - 1$, den Golić als Voraussetzung für seinen Angriff benutzt, existiert zumindest für die tatsächliche Implementierung des A5 nicht. Auch für größere n kann das resultierende Gleichungssystem noch mit signifikanter Wahrscheinlichkeit linear unabhängig sein. Aus diesem Grunde ist es auch nicht möglich, ab einem im voraus berechneten Punkt anzunehmen, daß die Hälfte der geratenen Gleichungen sofort wieder verworfen werden kann. Ein Angriff, der auf einem solchen Splitpunkt basiert, besitzt somit in jedem Falle eine höhere als die angegebene Komplexität; ihre genaue Größenordnung ist jedoch nicht ohne weiteres ersichtlich. Aus diesem Grunde wurde in Abschnitt 7.5.2 ein Angriff vorgestellt, dessen Komplexität ohne größere Schwierigkeiten abgeschätzt werden kann. Es ist allerdings anzunehmen, daß die tatsächliche Komplexität des Angriffes nach Golić sich in einer ähnlichen Größenordnung bewegt.

Kapitel 8

Rekonstruktion des Anfangszustandes

Der zweite Schritt des Inversionsangriffes auf die Chiffre A5 besteht in der Rekonstruktion des Anfangszustandes $S(0)$. Voraussetzung hierfür ist, daß ein beliebiger Zwischenzustand $S(t), t \geq 0$, bekannt ist. In der folgenden Beschreibung wird von $t = 101$ ausgegangen, da dies der frühestmögliche Zustand ist, der mit den in Kapitel 7 beschriebenen Methoden rekonstruiert werden kann. Die Vorgehensweisen, die im folgenden dargelegt werden, sind jedoch auch für jedes andere t anwendbar.

8.1 Mathematische Vorbemerkungen

Die Arbeitsweise eines linear rückgekoppelten Schieberegisters entspricht mathematisch gesehen einer linearen Abbildung: Sie überführt einen inneren Zustand $S_r(t)$ auf lineare Weise in einen inneren Zustand $S_r(t+1)$. Schreibt man den inneren Zustand des Registers zum Zeitpunkt t nun als Zeilenvektor $\vec{s}_r(t)$, so muß es eine Zustandsübergangsmatrix A_r geben mit

$$\vec{s}_r(t+1) = \vec{s}_r(t) \cdot A_r \tag{8.1}$$

Diese Zustandsübergangsmatrix hat die folgende Gestalt:

$$A_r = \begin{pmatrix} 0 & 0 & \cdots & 0 & a_{r,1} \\ 1 & 0 & \cdots & 0 & a_{r,2} \\ 0 & 1 & \cdots & 0 & a_{r,3} \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 1 & a_{r,L_r} \end{pmatrix} \tag{8.2}$$

wobei $a_{r,\lambda}$ mit $\lambda = 1, \cdots, L_r$ die Repräsentationen der Rückkopplungsausgänge sind wie in Abschnitt 4.2.1 beschrieben.

Lemma 8.1 *Besitzt die von einem linear rückgekoppelten Schieberegister r erzeugte Sequenz maximale Periode, so ist die zugehörige Matrix A_r invertierbar.*

Bew.:

Eine Matrix ist invertierbar genau dann, wenn ihr Zeilenrang maximal ist (siehe z.B. [Fis86], S. 88). Es genügt also, zu zeigen, daß die Matrix A_r maximalen Zeilenrang hat.

Nach Lemma 6.1 ist $a_{r,1} = 1$ für alle linear rückgekoppelten Schieberegister mit maximaler Periode. Setzt man $a_{r,1} = 1$ ein, so kann man A_r durch Zeilenvertauschung in die folgende Form bringen:

$$A'_r = \begin{pmatrix} 1 & 0 & \cdots & 0 & a_{r,2} \\ 0 & 1 & \cdots & 0 & a_{r,3} \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 1 & a_{r,L_r} \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$$

Die Matrix befindet sich nun in oberer Dreiecksform, und alle Einträge auf der Diagonalen sind gleich 1. Sie hat also maximalen Zeilenrang und ist somit invertierbar. ■

Sei $c_r(\vartheta)$ die Taktung von LFSR$_r$ zum Zeitpunkt ϑ wie in Abschnitt 4.2.2 beschrieben. Dann sei $\Theta_r(t)$ definiert als die Anzahl der Takte, um die das Register r zum Zeitpunkt t insgesamt weitergetaktet wurde, d.h.

$$\Theta_r(t) = \sum_{\vartheta=1}^{t} c_r(\vartheta)$$

Da der innere Zustand von LFSR$_r$ zum Zeitpunkt t genau dem inneren Zustand nach $\Theta_r(t)$ Zustandsübergängen entspricht, gilt

$$\vec{s}_r(t) = \vec{s}_r(0) \cdot {A_r}^{\Theta_r(t)} \tag{8.3}$$

was sich leicht aus Gleichung 8.1 durch vollständige Induktion herleiten läßt.

Im vorliegenden Fall ist jedoch $\vec{s}_r(101)$ bekannt, während $\vec{s}_r(0)$ gesucht wird. Deshalb muß Gleichung 8.3 entsprechend umgestellt werden, was aufgrund der Invertierbarkeit von A_r zulässig ist:

$$\vec{s}_r(0) = \vec{s}_r(t) \cdot {A_r}^{-\Theta_r(t)} \tag{8.4}$$

8.2 Rekonstruktion durch vollständige Suche

Der Kryptanalytiker kann nun versuchen, Gleichung 8.4 zu lösen. Er hat die $\vec{s}_r(101)$ mit einem der in Kapitel 7 vorgestellten Verfahren ermittelt, außerdem kennt er die Zustandsübergangsmatrizen A_r. Ihm fehlen somit nur noch die Werte der Θ_r[1] für eine erfolgreiche Rekonstruktion der $\vec{s}_r(0)$ und somit von $S(0)$.

8.2.1 Vorgehensweise

Die Rekonstruktion des Anfangszustandes $S(0)$ zerfällt im wesentlichen in zwei Phasen, nämlich

- eine Vorbereitungsphase, die nur einmal durchlaufen wird, gleichgültig, wie viele Angriffe auf den A5 später ausgeführt werden, sowie
- den eigentlichen Angriff, der für jedes ermittelte $S(101)$ erneut absolviert werden muß.

[1] Korrekterweise müßte es hier statt Θ_r richtiger $\Theta_r(101)$ heißen. Da im folgenden Text jedoch Θ_r ausschließlich für den Zeitpunkt $t = 101$ betrachtet wird, wird im weiteren die vereinfachte Schreibweise gewählt.

In der **Vorbereitungsphase** berechnet der Kryptanalytiker die Matrizen ${A_r}^{-i}$ für alle $r = 1, 2, 3$ und alle $i = 0, \cdots, 101$. Die so erhaltenen 306 Matrizen werden abgespeichert, da sie ständig wieder benötigt werden und da ihre Berechnung weitaus aufwendiger ist als ein Table-Lookup. Der von der Tabelle benötigte Speicherplatz von ca. 17 kByte stellt für moderne Rechner kein Problem dar.

Der eigentliche **Angriff** besteht dann darin, für jeden Durchlauf ein Tripel $(\tilde{\Theta}_1, \tilde{\Theta}_2, \tilde{\Theta}_3)$ zu wählen und auf Korrektheit zu überprüfen. Für jedes der drei Schieberegister errechnet man in jeder Runde mit Hilfe von Gleichung 8.4 eine potentielle Anfangsbelegung $\tilde{S}_r(0)$ (die zugehörigen Matrizen ${A_r}^{-\tilde{\Theta}_r}$ werden aus der in der Vorbereitungsphase gebildeten Tabelle abgelesen). Diese drei Werte zusammen ergeben einen potentiellen Anfangszustand $\tilde{S}(0)$, den man verifiziert, indem man das Register nun 101 Takte vorwärts laufen läßt. Stimmt der so erzeugte Zustand $\tilde{S}(101)$ mit dem bekannten inneren Zustand $S(101)$ überein, so hat man einen möglichen korrekten Anfangszustand gefunden. Bei Ungleichheit dagegen kann man das Tripel $(\tilde{\Theta}_1, \tilde{\Theta}_2, \tilde{\Theta}_3)$ als mit Sicherheit falsch verwerfen und den nächsten Durchlauf starten.

8.2.2 Komplexität

Am einfachsten kann dieser Angriff implementiert werden, indem man für jedes Register r alle 102 Werte für Θ_r ausprobiert. Da Θ_r für jedes der drei Register Werte zwischen 0 und 101 annehmen kann, zeigt eine erste, naive Überlegung zur Komplexität einer vollständigen Suche, daß man maximal $(102)^3 \approx 10^6$ Durchläufe benötigt, um alle möglichen korrekten Anfangszustände $S(0)$ zu finden. Obwohl auch diese Zahl bereits für kryptographische Verhältnisse sehr niedrig ist (entspricht $2^{20,02}$ Iterationen), läßt sie sich durch eine weitere Überlegung noch zusätzlich reduzieren, ohne daß der Algorithmus dadurch an Genauigkeit verliert.

Aus der Beschreibung des A5-Algorithmus geht hervor, daß mit jedem Takt entweder 2 oder 3 Register getaktet werden. Für die Kryptanalyse bedeutet das, daß die folgende Ungleichung erfüllt sein muß:

$$2 \cdot t \leq \sum_{r=1}^{3} \Theta_r \leq 3 \cdot t \tag{8.5}$$

Satz 8.2 *Die Zahl $\#T_{good}$ der Tripel $(\Theta_1, \Theta_2, \Theta_3)$, die der Bedingung 8.5 genügen, ist gegeben durch:*

$$\#T_{good} = \tfrac{1}{6} \cdot (t^3 + 6t^2 + 11t + 6)$$

Bew.:

Jedes Θ_r kann theoretisch Werte zwischen 0 und t annehmen, d.h. es gibt insgesamt $(t+1)^3$ denkbare Tripel $(\Theta_1, \Theta_2, \Theta_3)$.
Betrachtet man alle $(t+1)^2$ Tripel mit $\Theta_1 = 0$, so stellt man fest, daß genau ein solches Tripel die Bedingung 8.5 erfüllt, nämlich das Tripel $(0, t, t)$.
Nun betrachtet man alle $(t+1)^2$ Tripel mit $\Theta_1 = 1$. Es stellt sich heraus, das von diesen stets 3 Tripel die Bedingung 8.5 erfüllen, nämlich $(1, t, t)$, $(1, t-1, t)$ und $(1, t, t-1)$.
Setzt man diese Betrachtung fort, so erhält man 6 gültige Tripel für $\Theta_1 = 2$, 10 Tripel für $\Theta_1 = 3$ usw.
Für den allgemeinen Fall gilt: Hält man $\Theta_1 = i$ fest, so muß die Gleichung $\Theta_2 + \Theta_3 \geq 2t - i$ erfüllt sein. Dazu gibt es

1 Tripel	mit	$\Theta_2 + \Theta_3 = 2t$,
2 Tripel	mit	$\Theta_2 + \Theta_3 = 2t - 1$,
3 Tripel	mit	$\Theta_2 + \Theta_3 = 2t - 2$,
$\vdots$		$\vdots$
i+1 Tripel	mit	$\Theta_2 + \Theta_3 = 2t - i$.

Folglich erhält man für festes $\Theta_1 = i$ insgesamt $\sum_{j=1}^{i+1} j$ gültige Tripel. Für die Summe über alle i gilt somit:

$$\begin{aligned} \#T_{good} &= \sum_{i=0}^{t} \sum_{j=1}^{i+1} j \\ &= \sum_{i=1}^{t+1} \sum_{j=1}^{i} j \end{aligned}$$

Mit Gleichung C.4 aus dem Anhang ergibt sich also:

$$\#T_{good} = \frac{1}{6} \cdot (t^3 + 6t^2 + 11t + 6)$$

Damit ist Satz 8.2 bewiesen. ■

Betrachtet man nun den konkreten Fall $t = 101$, so erhält man:

$$\begin{aligned} \#T_{good} &= \tfrac{1}{6} \cdot ((101)^3 + 6(101)^2 + 11(101) + 6) \\ &= \tfrac{1}{6} \cdot (1030301 + 61206 + 1111 + 6) \\ &= \tfrac{1}{6} \cdot 1092624 \\ &= 182.104 \end{aligned}$$

Somit können durch das Einfügen der einfachen Prüfbedingung 8.5 bereits etwa fünf von sechs Tripeln a priori verworfen werden. Die Komplexität der vollständigen Suche sinkt auf etwa $2^{17,47}$.

8.3 Der Angriff nach Golić

Unter Umständen kann der ohnehin schon geringe Aufwand der vollständigen Suche durch stochastische Überlegungen weiter reduziert werden. Der folgende Abschnitt befaßt sich mit einem entsprechenden Vorschlag, den Golić in [Gol97] macht, und ergänzt ihn um konkrete Aussagen zur Erfolgswahrscheinlichkeit und zur Komplexität.

8.3.1 Stochastische Vorbemerkung

In Abschnitt 6.2 wurde gezeigt, daß jedes Register im Mittel $\frac{3}{4}$-mal mit jedem Mastertakt getaktet wird, d.h. $E(c_r) = \frac{3}{4}$.[2] Daraus folgt für die Gesamtzahl der Taktungen zum Zeitpunkt $t = 101$:

$$\begin{aligned} E(\Theta_r) &= \sum_{\vartheta=1}^{101} E(c_r) \\ &= 101 \cdot \tfrac{3}{4} \\ &= 75,75 \\ &\approx 76 \end{aligned}$$

Im Mittel wird also jedes Register 75,75 mal getaktet. Wie nützlich dieser Erwartungswert für die Rekonstruktion von $S(0)$ ist, hängt von der Standardabweichung

[2] Auch hier müßte es streng genommen statt c_r stets $c_r(\vartheta)$ heißen. Da das ϑ jedoch keinen Einfluß auf die Rechnungen hat, wurde der besseren Lesbarkeit halber darauf verzichtet.

für Θ_r ab:

$$\begin{aligned} var(\Theta_r) &= \sum_{\vartheta=1}^{101} E((c_r - E(c_r)^2) \\ &= 101 \cdot E((c_r - \tfrac{3}{4})^2) \\ &= 101 \cdot (\tfrac{3}{4} \cdot (1 - \tfrac{3}{4})^2 + \tfrac{1}{4} \cdot (0 - \tfrac{3}{4})^2 \\ &= 101 \cdot (\tfrac{3}{64} + \tfrac{9}{64}) \\ &= 101 \cdot \tfrac{12}{64} \\ &= \tfrac{303}{16} \\ &= 18,9375 \end{aligned}$$

und somit:

$$\begin{aligned} \sigma(\Theta_r) &= \sqrt{var(\Theta_r)} \\ &= \sqrt{\tfrac{303}{16}} \\ &= \tfrac{1}{4} \cdot \sqrt{303} \\ &\approx 4,35 \end{aligned}$$

Diese Standardabweichung ist erstaunlich niedrig. Dies ist ein Hinweis darauf, daß in den meisten Fällen das korrekte Θ_r in der unmittelbaren Umgebung von 76 zu suchen ist.

8.3.2 Vorgehensweise

Es macht also Sinn, bei der Suche nach Kandidaten für den Anfangszustand $S(0)$ zunächst diejenigen Taktsummen Θ_r zu überprüfen, die in der Nähe von 76 liegen. Der in Abschnitt 8.2 beschriebene Angriff muß dazu nur geringfügig modifiziert werden: Anstatt wie bisher alle denkbaren Θ_r zu durchlaufen, wählt man nun gezielt die wahrscheinlichsten unter ihnen aus.
Diese Vorgehensweise ist dann sinnvoll, wenn man

- entweder die Suche nach dem Auffinden des ersten Kandidaten für $S(0)$ abbricht, oder aber
- eine Grenzwahrscheinlichkeit festlegt, bei deren Unterschreiten ein Θ_r als "überwältigend unwahrscheinlich" angesehen wird.

Während das erstgenannte Abbruchkriterium wenig sinnvoll ist (immerhin gibt es im Mittel mindestens $\frac{8}{5}$ Anfangszustände $S(0)$ pro Zwischenzustand $S(t)$, siehe hierzu Abschnitt 6.3), macht das zweite Kriterium durchaus Sinn.
Bsp.: Die Taktsumme $\Theta_r = 0$ kann nur eintreten, wenn das Register r in keinem der 101 Takte getaktet wurde. Die Wahrscheinlichkeit dafür beträgt $(\frac{1}{4})^{101} = 2^{-202}$ und kann tatsächlich als "überwältigend unwahrscheinlich" bezeichnet werden.

8.3.3 Erfolgswahrscheinlichkeit

Es stellt sich nun also die Frage, welche Werte für Θ_r sinnvollerweise betrachtet werden sollten. Golić behauptet in [Gol97]: *"the number of trials required to find the correct numbers of clocks is with high probability not bigger than about* 10^4*"*. Obwohl aus dieser Quelle nicht unmittelbar ersichtlich ist, wie diese Zahl ermittelt wurde, ist doch anzunehmen, daß Golić vereinfachend das Konfidenzintervall $[E - 2\sigma, E + 2\sigma]$ betrachtet hat. Im folgenden Abschnitt soll genauer untersucht werden, welche Taktsummen sinnvollerweise betrachtet werden sollten und welche tatsächlich als unwahrscheinlich vernachlässigt werden können.

Satz 8.3 *Angenommen, ein Register werde mit jedem Mastertakt mit Wahrscheinlichkeit $p = \frac{3}{4}$ weitergetaktet. Dann beträgt die Wahrscheinlichkeit, daß das Register nach t Takten genau k mal getaktet wurde:*

$$p_{t,k} = \binom{t}{k} \cdot \frac{3^k}{4^t} \tag{8.6}$$

Bew.:

Betrachte k mit $0 \leq k \leq t$. Unter der Annahme, daß die Ergebnisse der einzelnen Mastertakte paarweise voneinander unabhängig sind, ist die Taktsumme k binomialverteilt über [0,t] mit Eintrittswahrscheinlichkeit $p = \frac{3}{4}$. Die Wahrscheinlichkeit $p_{t,k}$ läßt sich somit schreiben als:

$$\begin{aligned} p_{t,k} &= \binom{t}{k} \cdot \left(\frac{3}{4}\right)^k \cdot \left(\frac{1}{4}\right)^{t-k} \\ &= \binom{t}{k} \cdot \frac{3^k \cdot 1^{t-k}}{4^k \cdot 4^{t-k}} \\ &= \binom{t}{k} \cdot \frac{3^k}{4^t} \end{aligned}$$

■

Berechnet man diese Wahrscheinlichkeiten konkret für $t = 101$ und $k = 0 \ldots 101$, so erhält man die in Abbildung 8.1 gegebene Verteilung. Man erkennt sofort, daß die Verteilungsfunktion tatsächlich sehr steil ist und daß k mit hoher Wahrscheinlichkeit Werte annimmt, die in unmittelbarer Nähe des Erwartungswertes liegen.

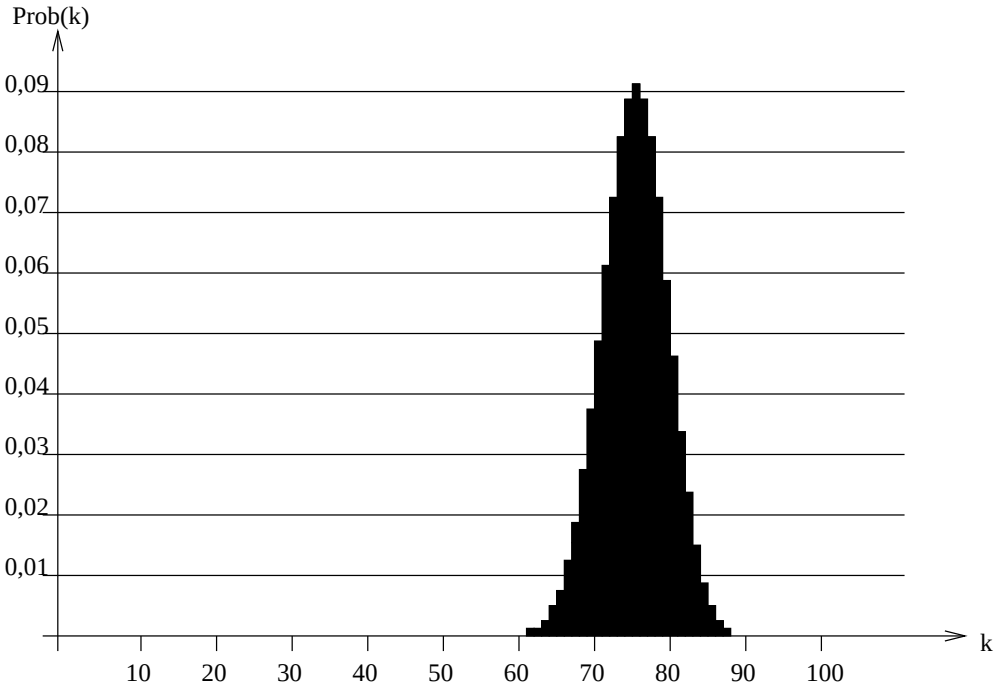

Abbildung 8.1: Wahrscheinlichkeitsverteilung der Taktsumme

Der Kryptanalytiker muß nun festlegen, wie die Menge $H = \{u, \ldots, v\}$ der h häufigsten Werte gewählt werden muß, damit daß Auftreten eines Wertes außerhalb von H als "überwältigend unwahrscheinlich" angesehen werden kann. Dabei ist der

Menge H eine Eintrittswahrscheinlichkeit zugeordnet, die gleich der Summe der Eintrittwahrscheinlichkeiten all ihrer Elemente ist, also:

$$\text{Prob}(k \in H) = \sum_{i=u}^{v} p_{t,i}$$

Mit Hilfe von $\text{Prob}(k \in H)$ kann nun die ursprüngliche Frage nach der Erfolgs- bzw. Mißerfolgswahrscheinlichkeit einer eingeschränkten Suche beantwortet werden.
Gegeben sei eine Menge H und ein konkretes Taktsummentripel $(\Theta_1, \Theta_2, \Theta_3)$. Dann beträgt die **Erfolgswahrscheinlichkeit**, d.h. die Wahrscheinlichkeit, daß alle drei Θ_r in H enthalten sind:

$$\begin{aligned} \text{Prob}^+(H) &:= \text{Prob}(\forall r : \Theta_r \in H) \\ &= \text{Prob}(k \in H)^3 \end{aligned}$$

Analog ist die **Mißerfolgswahrscheinlichkeit** $\text{Prob}^-(H)$ gleich dem Komplement der Erfolgswahrscheinlichkeit, also

$$\text{Prob}^-(H) = 1 - \text{Prob}(k \in H)^3$$

Tabelle 8.1 gibt eine Übersicht über konkrete Intervalle H und die zugehörigen Mißerfolgswahrscheinlichkeiten. Die Intervalle wurden so ausgewählt, daß die unter "Vergleich" aufgeführten Mißerfolgswahrscheinlichkeiten unterschritten werden. Die in der fünften Spalte aufgeführte Komplexität wird in Abschnitt 8.3.4 erläutert und bewiesen.

h	**H**	**Prob$^-(H)$**	**Vergleich**	**Komplexität**
11	[71,81]	$2^{-1,00}$	$\leq 5 \cdot 10^{-1}$	1.331
16	[68,83]	$2^{-2,45}$	$\leq 2 \cdot 10^{-1}$	4.096
19	[67,85]	$2^{-3,57}$	$\leq 1 \cdot 10^{-1}$	6.858
21	[66,86]	$2^{-4,42}$	$\leq 5 \cdot 10^{-2}$	9.241
24	[64,87]	$2^{-5,94}$	$\leq 2 \cdot 10^{-2}$	13.604
26	[63,88]	$2^{-7,00}$	$\leq 1 \cdot 10^{-2}$	17.121
28	[62,89]	$2^{-8,13}$	$\leq 5 \cdot 10^{-3}$	21.136
31	[60,90]	$2^{-10,09}$	$\leq 1 \cdot 10^{-3}$	27.767
36	[57,92]	$2^{-13,72}$	$\leq 1 \cdot 10^{-4}$	41.200
40	[55,94]	$2^{-16,88}$	$\leq 1 \cdot 10^{-5}$	54.861
44	[52,95]	$2^{-20,73}$	$\leq 1 \cdot 10^{-6}$	67.900
47	[50,96]	$2^{-23,73}$	$\leq 1 \cdot 10^{-7}$	79.124

Tabelle 8.1: Ausgewählte Testintervalle und ihre Mißerfolgswahrscheinlichkeiten

8.3.4 Komplexität

Eine kanonische Obergrenze für die Komplexität des eingeschränkten Suchalgorithmus erhält man mit h^3. Zur Berechnung der exakten Komplexität für den allgemeinen Fall benötigen wir die folgende Annahme:

Annahme 8.4 *Wenn u und v so gewählt sind, daß die Menge $H = \{u, \ldots, v\}$ die $v - u + 1$ wahrscheinlichsten Taktsummen k enthält, dann ist die Ungleichung $2v + u \geq 2t$ immer erfüllt.*

Diese Annahme beruht auf der Beobachtung, daß die Gleichheit $2v+u = 2t$ gegeben ist genau dann, wenn u minimal ist, also wenn $u = 0$ und $v = t$ gilt. Wird der Wert

von u erhöht, so sinkt v zunächst weniger stark, so daß sich auch die Summe $2v+u$ kontiniuerlich erhöht, bis sie bei $u \approx \frac{t}{2}$ ihr Maximum erreicht. Steigert man u weiter, so nimmt die Summe $2v+u$ langsam wieder ab, erreicht ihr Minimum aber erst für $u = v \approx \frac{3t}{4}$. Man errechnet leicht, daß die Summe nun noch immer $3 \cdot \frac{3t}{4} = \frac{9t}{4}$ beträgt und somit echt größer ist als $2t$. Ein formaler Nachweis dieser Eigenschaft für den allgemeinen Fall erfordert jedoch erheblichen mathematischen Aufwand[3].

Mit Hilfe dieser Annahme kann nun die exakte Komplexität des Angriffes bewiesen werden. Diese wird gemessen durch die Anzahl möglicher Kombinationen $(\Theta_1, \Theta_2, \Theta_3)$, deren Summe größer oder gleich 202 ist.

Satz 8.5 *Sei $H = \{u, \ldots, v\}$ mit $u \leq v \leq t$ und $h = |H|$ die Menge der h wahrscheinlichsten Taktsummen k. Sei weiterhin Annahme 8.4 erfüllt. Dann ist die Anzahl $\#T_{good}$ möglicher Tripel $(\Theta_1, \Theta_2, \Theta_3)$ mit $u \leq \Theta_r \leq v$ und*

$$2 \cdot t \leq \sum_{r=1}^{3} \Theta_r \leq 3 \cdot t$$

gegeben durch die folgende Gleichung:

$$\#T_{good} = \begin{cases} h^3 & \text{falls } u \geq \frac{2t}{3} \\ h^3 - F(k) & \text{falls } u < \frac{2t}{3} \text{ und } v \geq 2t - 2u - 1 \\ h^3 - F(k) + G(m) & \text{sonst.} \end{cases}$$

Dabei ist

- $F(k) = \frac{1}{6} \cdot (k^3 + 6k^2 + 11k + 6)$ *mit* $k = (2t-1) - 3u$ *und*
- $G(m) = \frac{1}{2} \cdot (m^3 + 3m^2 + 2m)$ *mit* $m = (2t-1) - 2u - v$.

Bew.:

Vorbemerkung: Da $\Theta_r \leq v \leq t$ ist, ist die Teilgleichung

$$\sum_{r=1}^{3} \Theta_r \leq 3 \cdot t$$

für alle obigen Θ_r erfüllt. Man überlegt sich daher zunächst, daß die gesuchte Anzahl der Tripel $\#T_{good}$ gleich der Gesamtzahl h^3 aller möglicher Kombinationen ist weniger die Anzahl Δ der Tripel, die die Bedingung

$$2 \cdot t \leq \sum_{r=1}^{3} \Theta_r$$

nicht erfüllen. Es gilt also, dieses Δ zu ermitteln.

Fall 1: $u \geq \frac{2t}{3}$
In diesem Fall gilt

$$\sum_{r=1}^{3} \Theta_r \geq 3 \cdot u \geq 3 \cdot \frac{2t}{3} = 2t$$

Es gibt somit in diesem Fall keine Tripel mit $\sum_{r=1}^{3} \Theta_r < 2t$. Δ ist somit gleich 0.

[3] Ein denkbarer Ansatz besteht darin, für ein beliebiges festes v zu fordern, daß $\text{Prob}(u) \geq \text{Prob}(v+1)$ gelten muß. Auf diese Weise erhält man in Abhängigkeit von v eine kanonische Untergrenze für u. Leider ist die daraus resultierende Ungleichung nur sehr schwer aufzulösen. Insbesondere führt eine Annäherung eines der Terme durch einen einfacheren mathematischen Ausdruck sehr schnell zu kritischen Informationsverlusten.

Fall 2: $u < \frac{2t}{3}$ und $v \geq (2t-1) - 2u$
Wir nehmen zunächst an, daß v beliebig groß werden kann. Wir betrachten nun einen Wert σ mit $3u \leq \sigma < 2t$ und fragen, auf wieviele Arten dieser Wert aus drei Θ_r konstruiert werden kann.

- $\sigma = 3u \rightarrow 1$ Kombination
- $\sigma = 3u + 1 \rightarrow 3$ Kombinationen
- $\sigma = 3u + 2 \rightarrow 6$ Kombinationen

  $\vdots$
- $\sigma = 3u + \alpha \rightarrow \sum_{i=1}^{\alpha+1} i$ Kombinationen

Also ist Δ gleich der Summe all dieser Kombinationen. Da σ von $3u$ bis $2t-1$ läuft, muß gelten: $0 \leq \alpha \leq (2t-1) - 3u$. Wir setzen $k := (2t-1) - 3u$ und erhalten:

$$\Delta = \sum_{\alpha=0}^{k} \sum_{i=1}^{\alpha+1} i.$$

Diese Gleichung entspricht erneut der Gleichung C.4 aus dem Anhang, wir können also schreiben:

$$\Delta = \frac{1}{6}(k^3 + 6k^2 + 11k + 6)$$

Diese Formel gilt genau dann, wenn alle zur Konstruktion verwendeten Werte kleiner oder gleich v sind. Der größte verwendete Summand Θ_{max} kommt vor in dem Tripel $(u, u, 2t-1-2u)$, dessen Summe genau gleich $2t-1$ ist. Da $v \geq (2t-1) - 2u$ gemäß Annahme, ist die Formel exakt.

Fall 3: $u < \frac{2t}{3}$ und $v < (2t-1) - 2u$
In diesem Fall können gar nicht alle unter Fall 2 mitgezählten Kombinationen mit $\sum_{r=1}^{3} \Theta_r < 2t$ tatsächlich konstruiert werden, da in einigen von ihnen Summanden vorkamen, die größer waren als v. Es muß also gelten:

$$\#T_{good} = h^3 - F(k) + \delta$$

wobei δ gleich der Zahl der nicht kombinierbaren Tripel aus Fall 2 ist.
Wir betrachten nun ein ρ mit $v < \rho \leq 2t - 2u - 1$, das unter Fall 2 "versehentlich" mitgezählt wurde. In wievielen Tripeln ist dieses ρ vorgekommen?

- In Tripeln der Art (ρ, ρ, ρ) kann es nicht vorgekommen sein, da für solche Tripel gilt:

  $$\sum_{r=1}^{3} \Theta_r = 3\rho > 3v > 3 \cdot \frac{3t}{4} = \frac{9}{4}t > 2t$$

  Begr.: Aufgrund der Struktur des Intervalles $[u, v]$ ist v stets größer als der Erwartungswert der Taktsumme, also $v > \frac{3t}{4}$.
- In Tripeln der Art (ρ, ρ, x) mit $x \neq \rho$ kann es ebenfalls nicht vorgekommen sein, da gilt:

  $$\sum_{r=1}^{3} \Theta_r = 2\rho + x > 2v + u \geq 2t$$

  Begr.: gemäß Annahme 8.4 ist $2v + u \geq 2t$ für alle zulässigen Kombinationen aus t, v und u.
- Es verbleiben nur noch Tripel der Art (ρ, x_1, x_2) mit $x_1, x_2 \neq \rho$. Für diese gilt:
  - $\rho = 2t - 2u - 1 \rightarrow 3$ Kombinationen
  - $\rho = 2t - 2u - 2 \rightarrow 9$ Kombinationen
  - $\rho = 2t - 2u - 3 \rightarrow 18$ Kombinationen

    $\vdots$
  - $\rho = 2t - 2u - \beta \rightarrow 3 \cdot \sum_{i=1}^{\beta} i$ Kombinationen.

Das gesuchte δ ergibt sich folglich als Summe all dieser Kombinationen. Da ρ zwischen $v+1$ und $2t-2u-1$ liegt, liegt β zwischen 1 und $2t-2u-v-1$. Wir setzen also $m := (2t-1)-2u-v$ und erhalten:

$$\begin{aligned} \delta &= \sum_{\beta=1}^{m} \left(3 \cdot \sum_{i=1}^{\beta} i \right) \\ &= 3 \cdot \sum_{\beta=1}^{m} \sum_{i=1}^{\beta} i \end{aligned}$$

Mit Formel C.3 aus dem Anhang ergibt sich:

$$\begin{aligned} \delta &= 3 \cdot \left(\frac{1}{6} \cdot (m^3 + 3m^2 + 2m) \right) \\ &= \frac{1}{2} \cdot (m^3 + 3m^2 + 2m) \end{aligned}$$

■

In Tabelle 8.1 sind einige ausgewählte Komplexitäten in Abhängigkeit von u und v gegeben. Eine Testrechnung mit $u = 0$ und $v = t = 101$ bestätigt außerdem das Ergebnis von Satz 8.2, nämlich $\#\mathrm{T}_{good} = 182.104$.

Kapitel 9

Rekonstruktion des Sitzungsschlüssels

Ziel des dritten und letzten Schrittes des Inversionsangriffes auf die Chiffre A5 ist es, aus einem bekannten Anfangszustand $S(0)$, den man mit einem der in Kapitel 8 vorgestellten Verfahren gefunden hat, den Sitzungsschlüssel Kc zu rekonstruieren. Es wird dabei davon ausgegangen, daß es für den Kryptanalytiker kein Problem darstellt, in Besitz der Rahmennummer Nf zu gelangen.

9.1 Vorbemerkungen

In Kapitel 4 wurde beschrieben, wie aus dem Benutzerschlüssel Ki und der Rahmennummer Nf der Anfangszustand $S(0)$ erzeugt wurde:

- Alle Speicherzellen der drei Schieberegister werden mit dem Wert 0 initialisiert. Der resultierende Zustand wird mit $S(-86)$ bezeichnet.
- Die Schieberegister werden 64 mal (ohne Taktkontrolle, aber mit normaler Rückkopplung) getaktet. Dabei wird mit dem k-ten Takt das k-te Bit des Sitzungsschlüssels in den Feedback-Pfad aller drei Schieberegister addiert. Die dabei entstehenden Output-Bits werden verworfen. Den resultierenden Zustand bezeichnen wir mit $S(-22)$.
- Zur Einspeisung der Rahmennummer wird analog vorgegangen: Die Register werden 22 mal getaktet, wobei mit dem k-ten Takt das k-te Bit der Rahmennummer in den Feedback-Pfad addiert wird. Entstehende Output-Bits verfallen wieder ungenutzt, das Ergebnis ist der Anfangszustand $S(0)$.

Bei der Kryptanalyse dieses Vorganges liegt es nun nahe, die obigen Schritte in umgekehrter Reihenfolge zu durchlaufen. Ausgehend von dem Anfangszustand $S(0)$ wird zuerst das Einspeisen der Rahmennummer rückgängig gemacht, also der Zustand $S(-22)$ wiederhergestellt. Danach versucht man, aus den bekannten Zuständen $S(-86)$ und $S(-22)$ der drei Register den Sitzungsschlüssel Kc zu ermitteln.

Bem.: Die Arbeit von Golić, die einigen der vorangegangenen Kapiteln zugrundeliegt, verwendet große Sorgfalt auf diesen letzten Schritt der Kryptanalyse. Bedauerlicherweise ging Golić dabei noch von der in Kapitel 4 angesprochenen Implementierung nach Roe aus, die mittlerweile widerlegt ist. Die Ergebnisse, zu denen Golić in seiner Arbeit kommt, sind auf die Implementierung nach Briceno et al. jedoch nicht mehr übertragbar.

9.2 Rekonstruktion des Zustandes $S(-22)$

Die Rahmennummer Nf läßt sich am einfachsten schrittweise entfernen. Dazu benutzt man die Rekursionsgleichungen, die das Verhalten eines linear rückgekoppelten Schieberegisters bei der Einspeisung der Rahmennummer Nf beschreiben (hier am Beispiel von LFSR_1):

$$s_i(t+1) = \begin{cases} s_{i+1}(t) & \text{für} \quad i = 1, \ldots, 18 \\ s_6(t) + s_3(t) + s_2(t) + s_1(t) + \mathrm{Nf}_{t+1} & \text{für} \quad i = 19 \end{cases} \tag{9.1}$$

Ist nun umgekehrt $S_r(t)$ und Nf_t gegeben und wird $S_r(t-1)$ gesucht, so lassen sich diese Gleichungen umformen wie folgt:

$$s_i(t-1) = \begin{cases} s_{19}(t) + s_5(t) + s_2(t) + s_1(t) + \mathrm{Nf}_t & \text{für} \quad i = 1 \\ s_{i-1}(t) & \text{für} \quad i = 2, \ldots, 19 \end{cases} \tag{9.2}$$

Auf diese Weise lassen sich also schrittweise aus $S(0)$ linear und eindeutig die inneren Zustände $S(-1), \ldots, S(-22)$ rekonstruieren.

9.3 Rekonstruktion des Sitzungsschlüssels Kc

Zur Rekonstruktion des Sitzungsschlüssels Kc ist die obige Vorgehensweise nicht praktikabel. Zwar existiert auch hier eine Rückkopplungsgleichung nach dem Muster von Formel 9.2 (man muß einfach Nf_t durch Kc_t ersetzen), doch enthält diese Gleichung nunmehr zwei Unbekannte, nämlich Kc_t und $s_i(t-1)$. Diese Gleichung ist also zunächst nicht lösbar.

Allerdings sind die Werte $s_i(0)$ bekannt - sie sind alle gleich 0, da das Register zu Beginn der Einspeisung mit dem Nullvektor initialisiert wird. Es besteht also die Möglichkeit, die Gleichung 9.1 iterativ auf sich selbst anzuwenden, wobei man für die $s_i(0)$ den Wert 0 einsetzt. Das Ergebnis ist ein Gleichungssystem der Gestalt

$$s_i(-22) = \alpha_{i,1}\mathrm{Kc}_1 + \alpha_{i,2}\mathrm{Kc}_2 + \ldots + \alpha_{i,64}\mathrm{Kc}_{64}$$

mit $\alpha_{i,j} = 1$, falls Kc_j das Bit $s_i(-22)$ beeinflußt hat, und $\alpha_{i,j} = 0$ andernfalls. Auf diese Weise erhält man je nach Register 19, 22 oder 23 Gleichungen, die aber bis zu 64 Unbekannte enthalten. Das Gleichungssystem ist also für ein einzelnes Register im Normalfall nicht lösbar.

Betrachtet man diesen Typ Gleichung für alle drei Register, so erhält man tatsächlich 64 Gleichungen mit 64 Unbekannten. Dieses Gleichungssystem ist aber dann und nur dann lösbar, wenn es bzw. die zugehörige Koeffizientenmatrix A linear unabhängig ist. Da diese Eigenschaft im allgemeinen Fall für eine 64×64-Matrix kaum nachzuweisen ist, wurde im Rahmen dieser Diplomarbeit die konkrete Matrix A mit Hilfe einer Computersimulation ermittelt. Das dazu verwendete Programm und auch die resultierende Matrix finden sich in Anhang D.

Ein Gleichungssystem mit n Gleichungen und n Unbekannten ist eindeutig lösbar genau dann, wenn der Rang der zugehörige Koeffizientenmatrix ebenfalls gleich n ist (siehe z.B. [Fis86], S. 126). Um diese Bedingung nachzuprüfen, wurde die Matrix A mit Hilfe des Gauß'schen Eliminationsverfahrens (siehe ebenfalls Anhang D) in obere Dreiecksform gebracht. Dabei stellte sich heraus, daß alle Einträge auf der Hauptdiagonalen gleich 1 sind, d.h. daß die Matrix tatsächlich maximalen Rang besitzt. Das Gleichungssystem

$$A \cdot \vec{Kc} = \vec{S}(-22)$$

mit Koeffizientenmatrix A wie im Anhang D beschrieben ist also eindeutig lösbar. Somit kann der Sitzungsschlüssel Kc bei gegebenem inneren Zustand $S(-22)$ durch Anwendung des Gauß'schen Algorithmus (oder eines anderen, effizienteren Verfahrens zur Lösung linearer Gleichungssysteme) eindeutig bestimmt werden.

Kapitel 10

Rekonstruktion des Benutzerschlüssels

Der Sitzungsschlüssel, der in den Kapiteln 5 bis 9 rekonstruiert wurde, kann genutzt werden, um den Inhalt eines verschlüsselten Telefonates zu decodieren. Gelangt ein Angreifer jedoch in den Besitz des geheimen Benutzerschlüssels, so kann er nicht nur sämtliche Gespräche entschlüsseln, die von dem betreffenden Handy aus geführt werden, er kann auch eigene Telefonate auf fremde Kosten führen. Die Rekonstruktion des Benutzerschlüssels hat also noch schwerwiegendere Konsequenzen als die Rekonstruktion des Sitzungsschlüssels.

Das folgende Kapitel beschreibt einen Angriff auf den Algorithmus COMP128, der von Briceno, Goldberg und Wagner in [BGW98b] vorgestellt wurde. Voraussetzung für die Ausführung ist es, daß der Angreifer für begrenzte Zeit (ca. 8 Stunden) Zugriff auf die SIM-Karte des Mobiltelefons hat. Konkret muß er in der Lage sein, etwa 165.000 Authentifikationsanfragen im Challenge-and-Response-Verfahren zu stellen: Er muß Anfragen RAND stellen und die Antworten SRES beobachten können. Dies ist am leichtesten möglich, wenn der Angreifer physikalisch im Besitz der SIM-Karte ist. Ob ein solcher Angriff auch direkt über die Luftschnittstelle möglich ist, ist noch unklar.

10.1 Konstruktion der Kollisionen

Die Sicherheit einer Hash-Funktion f wird auch davon beeinflußt, wie leicht es ist, zwei Eingaben x und y zu finden, die zur gleichen Ausgabe $f(x) = f(y)$ führen. Da der COMP128 eine Ausgabe von 128 Bit erzeugt und somit bis zu 2^{128} verschiedene Ausgaben zuläßt, scheint er auf den ersten Blick kollisionsresistent zu sein. Eine genauere Untersuchung zeigt jedoch, daß es bei freier Wahl von x und y nicht besonders schwierig ist, solche Kollisionen zu finden.

Der COMP128 erhält als Eingabe den (unveränderlich festen) Benutzerschlüssel Ki und die (variable) Herausforderung RAND. Da der variable Wert RAND nur einmal ganz zu Beginn des Algorithmus verwendet wird, muß gelten: Stimmen für zwei verschiedene Herausforderungen RAND_1 und RAND_2 zu irgendeinem Zeitpunkt die "Zwischenergebnisse" des COMP128 überein, so sind auch die Endergebnisse gleich. Anders ausgedrückt: Eine Kollision im Verlauf der Berechnung führt zur gewünschten Ausgabekollision.

Diese Eigenschaft kann man sich bei der Suche nach Kollisionen zunutze machen, da der COMP128 aufgrund der in Abbildung 3.2 auf Seite 13 dargestellten Rundenstruktur die einzelnen Bytes der Eingabe nur sehr langsam miteinander vermischt.

Abbildung 10.1 zeigt noch einmal die Rundenstruktur der ersten beiden Runden zur besseren Veranschaulichung.

10.1.1 Analyse der ersten Hash-Runde

Ideal wäre es natürlich, wenn man die gewünschten Kollisionen bereits am Ende der ersten Runde erzeugen könnte. In der ersten Runde beeinflussen sich nämlich nur die Bytes $x_0[i]$ und $x_0[i+16]$ gegenseitig, wobei das Byte $x_0[i]$ aus dem Benutzerschlüssel Ki stammt und auf dem SIM gespeichert ist. Das Byte $x_0[i+16]$ dagegen kann vom Kryptanalytiker variiert werden in der Hoffnung, zwei solche Bytes zu finden, die am Ende der ersten Runde zum gleichen Tupel $(x_1[i], x_1[i+16])$ führen. Die Durchmischung der Bytes ist in dieser Runde noch besonders gering, so daß auch die Berechnungskomplexität vergleichsweise niedrig wäre.

Untersucht man jedoch die erste Hash-Runde genauer (z.B. mit Hilfe einer einfachen Computersimulation), so stellt sich heraus, daß es sich dabei um eine bijektive Funktion handelt, sobald $x_0[i]$ bekannt und fest ist. Das heißt: Betrachtet man die erste Runde als eine Abbildung

$$\begin{aligned} f: \quad 2^8 \quad &\to 2^8 \times 2^8 \\ x_0[i+16] \quad &\mapsto (x_1[i], x_1[i+16]) \end{aligned}$$

so gibt es bei festem $x_0[i]$ zu jedem Ausgabebyte $x_1[i]$ bzw. $x_1[i+16]$ genau ein Eingabebyte $x_0[i+16]$, von dem es erzeugt wird. Diese Eigenschaft ist keinesfalls selbstverständlich, sie hängt vom Aufbau der S-Boxen ab.

Diese Eigenschaft der ersten Hash-Runde hat drei wichtige Auswirkungen auf die Kryptanalyse:

- Es ist nicht möglich, in der ersten Hash-Runde Kollisionen zu erzeugen, da bei festem $x_0[i]$ zwei unterschiedliche Anfragebytes $x_0[i+16]$ stets auch zu unterschiedlichen Ergebnissen $f(x_0[i+16])$ führen.
- Ist die Eingabe x_0 gleichverteilt über $\{0,1\}^{256}$, so gilt dies auch für die Ausgabe x_1.
- Es ist möglich, zwei Tabellen der Größe $2^8 \times 2^8$ zu berechnen, aus denen für bekanntes $x_0[i+16]$ und $x_1[i]$ bzw. $x_1[i+16]$ das Schlüsselbyte $x_0[i]$ abgelesen werden kann. Mit anderen Worten: Die erste Hash-Runde ist eine Permutation und erfüllt somit keine Hashing-Aufgaben.

10.1.2 Kollisionen nach der zweiten Runde

Im Gegensatz zur ersten Runde ist Runde 2 eine echte Hash-Abbildung. Hier werden nämlich jeweils Tupel der Größe $2^8 \times 2^8$ auf Tupel der Größe $2^7 \times 2^7$ abgebildet. Eine genauere Analyse zeigt, daß auf jedes Abbild-Tupel genau 4 Urbilder-Tupel kommen. Aus Gleichverteilung der Eingabe folgt somit erneut Gleichverteilung der Ausgabe. Es ist also möglich, nach der zweiten Runde Kollisionen zu erzeugen.

Abbildung 10.1 zeigt, daß sich am Ende der zweiten Runde jeweils 4 der ursprünglichen Eingabebytes gegenseitig beeinflußt haben, und zwar die Bytes $x_0[i], x_0[i+8], x_0[i+16]$ und $x_0[i+24]$. Davon können wir die letzteren beiden Bytes variieren, da sie aus RAND stammen. Das Ergebnis ist ein 4-Tupel $x_2[i], x_2[i+8], x_2[i+16]$ und $x_2[i+24]$ der Länge $4 \cdot 7 = 28$ Bit, das für den Kryptanalytiker zwar nicht erkennbar ist, das aber im Falle einer Kollision auch zu einer Kollision der Ausgabe des COMP128 führt.

Nach einer der Formeln zum Geburtstagsproblem beträgt die Wahrscheinlichkeit für mindestens eine Kollision bei n Tests über einer Menge von m möglichen

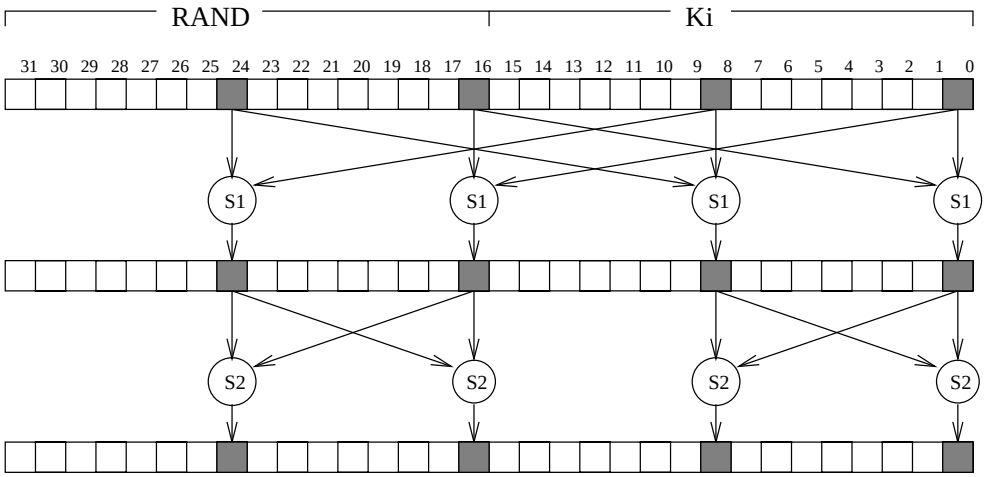

Abbildung 10.1: Die "Narrow Pipe" in den ersten beiden Hash-Runden

Ergebnissen (siehe z.B. [MOV96], S.53):

$$\text{Prob}_{Koll} = 1 - \exp\left(-\frac{n^2}{2m}\right) \tag{10.1}$$

Im vorliegenden Falle sind maximal $n = 2^{16}$ Tests und $m = 2^{28}$ Ergebnisse möglich. Die Wahrscheinlichkeit für eine Kollision beträgt somit:

$$\begin{aligned} \text{Prob}_{Koll} &= 1 - \exp\left(-\frac{2^{32}}{2^{29}}\right) \\ &= 1 - \exp(-2^3) \\ &\approx 0,9997 \end{aligned}$$

Offenbar würde also das Testen aller denkbarer Eingaben mit hoher Wahrscheinlichkeit zu mindestens einer Kollision führen. Eine weitere bekannte Formel zum Geburtstagsproblem (siehe wiederum [MOV96], S.53) liefert uns den ungefähren Erwartungswert für n, ab dem (für große m) im Mittel die erste Kollision aufgetreten ist:

$$\begin{aligned} \text{E}(n) &\approx \sqrt{\frac{\pi \cdot m}{2}} \\ &= \sqrt{\frac{\pi \cdot 2^{28}}{2}} \\ &= 2^{14}\sqrt{\frac{\pi}{2}} \\ &\approx 2^{14,326} \end{aligned} \tag{10.2}$$

Im Mittel findet man also nach $2^{14,326}$ Versuchen eine Kollision, die uns Informationen über 2 Bytes des Benutzerschlüssels liefert. Da der Benutzerschlüssel insgesamt 16 Byte lang ist, müssen wir diesen Vorgang 8 mal wiederholen. Die Gesamtkomplexität des Angriffes liegt also bei $2^3 \cdot 2^{14,326} = 2^{17,326} \approx 164.300$ Anfragen. Briceno et al. geben an, daß sich diese Zahl noch auf etwa 150.000 Anfragen reduzieren läßt, machen aber keine weiteren Angaben über ihre konkrete Vorgehensweise, so daß wir hier mit dem oben ermittelten Wert weiterrechnen wollen. Bei der angegebenen Geschwindigkeit von 6,25 Anfragen pro Sekunde, die ein Kartenleser schaffen kann, dauert der Angriff dann rein rechnerisch etwa 7 Stunden und 20 Minuten.

10.1.3 Beobachtbarkeit einer Kollision

Eine Frage, die noch zu beantworten ist (und die von Briceno et al. völlig vernachlässigt wird), ist die nach der Beobachtbarkeit einer Kollision. Man kann diese Frage wie folgt formulieren: Angenommen, wir beobachten für zwei Eingaben RAND_1, RAND_2 eine Kollision von SRES_1, SRES_2. Muß diese Kollision dann notwendigerweise von einer Kollision in der 2. Runde herrühren?

Wir nehmen dazu an, daß alle Werte für SRES gleich wahrscheinlich sind (was sie bei guter Konstruktion der S-Boxen auch sein sollten). Dann können wir erneut die Formel 10.1 benutzen, um festzustellen, wie wahrscheinlich eine "zufällige" Kollision auf SRES ist, die nichts mit der gesuchten Kollision nach der 2. Runde zu tun haben muß. Dabei ist $m = 2^{32}$ und $n = 2^{14,326}$, und es folgt:

$$\begin{aligned} \mathrm{Prob}_{Koll} &= 1 - \exp\left(-\frac{2^{28,652}}{2^{33}}\right) \\ &= 1 - \exp\left(-\frac{1}{2^{4,348}}\right) \\ &\approx 0,0479 \end{aligned}$$

Es besteht also immerhin eine Wahrscheinlichkeit von annähernd 5%, daß eine festgestellte Kollision zweier Werte SRES_1 und SRES_2 ein Produkt des Zufalls ist, ohne dabei notwendigerweise in Runde 2 des ersten Hashing-Durchlaufes erzeugt worden zu sein. Auf die Frage, wie solche störenden "Zufallskollisionen" erkannt und behandelt werden sollen, wird in [BGW98b] nicht eingegangen.

10.2 Auswertung von Kollisionen

Ist eine Kollision einmal gefunden, so ist die Rekonstruktion der involvierten Schlüsselbytes keine große Schwierigkeit mehr.

Die einfachste Methode, um die beiden Schlüsselbytes $x_0[i]$ und $x_0[i+8]$ zu ermitteln, besteht darin, alle 2^{16} Möglichkeiten auszuprobieren. Dazu kombiniert man jeweils $x_0[i] = 0, \ldots, 255$ und $x_0[i+8] = 0, \ldots, 255$ mit den beiden Tupel $x_0[i+16]$, $x_0[i+24]$, die die Kollision erzeugt haben, und läßt den COMP128 zwei Runden lang laufen. Dann überprüft man das Ergebnis. Stimmen nach Ende der zweiten Runde alle vier Zwischenwerte $x_2[i]$, $x_2[i+8]$, $x_2[i+16]$ und $x_2[i+24]$ überein, so hat man die korrekten Schlüsselbytes gefunden. Wiederholt man den Vorgang für $i = 0, \ldots, 7$, so erhält man den kompletten Schlüssel.

Die maximale Komplexität dieses Vorgehens liegt bei $8 \cdot 2^{16} = 2^{19}$, die mittlere Komplexität somit bei 2^{18} Durchläufen. Obwohl diese Zahl auf den ersten Blick größer scheint als die Komplexität zum Auffinden der Kollisionen, bildet doch letztere den "Bottleneck" des Angriffes: Die Zugriffe mit Hilfe des Kartenlesers zum Auffinden der Kollisionen sind entschieden langsamer als die rechnerinterne Simulation zur Bestimmung der Schlüsselbits. Während erstere mehr als 7 Stunden in Anspruch nimmt, benötigt letztere selbst auf einem langsamen PC nur wenige Sekunden oder schlechtestenfalls Minuten.

Es gibt verschiedene andere Methoden, um die Zahl der Berechnungen in diesem zweiten Schritt zu reduzieren, so z.B.:

- Table-Lookup in einer durch massive Vorberechnungen erzeugten Tabelle, aus der man, wenn man zwei Kollisionstupel $(x_0[i+16], x_0[i+24])$ gefunden hat, einfach die Schlüsselbytes $x_0[i]$ und $x_0[i+8]$ ablesen kann. Diese Tabelle besitzt allerdings 2^{32} Einträge zu je 2 Byte und benötigt daher 8 Gigabyte an Speicherplatz - unnötig viel, wenn man sich den dadurch erzielten minimalen Zeitgewinn vor Augen hält.

- Differentielle Kryptanalyse. Das sehr komplexe Instrumentarium der differentiellen Kryptanalyse läßt sich natürlich auch für vergleichsweise einfache Aufgabenstellungen wie die obige nutzen. Auch Briceno et al. scheinen diese Methode verwendet zu haben, ohne jedoch näher darauf einzugehen: *"[...] with a bit of analysis of the first two rounds (i.e. perform a '2-R attack', in the terminology of differential cryptanalysis)."*

Aufgrund der ohnehin extrem schnellen Berechnung selbst bei vollständiger Suche spielt die Eleganz der Rekonstruktion aber nur eine untergeordnete Rolle. Entscheidend ist, daß der geheime Benutzerschlüssel ohne Schwierigkeiten rekonstruiert werden kann, wenn die entsprechenden Kollisionen gefunden worden sind.

Kapitel 11

Ergebnis und Ausblick

Ziel der Arbeit war es, die im Rahmen der GSM-Norm verwendeten kryptographischen Protokolle und Algorithmen zu beschreiben und zu wissenschaftlich fundierten Aussagen bezüglich ihrer Sicherheit zu gelangen. Das letzte Kapitel soll daher diese Ergebnisse noch einmal kurz zusammenfassen, ihre Bedeutung für die Verläßlichkeit der GSM-Norm erläutern und Verbesserungsmöglichkeiten aufzeigen.

11.1 Sicherheit der Protokolle

Authentifikationsverfahren: Der von uns vorgestellte Angriff auf den Hashing-Algorithmus COMP128 wies eine errechnete und von unabhängigen Stellen getestete Laufzeit von nicht einmal acht Stunden auf. In Anbetracht der Bedeutung des geheimen Benutzerschlüssels im Rahmen des GSM-Standards ist COMP128 somit ein ernstzunehmendes Sicherheitsrisiko. Der Algorithmus sollte in keinem Fall mehr verwendet werden.

Die Frage, ob der von der GSM-Kommission ersatzweise entwickelte Algorithmus COMP128-2 eine größere Sicherheit gewährleistet, muß unbeantwortet bleiben, da der Code erneut geheimgehalten wird. Ein solches Vorgehen legt den Verdacht nahe, daß die Entwickler eine Überprüfung des Algorithmus durch unabhängige Wissenschaftler vermeiden wollen. Die Tatsache, daß es sich scheinbar nur um eine modifizierte Variante des COMP128 handelt, verstärkt noch die Zweifel an der Sicherheit. In der Vergangenheit sind Hashing-Algorithmen, die auf FFT-artigen[1] Butterfly-Strukturen basieren (wie eben COMP128), wiederholt durch Kollisionsangriffe gebrochen worden (siehe dazu beispielsweise die Geschichte des FFT-Hash in [Shr91], [BGG92], [Shr92] und [Vau92]).

Zusammenfassend kann gesagt werden, daß Dienstanbieter, die tatsächlich an der Sicherheit der verwendeten Protokolle interessiert sind, gut beraten wären, eigene Algorithmen zu verwenden, wie dies eigentlich von der GSM-Kommission vorgesehen war (und wie es die meisten deutschen Provider auch tun).

Verschlüsselungsverfahren: Wir haben verschiedene Angriffe zur Rekonstruktion des Sitzungsschlüssels vorgestellt. Dabei benötigt die vollständige Suche (Ciphertext-Only-Angriff) im Mittel 2^{53} Suchdurchläufe, falls die vermutete vorsätzliche Schwächung des Sitzungsschlüssels von 64 auf 54 Bits tatsächlich vorliegt.

Wir haben weiterhin gezeigt, daß es im Falle eines Known-Plaintext-Angriffes verschiedene Varianten eines dreistufigen Inversionsangriffes gibt, dessen Komplexität sich nach der des ersten Schrittes, nämlich der Rekonstruktion eines Zwi-

[1] FFT = Fast Fourier Transformation

schenzustandes $S(t)$ richtet. Für diesen ersten Schritt haben wir einen Algorithmus vorgestellt, der im Mittel zwischen $2^{41,49}$ und $2^{42,04}$ Suchdurchläufe benötigt.

Insbesondere der Known-Plaintext-Angriff ist somit für interessierte Personenkreise durchaus praktikabel. Die vorsätzliche Schwächung des Sitzungsschlüssels und das Veröffentlichungsverbot für die Arbeit von Shepherd ([Shep94]) legen die Vermutung nahe, daß diese Schwäche des Algorithmus A5/1 bei der Entwicklung durchaus beabsichtigt war. Von einem unbeabsichtigten Fehler, der in Kürze behoben wird, ist dagegen nicht auszugehen. Benutzern von Mobiltelefonen kann daher nur geraten werden, sensible Daten privater, geschäftlicher oder politischer Natur niemals einem Mobiltelefon anzuvertrauen.

11.2 Weiterführende Forschung

Kryptanalyse des COMP128: Von Interesse wäre die Frage, ob es möglich ist, den Hash-Algorithmus COMP128 zu invertieren. Wie gezeigt wurde, kann ein Angreifer durch Inversion des A5/1 und Abhören der Authentifikation in den Besitz von Tripeln (Kc, RAND, SRES) gelangen. Wenn es ihm nun gelingt, mit Hilfe einiger solcher Tripel den geheimen Benutzerschlüssel Ki zu rekonstruieren, so wäre jeder denkbare Angriff auf die Sicherheit der GSM-Protokolle sogar über die Luftschnittstelle möglich.

Von besonderer Bedeutung in diesem Zusammenhang ist auch die Frage, ob der Nachfolgealgorithmus COMP128/2 bzw. die von einigen Unternehmen verwendeten proprietären Algorithmen die Fehler des COMP128 vermeiden und einer gründlichen kryptanalytischen Untersuchung standhalten. Solange diese Algorithmen jedoch unveröffentlicht bleiben, ist eine Beantwortung dieser Frage nicht möglich.

Verbesserung der Kryptanalyse des A5: Obwohl die bekannten Algorithmen zur Kryptanalyse hinreichend schnell sind, um eine Rekonstruktion des Sitzungsschlüssels A5 zu ermöglich, ist es dennoch denkbar, daß schnellere Algorithmen existieren, die eine noch nicht entdeckte Schwäche der Chiffre ausnutzen.

So soll im Anschluß an diese Arbeit ein Angriff auf den A5 untersucht werden, der in seinem Aufbau dem vorgestellten Backtracking-Algorithmus sehr ähnlich ist. Allerdings sollen bei diesem Angriff nicht die Taktkontrollbits geraten werden, sondern lediglich das Verhalten der Taktkontrolle. Das Verhalten der Taktkontrolle läßt sich durch zwei lineare Gleichungen beschreiben, so daß man pro geratenem Takt genau drei lineare Gleichungen erhält (zwei über die Taktkontrolle und eine über den Schlüsselstrom). Während der in Abschnitt 7.5.2 beschriebene Baum im Mittel 5 Verzweigungen und $\frac{13}{4}$ lineare Gleichungen pro Blatt aufweist, sind es bei dem neuen Algorithmus exakt 4 Verzweigungen und 3 Gleichungen. Ob das neue Verfahren effizienter ist als die bekannte Vorgehensweise, hängt wiederum davon ab, ab welchem Punkt und mit welcher Häufigkeit die gefundenen Gleichungen linear abhängig werden. Es ist anzunehmen, daß die Komplexität nahe bei der des Angriffes aus dieser Arbeit liegt, genauere Aussagen können aber nur nach einer sorgfältigen Untersuchung getroffen werden.

Denkbar ist natürlich auch, daß noch gänzlich andere Angriffe auf den A5/1 praktikabel sind. Das Verhalten des Algorithmus ist noch immer nicht völlig verstanden. So stellte Chambers im Rahmen seiner Untersuchung "On Random Mappings and Random Permutations" (siehe [Cha94]) fest, daß die Periode des Schlüsselstromgenerators mit den von Roe vorgegebenen Feedbackpolynomen eine Periode von im Mittel lediglich etwa $\frac{4}{3} \cdot 23$ besaß - überraschend wenig also in Anbetracht von 2^{64} möglichen inneren Zuständen. Zwar kann diese Eigenschaft aufgrund der geringen Zahl tatsächlich verwendeter Schlüsselstrombits kryptographisch nicht ausgenutzt werden, doch scheint sie anzudeuten, daß das Verhalten der Chiffre nicht

in allen Punkten den Erwartungen entspricht. Die Aufdeckung weiterer statistischer Besonderheiten könnte somit sehr wohl zur Entdeckung neuer Verfahren zur Kryptanalyse führen.

Verbesserung des A5: Unabhängig von der Frage, ob die Entwickler des Algorithmus A5/1 überhaupt an einer Stärkung des Verschlüsselungsverfahrens interessiert sind, kann man sich natürlich fragen, wie eine solche Stärkung zu erzielen wäre. Die folgenden Ansätze dazu sind denkbar:

- Die Entropie des Sitzungsschlüssels muß wieder auf 64 Bit erhöht werden, da eine Entropie von 54 Bit für eine einfache (leicht in Hardware zu implementierende) Stromchiffre zu gering ist.

- Erhöht man die Schlüssellänge gar auf über 64 Bit, läßt jedoch Aufbau und Länge der Register gleich, so wird der dritte Schritt der Kryptanalyse erheblich verkompliziert, da ein Gleichungssystem mit 64 Gleichungen, aber mehr als 64 Unbekannten nicht mehr lösbar ist. Selbst das Rekonstruieren mehrerer Anfangszustände $S(0)$ würde das Problem dann nicht lösen, da das Gleichungssystem, das durch die Matrix A gegeben ist, stets gleich bliebe und man somit nie mehr als 64 linear unabhängige Gleichungen erhielte.

- Die Effizienz des im ersten Schritt der Kryptanalyse vorgestellten Verfahrens basiert darauf, daß dem Angreifer eine Reihe von unmittelbar aufeinanderfolgenden Bits des Generatorstromes zur Verfügung stehen (üblicherweise die Bits $101, \ldots, 214$). Verwendet man stattdessen voneinander getrennte Bits des Generatorstromes als Schlüsselstrom (z.B. die Bits $1, 4, 5, 7, 10, \ldots$), so ist der beschriebene Angriff nicht mehr praktikabel.

  Eine Möglichkeit, eine solche Auswahl an Bits zu treffen, ohne die Entropie des Sitzungsschlüssels erhöhen zu müssen, bestünde darin, den Generatorstrom als Eingabe eines sogenannten *Self-Shrinking Generators* verwendet (siehe hierzu [Mei94]). Ist der so erzeugte Schlüsselstrom kurz (z.B. 114 Bit wie bisher), so bleiben auch die bisher bekannten probabilistischen Verfahren zur Kryptanalyse des Self-Shrinking Generators wirkungslos. Auf eine genauere Analyse eines solchen modifizierten Generators A5 soll aber an dieser Stelle verzichtet werden.

Anhang A

Implementierung des Algorithmus COMP128

```
/* An implementation of the GSM A3A8 algorithm.
 * (Specifically, COMP128.)
 */

/* Copyright 1998, Marc Briceno, Ian Goldberg, and David Wagner.
 * All rights reserved.
 */

/*
 * For expository purposes only.  Coded in C merely because C is
 * a much more precise, concise form of expression for these
 * purposes. See Judge Patel if you have any problems with this...
 * Of course, it's only authentication, so it should be exportable
 * for the usual boring reasons.
 */

typedef unsigned char Byte;

#include <stdio.h>
/* #define TEST */

/*
 * rand[0..15]: the challenge from the base station
 * key[0..15]: the SIM's A3/A8 long-term key Ki
 * simoutput[0..11]: what you'd get back if you fed rand and key
 * to a real SIM.
 *
 *   The GSM spec states that simoutput[0..3] is SRES,
 *   and simoutput[4..11] is Kc (the A5 session key).
 *   (See GSM 11.11, Section 8.16.  See also the leaked document
 *   referenced below.)
 *   Note that Kc is bits 74..127 of the COMP128 output, followed
 *   by 10 zeros.
 *   In other words, A5 is keyed with only 54 bits of entropy. This
 *   represents a deliberate weakening of the key used for voice
 *   privacy by a factor of over 1000.
```

```
 *
 * Verified with a Pacific Bell Schlumberger SIM.
 * Your mileage may vary.
 *
 * Marc Briceno <marc@scard.org>,
 * Ian Goldberg <iang@cs.berkeley.edu>,
 * and David Wagner <daw@cs.berkeley.edu>
 */

void A3A8(/* in */ Byte rand[16], /* in */ Byte key[16],
/* out */ Byte simoutput[12]);

/* The compression tables. */
static const Byte table_0[512] = {
    102,177,186,162,  2,156,112, 75, 55, 25,  8, 12,251,193,246,188,
    109,213,151, 53, 42, 79,191,115,233,242,164,223,209,148,108,161,
    252, 37,244, 47, 64,211,  6,237,185,160,139,113, 76,138, 59, 70,
     67, 26, 13,157, 63,179,221, 30,214, 36,166, 69,152,124,207,116,
    247,194, 41, 84, 71,  1, 49, 14, 95, 35,169, 21, 96, 78,215,225,
    182,243, 28, 92,201,118,  4, 74,248,128, 17, 11,146,132,245, 48,
    149, 90,120, 39, 87,230,106,232,175, 19,126,190,202,141,137,176,
    250, 27,101, 40,219,227, 58, 20, 51,178, 98,216,140, 22, 32,121,
     61,103,203, 72, 29,110, 85,212,180,204,150,183, 15, 66,172,196,
     56,197,158,  0,100, 45,153,  7,144,222,163,167, 60,135,210,231,
    174,165, 38,249,224, 34,220,229,217,208,241, 68,206,189,125,255,
    239, 54,168, 89,123,122, 73,145,117,234,143, 99,129,200,192, 82,
    104,170,136,235, 93, 81,205,173,236, 94,105, 52, 46,228,198,  5,
     57,254, 97,155,142,133,199,171,187, 50, 65,181,127,107,147,226,
    184,218,131, 33, 77, 86, 31, 44, 88, 62,238, 18, 24, 43,154, 23,
     80,159,134,111,  9,114,  3, 91, 16,130, 83, 10,195,240,253,119,
    177,102,162,186,156,  2, 75,112, 25, 55, 12,  8,193,251,188,246,
    213,109, 53,151, 79, 42,115,191,242,233,223,164,148,209,161,108,
     37,252, 47,244,211, 64,237,  6,160,185,113,139,138, 76, 70, 59,
     26, 67,157, 13,179, 63, 30,221, 36,214, 69,166,124,152,116,207,
    194,247, 84, 41,  1, 71, 14, 49, 35, 95, 21,169, 78, 96,225,215,
    243,182, 92, 28,118,201, 74,  4,128,248, 11, 17,132,146, 48,245,
     90,149, 39,120,230, 87,232,106, 19,175,190,126,141,202,176,137,
     27,250, 40,101,227,219, 20, 58,178, 51,216, 98, 22,140,121, 32,
    103, 61, 72,203,110, 29,212, 85,204,180,183,150, 66, 15,196,172,
    197, 56,  0,158, 45,100,  7,153,222,144,167,163,135, 60,231,210,
    165,174,249, 38, 34,224,229,220,208,217, 68,241,189,206,255,125,
     54,239, 89,168,122,123,145, 73,234,117, 99,143,200,129, 82,192,
    170,104,235,136, 81, 93,173,205, 94,236, 52,105,228, 46,  5,198,
    254, 57,155, 97,133,142,171,199, 50,187,181, 65,107,127,226,147,
    218,184, 33,131, 86, 77, 44, 31, 62, 88, 18,238, 43, 24, 23,154,
    159, 80,111,134,114,  9, 91,  3,130, 16, 10, 83,240,195,119,253
    }, table_1[256] = {
     19, 11, 80,114, 43,  1, 69, 94, 39, 18,127,117, 97,  3, 85, 43,
     27,124, 70, 83, 47, 71, 63, 10, 47, 89, 79,  4, 14, 59, 11,  5,
     35,107,103, 68, 21, 86, 36, 91, 85,126, 32, 50,109, 94,120,  6,
     53, 79, 28, 45, 99, 95, 41, 34, 88, 68, 93, 55,110,125,105, 20,
     90, 80, 76, 96, 23, 60, 89, 64,121, 56, 14, 74,101,  8, 19, 78,
     76, 66,104, 46,111, 50, 32,  3, 39,  0, 58, 25, 92, 22, 18, 51,
     57, 65,119,116, 22,109,  7, 86, 59, 93, 62,110, 78, 99, 77, 67,
```

```
    12,113, 87, 98,102,  5, 88, 33, 38, 56, 23,  8, 75, 45, 13, 75,
    95, 63, 28, 49,123,120, 20,112, 44, 30, 15, 98,106,  2,103, 29,
    82,107, 42,124, 24, 30, 41, 16,108,100,117, 40, 73, 40,  7,114,
    82,115, 36,112, 12,102,100, 84, 92, 48, 72, 97,  9, 54, 55, 74,
   113,123, 17, 26, 53, 58,  4,  9, 69,122, 21,118, 42, 60, 27, 73,
   118,125, 34, 15, 65,115, 84, 64, 62, 81, 70,  1, 24,111,121, 83,
   104, 81, 49,127, 48,105, 31, 10,  6, 91, 87, 37, 16, 54,116,126,
    31, 38, 13,  0, 72,106, 77, 61, 26, 67, 46, 29, 96, 37, 61, 52,
   101, 17, 44,108, 71, 52, 66, 57, 33, 51, 25, 90,  2,119,122, 35
   }, table_2[128] = {
    52, 50, 44,  6, 21, 49, 41, 59, 39, 51, 25, 32, 51, 47, 52, 43,
    37,  4, 40, 34, 61, 12, 28,  4, 58, 23,  8, 15, 12, 22,  9, 18,
    55, 10, 33, 35, 50,  1, 43,  3, 57, 13, 62, 14,  7, 42, 44, 59,
    62, 57, 27,  6,  8, 31, 26, 54, 41, 22, 45, 20, 39,  3, 16, 56,
    48,  2, 21, 28, 36, 42, 60, 33, 34, 18,  0, 11, 24, 10, 17, 61,
    29, 14, 45, 26, 55, 46, 11, 17, 54, 46,  9, 24, 30, 60, 32,  0,
    20, 38,  2, 30, 58, 35,  1, 16, 56, 40, 23, 48, 13, 19, 19, 27,
    31, 53, 47, 38, 63, 15, 49,  5, 37, 53, 25, 36, 63, 29,  5,  7
   }, table_3[64] = {
     1,  5, 29,  6, 25,  1, 18, 23, 17, 19,  0,  9, 24, 25,  6, 31,
    28, 20, 24, 30,  4, 27,  3, 13, 15, 16, 14, 18,  4,  3,  8,  9,
    20,  0, 12, 26, 21,  8, 28,  2, 29,  2, 15,  7, 11, 22, 14, 10,
    17, 21, 12, 30, 26, 27, 16, 31, 11,  7, 13, 23, 10,  5, 22, 19
   }, table_4[32] = {
    15, 12, 10,  4,  1, 14, 11,  7,  5,  0, 14,  7,  1,  2, 13,  8,
    10,  3,  4,  9,  6,  0,  3,  2,  5,  6,  8,  9, 11, 13, 15, 12
   }, *table[5] = { table_0, table_1, table_2, table_3, table_4 };

/*
 * This code derived from a leaked document from the GSM standards.
 * Some missing pieces were filled in by reverse-engineering a
 * working SIM. We have verified that this is the correct COMP128
 * algorithm.
 *
 * The first page of the document identifies it as
 *  _Technical Information: GSM System Security Study_.
 *  10-1617-01, 10th June 1988.
 * The bottom of the title page is marked
 *  Racal Research Ltd.
 *  Worton Drive, Worton Grange Industrial Estate,
 *  Reading, Berks. RG2 0SB, England.
 *  Telephone: Reading (0734) 868601   Telex: 847152
 * The relevant bits are in Part I, Section 20 (pages 66--67).
 * Enjoy!
 *
 * Note: There are three typos in the spec (discovered by
 * reverse-engineering).
 * First, "z = (2 * x[n] + x[n]) mod 2^(9-j)" should clearly read
 * "z = (2 * x[m] + x[n]) mod 2^(9-j)".
 * Second, the "k" loop in the "Form bits from bytes" section is
 * severely botched: the k index should run only from 0 to 3, and
 * clearly the range on "the (8-k)th bit of byte j" is also off
 * (should be 0..7, not 1..8, to be consistent with the subsequent
 * section).
```

```
 * Third, SRES is taken from the first 8 nibbles of x[], not the
 * last 8 as claimed in the document.  (And the document doesn't
 * specify how Kc is derived, but that was also easily discovered
 * with reverse engineering.)
 * All of these typos have been corrected in the following code.
 */

void A3A8(/* in */ Byte rand[16], /* in */ Byte key[16],
   /* out */ Byte simoutput[12])
{
   Byte x[32], bit[128];
   int i, j, k, l, m, n, y, z, next_bit;

   /* ( Load RAND into last 16 bytes of input ) */
   for (i=16; i<32; i++)
      x[i] = rand[i-16];

   /* ( Loop eight times ) */
   for (i=1; i<9; i++) {
      /* ( Load key into first 16 bytes of input ) */
      for (j=0; j<16; j++)
         x[j] = key[j];
      /* ( Perform substitutions ) */
      for (j=0; j<5; j++)
         for (k=0; k<(1<<j); k++)
            for (l=0; l<(1<<(4-j)); l++) {
               m = l + k*(1<<(5-j));
               n = m + (1<<(4-j));
               y = (x[m]+2*x[n]) % (1<<(9-j));
               z = (2*x[m]+x[n]) % (1<<(9-j));
               x[m] = table[j][y];
               x[n] = table[j][z];
            }
      /* ( Form bits from bytes ) */
      for (j=0; j<32; j++)
         for (k=0; k<4; k++)
            bit[4*j+k] = (x[j]>>(3-k)) & 1;
      /* ( Permutation but not on the last loop ) */
      if (i < 8)
         for (j=0; j<16; j++) {
            x[j+16] = 0;
            for (k=0; k<8; k++) {
               next_bit = ((8*j + k)*17) % 128;
               x[j+16] |= bit[next_bit] << (7-k);
            }
         }
   }

   /*
    * ( At this stage the vector x[] consists of 32 nibbles.
    *   The first 8 of these are taken as the output SRES. )
    */

   /* The remainder of the code is not given explicitly in the
```

```
    * standard, but was derived by reverse-engineering.
    */

   for (i=0; i<4; i++)
      simoutput[i] = (x[2*i]<<4) | x[2*i+1];
   for (i=0; i<6; i++)
      simoutput[4+i] = (x[2*i+18]<<6) | (x[2*i+18+1]<<2)
                                      | (x[2*i+18+2]>>2);
   simoutput[4+6] = (x[2*6+18]<<6) | (x[2*6+18+1]<<2);
   simoutput[4+7] = 0;
}


#ifdef TEST
int hextoint(char x)
{
   x = toupper(x);
   if (x >= 'A' && x <= 'F')
      return x-'A'+10;
   else if (x >= '0' && x <= '9')
      return x-'0';
   fprintf(stderr, "bad input.\n");
   exit(1);
}

int main(int argc, char **argv)
{
   Byte rand[16], key [16], simoutput[12];
   int i;

   if (argc != 3 || strlen(argv[1]) != 34 || strlen(argv[2]) != 34
                 || strncmp(argv[1], "0x", 2) != 0
                 || strncmp(argv[2], "0x", 2) != 0) {
      fprintf(stderr, "Usage: %s 0x<key> 0x<rand>\n", argv[0]);
      exit(1);
   }

   for (i=0; i<16; i++)
      key[i] = (hextoint(argv[1][2*i+2])<<4)
             | hextoint(argv[1][2*i+3]);
   for (i=0; i<16; i++)
      rand[i] = (hextoint(argv[2][2*i+2])<<4)
              | hextoint(argv[2][2*i+3]);
   A3A8(key, rand, simoutput);
   printf("simoutput: ");
   for (i=0; i<12; i++)
      printf("%02X", simoutput[i]);
   printf("\n");
   return 0;
}
#endif
```

Anhang B

Implementierung des Algorithmus A5/1

```
/*
 * A pedagogical implementation of A5/1.
 * Performance will be terrible, but that's not the point.
 * This code is export-controlled by US law.
 * Copyright 1998: Lucky Green, David Wagner, and Ian Goldberg.
 * Typed by Georg Schneider, Universitaet Mannheim, Germany.
 */

#include <stdio.h>

/* Masks for the three shift registers */
#define R1MASK  0x07FFFF /* 19 bits, numbered 0..18 */
#define R2MASK  0x3FFFFF /* 22 bits, numbered 0..21 */
#define R3MASK  0x7FFFFF /* 23 bits, numbered 0..22 */

/* Middle bit of each of the three shift registers, for
 * clock control */
#define R1MID   0x000100 /* bit 8 */
#define R2MID   0x000400 /* bit 10 */
#define R3MID   0x000400 /* bit 10 */

/* Feedback taps, for clocking the shift registers.
 * These correspond to the primitive polynomials
 * x^19 + x^5 + x^2 + x + 1, x^22 + x + 1,
 * and x^23 + x^15 + x^2 + x + 1. */
#define R1TAPS  0x072000 /* bits 18,17,16,13 */
#define R2TAPS  0x300000 /* bits 21,20 */
#define R3TAPS  0x700080 /* bits 22,21,20,7 */

/* Output taps for output generation */
#define R1OUT   0x040000 /* bit 18 (the high bit) */
#define R2OUT   0x200000 /* bit 21 (the high bit) */
#define R3OUT   0x400000 /* bit 22 (the high bit) */

typedef unsigned char byte;
typedef unsigned long word;
```

```
typedef word bit;

/* Calculate the parity of a 32-bit word,
 * i.e. the sum of its bits modulo 2 */
bit parity(word x) {
   x ^= x>>16;
   x ^= x>>8;
   x ^= x>>4;
   x ^= x>>2;
   x ^= x>>1;
   return x&1;
}

/* Clock one shift register */
word clockone(word reg, word mask, word taps) {
   word t = reg & taps;
   reg = (reg << 1) & mask;
   reg |= parity(t);
   return reg;
}

/* The three shift registers. They're in global variables
 * to make the code easier to understand.
 * A better implementation would not use global variables. */
word R1, R2, R3;

/* Look at the middle bits of R1,R2,R3, take a vote, and
 * return the majority value of those 3 bits. */
bit majority() {
   int sum;
   sum = parity(R1&R1MID) + parity(R2&R2MID) + parity(R3&R3MID);
   if (sum >= 2)
      return 1;
   else
      return 0;
}

/* Clock two or three of R1,R2,R3, with clock control
 * according to their middle bits.
 * Specifically, we clock Ri whenever Ri's middle bit
 * agrees with the majority value of the three middle bits. */
void clock() {
   bit maj = majority();
   if (((R1&R1MID)!=0) == maj)
      R1 = clockone(R1, R1MASK, R1TAPS);
   if (((R2&R2MID)!=0) == maj)
      R2 = clockone(R2, R2MASK, R2TAPS);
   if (((R3&R3MID)!=0) == maj)
      R3 = clockone(R3, R3MASK, R3TAPS);
}

/* Clock all three of R1,R2,R3, ignoring their middle bits.
 * This is only used for key setup. */
void clockallthree() {
```

```
   R1 = clockone(R1, R1MASK, R1TAPS);
   R2 = clockone(R2, R2MASK, R2TAPS);
   R3 = clockone(R3, R3MASK, R3TAPS);
}

/* Generate an output bit from the current state.
 * You grab a bit from each register via the output generation
 * taps; then you XOR the resulting three bits. */
bit getbit() {
   return parity(R1&R1OUT)^parity(R2&R2OUT)^parity(R3&R3OUT);
}

/* Do the A5/1 key setup. This routine accepts a 64-bit key and
 * a 22-bit frame number. */
void keysetup(byte key[8], word frame) {
   int i;
   bit keybit, framebit;

   /* Zero out the shift registers. */
   R1 = R2 = R3 = 0;

   /* Load the key into the shift registers,
    * LSB of first byte of key array first,
    * clocking each register once for every
    * key bit loaded.  (The usual clock
    * control rule is temporarily disabled.) */
   for (i=0; i<64; i++) {
      clockallthree(); /* always clock */
      keybit = (key[i/8] >> (i&7)) & 1;
        /* The i-th bit of the key */
      R1 ^= keybit; R2 ^= keybit; R3 ^= keybit;
   }

   /* Load the frame number into the shift
    * registers, LSB first,
    * clocking each register once for every
    * key bit loaded.  (The usual clock
    * control rule is still disabled.) */
   for (i=0; i<22; i++) {
      clockallthree(); /* always clock */
      framebit = (frame >> i) & 1;
        /* The i-th bit of the frame # */
      R1 ^= framebit; R2 ^= framebit; R3 ^= framebit;
   }

   /* Run the shift registers for 100 clocks
    * to mix the keying material and frame number
    * together with output generation disabled,
    * so that there is sufficient avalanche.
    * We re-enable the majority-based clock control
    * rule from now on. */
   for (i=0; i<100; i++) {
      clock();
   }
```

```
   /* Now the key is properly set up. */
}

/* Generate output. We generate 228 bits of
 * keystream output. The first 114 bits is for
 * the A->B frame; the next 114 bits is for the
 * B->A frame. You allocate a 15-byte buffer
 * for each direction, and this function fills
 * it in. */
void run(byte AtoBkeystream[], byte BtoAkeystream[]) {
   int i;

   /* Zero out the output buffers. */
   for (i=0; i<=113/8; i++)
      AtoBkeystream[i] = BtoAkeystream[i] = 0;

   /* Generate 114 bits of keystream for the
    * A->B direction.  Store it, MSB first. */
   for (i=0; i<114; i++) {
      clock();
      AtoBkeystream[i/8] |= getbit() << (7-(i&7));
   }

   /* Generate 114 bits of keystream for the
    * B->A direction.  Store it, MSB first. */
   for (i=0; i<114; i++) {
      clock();
      BtoAkeystream[i/8] |= getbit() << (7-(i&7));
   }
}

/* Test the code by comparing it against
 * a known-good test vector. */
void test() {
   byte key[8] = {0x12, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF};
   word frame = 0x134;
   byte goodAtoB[15] =
     { 0x53, 0x4E, 0xAA, 0x58, 0x2F, 0xE8, 0x15,
       0x1A, 0xB6, 0xE1, 0x85, 0x5A, 0x72, 0x8C, 0x00 };
   byte goodBtoA[15] =
     { 0x24, 0xFD, 0x35, 0xA3, 0x5D, 0x5F, 0xB6,
       0x52, 0x6D, 0x32, 0xF9, 0x06, 0xDF, 0x1A, 0xC0 };
   byte AtoB[15], BtoA[15];
   int i, failed=0;

   keysetup(key, frame);
   run(AtoB, BtoA);

   /* Compare against the test vector. */
   for (i=0; i<15; i++)
      if (AtoB[i] != goodAtoB[i])
         failed = 1;
   for (i=0; i<15; i++)
```

```
      if (BtoA[i] != goodBtoA[i])
         failed = 1;

   /* Print some debugging output. */
   printf("key: 0x");
   for (i=0; i<8; i++)
      printf("%02X", key[i]);
   printf("\n");
   printf("frame number: 0x%06X\n", (unsigned int)frame);
   printf(" A->B: 0x");
   for (i=0; i<15; i++)
      printf("%02X", goodAtoB[i]);
   printf(" B->A: 0x");
   for (i=0; i<15; i++)
      printf("%02X", goodBtoA[i]);
   printf("\n");
   printf("observed output:\n");
   printf(" A->B: 0x");
   for(i=0; i<15; i++)
      printf("%02X", AtoB[i]);
   printf(" B->A: 0x");
   for(i=0; i<15; i++)
      printf("%02X", BtoA[i]);
   printf("\n");

   if (!failed) {
      printf("Self-check succeeded: everything looks ok.\n");
      return;
   } else {
      /* Problems!  The test vectors didn't compare */
      printf("I don't know why this broke; contact the authors.\n");
      exit(1);
   }
}

int main(void) {
   test();
   return 0;
}
```

Anhang C

Herleitung der verwendeten Summenformeln

In Kapitel 8 der Diplomarbeit werden im Zusammenhang mit kombinatorischen Überlegungen wiederholt Summenformeln verwendet. Manche dieser Formeln stehen in jedem besseren Mathematikbuch, andere müssen explizit hergeleitet werden. Um die Beweise in der Diplomarbeit nicht mit diesen Herleitungen zu überfrachten, wurden sie an diese Stelle im Anhang ausgegliedert.

Die folgenden beiden Summenformeln sind allgemein bekannt und können z.B. in [For83] nachgelesen werden:

$$\sum_{i=1}^{m} i = \frac{m \cdot (m+1)}{2} \tag{C.1}$$

und

$$\sum_{i=1}^{m} i^2 = \frac{m \cdot (2m+1) \cdot (m+1)}{6} \tag{C.2}$$

Sie bilden die Grundlage für die nachfolgenden Betrachtungen.

Lemma C.1 *Für ein beliebig festes $n \in \boldsymbol{N}$ gilt:*

$$\sum_{j=1}^{n} \sum_{i=1}^{j} i = \frac{1}{6} \cdot (n^3 + 3n^2 + 2n) \tag{C.3}$$

Bew.:

$$\begin{aligned}
\sum_{j=1}^{n} \sum_{i=1}^{j} i \quad &\overset{C.1}{=} \quad \sum_{j=1}^{n} \frac{j(j+1)}{2} \\
&= \quad \frac{1}{2} \cdot \sum_{j=1}^{n} (j^2 + j) \\
&\overset{C.1+C.2}{=} \quad \frac{1}{2} \cdot \left(\sum_{j=1}^{n} j^2 + \sum_{j=1}^{n} j \right) \\
&= \quad \frac{1}{2} \cdot \left(\frac{n(2n+1)(n+1)}{6} + \frac{n(n+1)}{2} \right) \\
&= \quad \frac{1}{12} \cdot ((2n^3 + 3n^2 + n) + (3n^2 + 3n))
\end{aligned}$$

$$\begin{aligned} &= \frac{1}{12} \cdot (2n^3 + 6n + 2 + 4n) \\ &= \frac{1}{6} \cdot (n^3 + 3n^2 + 2n) \end{aligned}$$

■

Aus dieser Gleichung läßt sich nun leicht die folgende Formel herleiten:

Lemma C.2 *Für ein beliebig festes $n \in \boldsymbol{N}$ gilt:*

$$\sum_{j=1}^{n+1} \sum_{i=1}^{j} i = \frac{1}{6} \cdot (n^3 + 6n^2 + 11n + 6) \tag{C.4}$$

Bew.:

Wir betrachten noch einmal Gleichung C.3

$$\sum_{j=1}^{k} \sum_{i=1}^{j} i = \frac{1}{6} \cdot (k^3 + 3k^2 + 2k)$$

und substituieren auf beiden Seiten $k = n + 1$. Auf diese Weise ergibt sich:

$$\begin{aligned} \sum_{j=1}^{n+1} \sum_{i=1}^{j} i &= \frac{1}{6} \cdot ((n+1)^3 + 3(n+1)^2 + 2(n+1)) \\ &= \frac{1}{6} \cdot ((n^3 + 3n^2 + 3n + 1) + (3n^2 + 6n + 3) + (2n + 2)) \\ &= \frac{1}{6} \cdot (n^3 + 6n^2 + 11n + 6) \end{aligned}$$

■

Anhang D

Untersuchung der Schlüsseleinspeisung auf Umkehrbarkeit

Wie in Kapitel 9 beschrieben, erfolgt die Einspeisung des Sitzungsschlüssels Kc in die Register nach dem folgenden Schema:

- Lade den Wert 0 in alle Registerzellen.
- Takte jedes der drei Register 64 mal mit Rückkopplung, aber ohne Taktkontrolle. Mit jedem Takt t wird das t-te Schlüsselbit in den Feedback-Pfad addiert.

Das Endergebnis ist der innere Zustand $S(-22)$. Das folgende Kapitel soll der Frage nachgehen, ob dieser Einspeisungsvorgang umkehrbar ist. Da der Einspeisungsvorgang offensichtlich linear ist, können wir ihn durch die folgende lineare Gleichung beschreiben:

$$A \cdot \vec{\mathrm{Kc}} = \vec{S}(-22),$$

wobei $\vec{\mathrm{Kc}}$ der Darstellung von Kc als Spaltenvektor entspricht. (siehe dazu ebenfalls Kapitel 9). Der Einspeisungsvorgang ist umkehrbar genau dann, wenn die Matrix A maximalen Rang hat. Um eine entsprechende Aussage treffen zu können, müssen wir sie zunächst explizit berechnen (Abschnitt D.1) und dann das Gleichungssystem auf seine Lösbarkeit untersuchen (Abschnitt D.2)

D.1 Bestimmung der Matrix A

Die Matrix A wurde bestimmt durch eine Computersimulation. Zu diesem Zwecke wurden die drei Register des Generators 64 mal getaktet. Der Simulator führte dabei Buch darüber, welche Registerzelle nach welchem Takt von welchen Schlüsselbits beeinflußt wurde.

Aufgrund der geringen Ansprüche in Bezug auf Rechenzeit und Speicherplatz wurde darauf verzichtet, eine besonders effiziente Implementierung zu finden. Stattdessen wurde der Algorithmus objektorientiert implementiert, um sowohl Funktionsweise als auch Korrektheit besser dokumentieren zu können. Die folgenden Abschnitte enthalten den Programmcode in C++. Auf den Abdruck von Präprozessordirektiven wurde aus Gründen der Übersichtlichkeit verzichtet.

Das zugrundeliegende Objektmodell ist dabei trivial: Es gibt zwei Klassen, `register` und `zelle`. Ein Register besteht dabei aus einer Anzahl von Speicherzellen (konkret: 19, 22 bzw. 23 Zellen). Die Steuerung der Register wiederum obliegt dem Hauptprogramm.

D.1.1 Die Klasse "Zelle"

Jede Instanz der Klasse `zelle` simuliert ein Flip-Flop in einem linearen Feedback-Shift-Register. Allerdings speichert ein solches Objekt im Gegensatz zu gewöhnlichen Flip-Flops nicht einen boole'schen Wert `true` oder `false`. Vielmehr führt die Zelle in einem Feld der Größe 64 Bit Buch darüber, ob es zum aktuellen Zeitpunkt vom Schlüsselbit i beeinflußt worden ist (`inhalt[i]=true`) oder nicht.

Quellcode in C++

```
class Zelle {
  private:  // Datenelemente
    bool inhalt[64];
  public:   // Methoden
    Zelle();               // Konstruktor
    bool giveBit(int);     // gibt Inhalt an Stelle c zurueck
    void showAll();        // schreibe Inhalt an Standardausgabe
    void replaceBy(Zelle); // ersetzt Inhalt der Zelle
    void addCell(Zelle);   // addiert Inhalt einer anderen Zelle
    void addKey(int);      // addiert ein Schluesselbit
};

Zelle::Zelle() { // Konstruktor: Initialisiert das Feld inhalt[64]
  int i;
  for(i=0; i<64; i++) {
    inhalt[i]=0;}
}

bool Zelle::giveBit(int c) { // Gibt inhalt[c] zurueck
  return inhalt[c];
}

void Zelle::showAll() { // Gibt das Feld inhalt[64]
                        // am Bildschirm aus
  int i;
  char p;
  for(i=0; i<64; i++) {
    if(inhalt[i]) p='1';
    else p='-';
    cout << p << " ";}
  cout << "\n";
}

void Zelle::replaceBy(Zelle f) { // Ersetzt Inhalt durch den einer
                                 //anderen Zelle
  int i;
  for(i=0; i<64; i++) {
    inhalt[i] = f.giveBit(i);}
}
```

```
void Zelle::addCell(Zelle f) { // Addiert den Inhalt einer anderen
                               // Zelle
  int i;
  for(i=0; i<64; i++) {
    inhalt[i] = inhalt[i] ^ f.giveBit(i);}
}

void Zelle::addKey(int c) { // Addiert ein einzelnes Schluesselbit
  inhalt[c] = inhalt[c] ^ 1;
}
```

D.1.2 Die Klasse "Register"

Eine Instanz der Klasse `register` simuliert ein linear rückgekoppeltes Schieberegister. Ein solches Register hat die Größe `size` und besteht aus ebensovielen Flip-Flops wie Feedback-Taps. Aus Gründen der Einfachheit und Übersichtlichkeit wurden beide als Felder einer vorher definierten Höchstgröße `MAXSIZE` implementiert.

Quellcode in C++

```
#define MAXSIZE 30

class Register {
  private:  // Datenelemente
    int size;                // Registerlaenge
    Zelle flipflop[MAXSIZE]; // Zelleninhalte
    bool fbtap[MAXSIZE];     // Feedback-Taps
  public:   // Methoden
    Register(int, bool[]);   // Konstruktor
    void toScreen();         // Gibt Inhalt auf Bildschirm aus
    void step();             // Takte einen Schritt weiter
    void inKey(int);         // speise KeyBit i ein
};

Register::Register(int length, bool taps[]) {
          // Konstruktor: Initialisert size und fbtap
          // (zelle initialisiert sich selbst)
  int i;
  size = length;
  for(i=0; i<size; i++)
    fbtap[i] = taps[i];
}

void Register::toScreen() { // Gibt Inhalt am Bildschirm aus
  int i;
  for(i=0; i<size; i++) {
    cout.width(5);
    cout << i << ": ";
    flipflop[i].showAll();}
}

void Register::step() { // Taktet Register um 1 Takt weiter
  Zelle puffer;
```

```
  int i;
  // 1) Bilde Puffer fuer Rueckkopplung
  for(i=0; i<size; i++) {
    if(fbtap[i]==1) puffer.addCell(flipflop[i]);}
  // 2) Schiebe alle Zelleninhalte um 1 Stelle nach rechts
  for(i=0; i<size-1; i++) {
    flipflop[i].replaceBy(flipflop[i+1]);}
  // 3) Ersetze hoechstwertigsten Zelleninhalt durch Puffer
  flipflop[size-1].replaceBy(puffer);
}

void Register::inKey(int n) { // Taktet Schluesselbit ein
  flipflop[size-1].addKey(n);
}
```

D.1.3 Das Hauptprogramm

Wie bei vielen objektorientierten Codes ist das Hauptprogramm recht einfach aufgebaut. Es initialisiert drei Instanzen der Klasse `register`, taktet sie 64 mal und gibt am Ende das Ergebnis am Bildschirm aus.

Quellcode in C++

```
int main() {

  int i;

  // Initialisiere die 3 LFSRs
  bool tap1[19] = {1,1,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0};
  Register reg1(19,tap1);
  bool tap2[22] = {1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
  Register reg2(22,tap2);
  bool tap3[23] = {1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0};
  Register reg3(23,tap3);

  // Versende 64 Taktsignale
  for(i=0; i<64; i++) {
    reg1.step();
    reg1.inKey(i);
    reg2.step();
    reg2.inKey(i);
    reg3.step();
    reg3.inKey(i);
  }

  // Gib Ergebnis an Standardausgabe aus
  reg1.toScreen();
  reg2.toScreen();
  reg3.toScreen();

  return 0;
}
```

D.1.4 Die Matrix A

Tabelle D.1 stellt die Koeffizientenmatrix A dar, die vom obigen Programm als Ausgabe erzeugt wird. Dabei entsprechen die Zeilen der Matrix den jeweiligen Speicherzellen (Zellen $1 \ldots 19$ von Register 1, Zellen $1 \ldots 22$ von Register 2 und Zellen $1 \ldots 23$ von Register 3). Eine '1' in Zeile i und Spalte j bedeutet dabei, daß sich das Schlüsselbit Kc_j auf die Zelle i auswirkt. Der besseren Lesbarkeit halber wurde anstatt einer '0' ein Strich '-' in die Matrix eingetragen.

```
1/01: 1 - - 1 - - - 1 - 1 - 1 - - - - - 1 - - - - - - - - - 1 1 1 - - 1 - - - - - - - - - - - - - - 1 - - - - - - - - - - - - - - - - - -
1/02: 1 1 - - 1 - - - 1 - 1 - 1 - - - - - 1 - - - - - - - - - 1 1 1 - - 1 - - - - - - - - - - - - - - 1 - - - - - - - - - - - - - - - - - -
1/03: 1 1 1 - - 1 - - - 1 - 1 - 1 - - - - - 1 - - - - - - - - - 1 1 1 - - 1 - - - - - - - - - - - - - - 1 - - - - - - - - - - - - - - - - - -
1/04: 1 1 1 1 - - 1 - - - 1 - 1 - 1 - - - - - 1 - - - - - - - - - 1 1 1 - - 1 - - - - - - - - - - - - - - 1 - - - - - - - - - - - - - - - - - -
1/05: - 1 1 1 1 - - 1 - - - 1 - 1 - 1 - - - - - 1 - - - - - - - - - 1 1 1 - - 1 - - - - - - - - - - - - - - 1 - - - - - - - - - - - - - - - - - -
1/06: 1 - 1 1 1 1 - - 1 - - - 1 - 1 - 1 - - - - - 1 - - - - - - - - - 1 1 1 - - 1 - - - - - - - - - - - - - - 1 - - - - - - - - - - - - - - - - - -
1/07: 1 1 - 1 1 1 1 - - 1 - - - 1 - 1 - 1 - - - - - 1 - - - - - - - - - 1 1 1 - - 1 - - - - - - - - - - - - - - 1 - - - - - - - - - - - - - - - - - -
1/08: - 1 1 - 1 1 1 1 - - 1 - - - 1 - 1 - 1 - - - - - 1 - - - - - - - - - 1 1 1 - - 1 - - - - - - - - - - - - - - 1 - - - - - - - - - - - - - - - - - -
1/09: - - 1 1 - 1 1 1 1 - - 1 - - - 1 - 1 - 1 - - - - - 1 - - - - - - - - - 1 1 1 - - 1 - - - - - - - - - - - - - - 1 - - - - - - - - - - - - - - - - - -
1/10: 1 - - 1 1 - 1 1 1 1 - - 1 - - - 1 - 1 - 1 - - - - - 1 - - - - - - - - - 1 1 1 - - 1 - - - - - - - - - - - - - - 1 - - - - - - - - - - - - - - - - - -
1/11: - 1 - - 1 1 - 1 1 1 1 - - 1 - - - 1 - 1 - 1 - - - - - 1 - - - - - - - - - 1 1 1 - - 1 - - - - - - - - - - - - - - 1 - - - - - - - - - - - - - - - - - -
1/12: - - 1 - - 1 1 - 1 1 1 1 - - 1 - - - 1 - 1 - 1 - - - - - 1 - - - - - - - - - 1 1 1 - - 1 - - - - - - - - - - - - - - 1 - - - - - - - - - - - - - - - - - -
1/13: 1 - - 1 - - 1 1 - 1 1 1 1 - - 1 - - - 1 - 1 - 1 - - - - - 1 - - - - - - - - - 1 1 1 - - 1 - - - - - - - - - - - - - - 1 - - - - - - - - - - - - - - - - - -
1/14: - 1 - - 1 - - 1 1 - 1 1 1 1 - - 1 - - - 1 - 1 - 1 - - - - - 1 - - - - - - - - - 1 1 1 - - 1 - - - - - - - - - - - - - - 1 - - - - - - - - - - - - - - - - - -
1/15: - - 1 - - 1 - - 1 1 - 1 1 1 1 - - 1 - - - 1 - 1 - 1 - - - - - 1 - - - - - - - - - 1 1 1 - - 1 - - - - - - - - - - - - - - 1 - - - - - - - - - - - - - - - - - -
1/16: - - - 1 - - 1 - - 1 1 - 1 1 1 1 - - 1 - - - 1 - 1 - 1 - - - - - 1 - - - - - - - - - 1 1 1 - - 1 - - - - - - - - - - - - - - 1 - - - - - - - - - - - - - - - - - -
1/17: - - - - 1 - - 1 - - 1 1 - 1 1 1 1 - - 1 - - - 1 - 1 - 1 - - - - - 1 - - - - - - - - - 1 1 1 - - 1 - - - - - - - - - - - - - - 1 - - - - - - - - - - - - - - - - - -
1/18: - - - - - 1 - - 1 - - 1 1 - 1 1 1 1 - - 1 - - - 1 - 1 - 1 - - - - - 1 - - - - - - - - - 1 1 1 - - 1 - - - - - - - - - - - - - - 1 - - - - - - - - - - - - - - - - - -
1/19: - - - - - - 1 - - 1 - - 1 1 - 1 1 1 1 - - 1 - - - 1 - 1 - 1 - - - - - 1 - - - - - - - - - 1 1 1 - - 1 - - - - - - - - - - - - - - 1
2/01: 1 - - - - - - - - - - - - - - - - - - - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/02: - 1 - - - - - - - - - - - - - - - - - - - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/03: 1 - 1 - - - - - - - - - - - - - - - - - - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/04: - 1 - 1 - - - - - - - - - - - - - - - - - - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/05: - - 1 - 1 - - - - - - - - - - - - - - - - - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/06: - - - 1 - 1 - - - - - - - - - - - - - - - - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/07: - - - - 1 - 1 - - - - - - - - - - - - - - - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/08: - - - - - 1 - 1 - - - - - - - - - - - - - - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/09: - - - - - - 1 - 1 - - - - - - - - - - - - - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/10: - - - - - - - 1 - 1 - - - - - - - - - - - - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/11: - - - - - - - - 1 - 1 - - - - - - - - - - - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/12: - - - - - - - - - 1 - 1 - - - - - - - - - - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/13: - - - - - - - - - - 1 - 1 - - - - - - - - - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/14: - - - - - - - - - - - 1 - 1 - - - - - - - - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/15: - - - - - - - - - - - - 1 - 1 - - - - - - - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/16: - - - - - - - - - - - - - 1 - 1 - - - - - - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/17: - - - - - - - - - - - - - - 1 - 1 - - - - - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/18: - - - - - - - - - - - - - - - 1 - 1 - - - - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/19: - - - - - - - - - - - - - - - - 1 - 1 - - - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/20: - - - - - - - - - - - - - - - - - 1 - 1 - - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/21: - - - - - - - - - - - - - - - - - - 1 - 1 - - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2/22: 1 - - - - - - - - - - - - - - - - - - - 1 - 1 - 1 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 1
3/01: - 1 1 1 1 - - - 1 - - - - - - - - 1 1 1 1 - - - 1 - - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/02: 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/03: - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/04: 1 - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/05: - 1 - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/06: 1 - 1 - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/07: - 1 - 1 - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/08: 1 - 1 - 1 - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/09: - 1 - 1 - 1 - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/10: 1 - 1 - 1 - 1 - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/11: - 1 - 1 - 1 - 1 - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/12: 1 - 1 - 1 - 1 - 1 - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/13: 1 1 - 1 - 1 - 1 - 1 - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/14: - 1 1 - 1 - 1 - 1 - 1 - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/15: 1 - 1 1 - 1 - 1 - 1 - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/16: 1 1 - 1 1 - 1 - 1 - 1 - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/17: - 1 1 - 1 1 - 1 - 1 - 1 - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/18: - - 1 1 - 1 1 - 1 - 1 - 1 - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/19: - - - 1 1 - 1 1 - 1 - 1 - 1 - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/20: - - - - 1 1 - 1 1 - 1 - 1 - 1 - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/21: - - - - - 1 1 - 1 1 - 1 - 1 - 1 - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/22: - - - - - - 1 1 - 1 1 - 1 - 1 - 1 - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3/23: 1 - - - - - - 1 1 - 1 1 - 1 - 1 - 1 - 1 - 1 1 1 1 - - - 1 - - - - - - - 1 1 1 1 - - - 1 - - - - - - - 1 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Tabelle D.1: Die Koeffizientenmatrix A

D.2 Lösbarkeit des Gleichungssystems

Bezeichnen wir die Matrizeneinträge mit $a_{i,j}$, wobei $i, j \in \{1, \ldots, 64\}$, so kann man aus der Matrix ein Gleichungssystem der folgenden Art ableiten:

$$s_i(-22) = \sum_{j=1}^{64} a_{i,j} \mathrm{Kc}_j$$

Im nächsten Schritt muß die Frage beantwortet werden, ob dieses Gleichungssystem lösbar ist. Dies kann geschehen, indem man die Matrix A in obere Dreiecksform transformiert und nachprüft, ob sie maximalen Rang hat (d.h., ob die Einträge auf der Hauptdiagonalen alle ungleich 0 sind).

Zu diesem Zwecke wurde ein kurzes Programm implementiert, das sich an das Gauß'sche Eliminationsverfahren anlehnt, dabei jedoch die besonderen Gegebenheiten des Rechnens über $\mathbf{Z}_2$ ausnutzt. Aufgrund der erneut geringen Ansprüche in Bezug auf Rechenzeit und Speicherplatz wurde auf eine Effizienzoptimierung des Programmes verzichtet.

D.2.1 Programm zur Matrizendiagonalisierung

Das nachfolgende Programm in C++ leistet die gewünschte Diagonalisierung. Auf den Abdruck von Präprozessordirektiven wurde verzichtet. Außerdem wurde die Initialisierung des Feldes `matrix` aus Gründen der Lesbarkeit nicht mit abgedruckt. Es genügt zu wissen, daß bei der Initialisierung die in Abbildung D.1 gegebene Matrix A eingelesen wird.

Quellcode in C++

```
bool puffer[64];
bool matrix[64][64] = {...};
  //Initialisierung der Matrix aus Gr"unden der Lesbarkeit
  //ausgelassen.

int main() {
  int i,j,k;

  // Wiederhole fuer alle Zeilen i=0..63:
  for(i=0; i<64; i++) {

    // Setze eine Zeile ein, die in Spalte i eine 1 enthaelt
    j=i+1;
    while((!matrix[i][i])&&j<64) {
      if(matrix[j][i]) { // Falls eine solche Zeile gefunden wurde:
        // Vertausche Zeilen
        puffer = matrix[j];
        matrix[j] = matrix[i];
        matrix[i] = puffer;}
      j++;}

    // 'Subtrahiere' Zeile i von allen nachfolgenden Zeilen,
    // die in Spalte i eine 1 enthalten
    for(j=i+1; j<64; j++) {
      if(matrix[j][i]) {
        for(k=i; k<64; k++) {
          matrix[j][k]=matrix[j][k]^matrix[i][k];}}}

    // Gibt Zeile i aus
    for(k=0; k<64; k++) {
      cout << matrix[i][k] << " ";}
    cout << "\n";}

  return 0;
}
```

Bem.: Obwohl im Quellcode stets von Zeilen die Rede ist, die in der i-ten Spalte eine '1' enthalten, liefert der Algorithmus auch dann ein korrektes Ergebnis,

wenn eine solche Zeile nicht existiert (auch wenn er einige überflüssige Operationen ausführt). In diesem Falle besitzt die Matrix nicht maximalen Rang, was man im Ergebnis daran erkennt, daß in der betreffenden Zeile eine '0' in der Hauptdiagonalen steht. Die Effizienz des Algorithmus lïeße sich also verbessern, wenn die Diagonalisierung in einem solchen Fall vorzeitig abgebrochen würde.

D.2.2 Die diagonalisierte Matrix A'

Abbildung D.2 zeigt die Ausgabe des Diagonalisierungsprogrammes. Erneut wurde aus Gründen der Lesbarkeit jede '0' durch einen Querstrich '-' ersetzt.

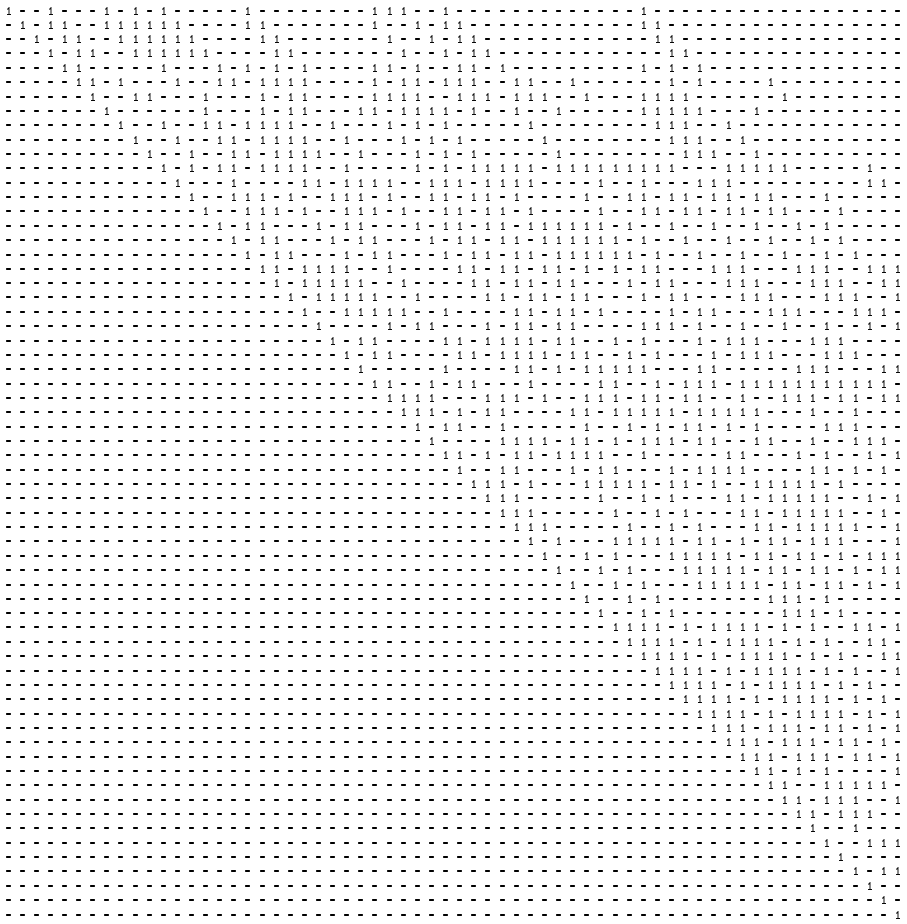

Tabelle D.2: Die diagonalisierte Koeffizientenmatrix A'

Es ist ersichtlich, daß alle Einträge auf der Hauptdiagonalen der Matrix gleich 1 sind. Somit ist der Rang der Matrix maximal, das durch die Koeffizientenmatrix A beschriebene Gleichungssystem ist somit lösbar.

Literaturverzeichnis

[And94] Ross J. Anderson, Newsgroup-Beitrag vom 17. Juni 1994.

[BGG92] T. Baritaud, H. Gilbert, M. Girault. "FFT Hashing is not Collision-free". *Advances in Cryptology - EUROCRYPT 92*. Lecture Notes in Computer Science, vol. 658, Editor: Rainer A. Rueppel, S. 35-44, Springer-Verlag 1993.

[Bri99] Marc Briceno, Mailinglistenbeitrag vom 10. Mai 1999.

[BGW98a] Marc Briceno, Ian Goldberg, David Wagner. An implementation of the GSM A3A8 algorithm. URL: "http://www.scard.org/gsm/a3a8.txt".

[BGW98b] Marc Briceno, Ian Goldberg, David Wagner. "GSM Cloning". URL: "http://www.isaac.cs.berkeley.edu/isaac/gsm-faq.html".

[BGW99] Marc Briceno, Ian Goldberg, David Wagner. A pedagogical implementation of A5/1. URL: "http://www.scard.org/gsm/a51.html".

[Cha94] W. G. Chambers. "On Random Mappings and Random Permutations". *Fast Software Encryption 1994*. Lecture Notes in Computer Science, vol. 1008, Editor: Bart Preneel, S. 26-27, Springer-Verlag 1995

[CCC98] Chaos Computer Club Deutschland. GSM-Cloning: Technischer Hintergrund. URL: "http://www.ccc.de/D2Pirat/index.html".

[Cha98] Florent Chabaud. GF(2) primitive irreducible polynomials. URL: "http://www.dmi.ens.fr/čhabaud/Poly/GF2pri.html".

[Fis86] Gerd Fischer. *Lineare Algebra*. 9. Auflage, Vieweg 1986.

[For83] Otto Forster. *Analysis 1*. 4. Auflage, Vieweg 1983.

[FR88] Walter Fumy, Hans Peter Rieß. *Kryptographie: Entwurf und Analyse symmetrischer Kryptosysteme*. Oldenbourg 1988.

[Gol97] Jovan Dj. Golić. "Cryptanalysis of Alleged A5 Stream Cipher". *Advances in Cryptology - EUROCRYPT '97*. Lecture Notes in Computer Science, vol. 1233, Editor: Walter Fumy, S. 239-255, Springer-Verlag 1997.

[GSM98] GSM Association. History of GSM. URL: "http://www.gsm.org/history/history.htm".

[GSM88] TECHNICAL INFORMATION - GSM System Security Study. URL: "http://jya.com/gsm061088.htm".

[LN94] Rudolf Lidl, Harald Niederreiter. *Introduction to finite fields and their applications*. Cambridge University Press 1994.

[Lub96] Michael Luby. *Pseudorandomness and cryptographic applications*. Princeton University Press 1996.

[Mei94] W. Meier, O. Staffelbach. "The self-shrinking generator". *Advances in Cryptology - EUROCRYPT '94*. Lecture Notes in Computer Science, vol. 950, Editor: Alfredo de Santis, S. 205-214, Springer-Verlag 1995.

[MOV96] Alfred J. Menezes, Paul C. von Oorschot, Scott A. Vanstone. *Handbook of applied cryptography*. CRC Press 1996.

[RE96] Wolfgang Rankl, Wolfgang Effing. *Handbuch der Chipkarten*. Hanser-Verlag 1996.

[Rue92] Rainer A. Rueppel. "Stream Ciphers". *Contemporary Cryptology: The Science of Information Integrity*, Editor: Gustavus Simmons, S. 65-134, IEEE Press 1992.

[Shn96] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms and Source Code in C*. 2. Auflage, John Wiley & Sons 1996.

[Shr91] Claus P. Schnorr. "An Efficient Cryptographic Hash Function", vorgestellt auf der Rump Session zur CRYPTO '91.

[Shr92] Claus P. Schnorr. "FFT-Hash II, Efficient Cryptographic Hashing". *Advances in Cryptology - EUROCRYPT '92*. Lecture Notes in Computer Science, vol. 658, Editor: Rainer A. Rueppel, S. 45-54, Springer-Verlag 1993.

[SH98] Christiane Schulzki-Haddouti. *"Hintertür für Spione"*, in: Die Zeit, Nr. 39/1998.

[Shep94] Simon Shepherd. "Cryptanalysis of the GSM A5 Cipher Algorithm". *IEE Colloquium on Security and Cryptography Applications to Radio Systems*, digest no. 1994/141. (COMMERCIAL-IN-CONFIDENCE).

[Vau92] Serge Vaudenay. "FFT-Hash II is not yet Collision-free". *Advances in Cryptology - CRYPTO '92*. Lecture Notes in Computer Science, vol. 740, Editor: Ernest F. Brickell, S. 587-593, Springer-Verlag 1993.

[Ved98] Klaus Vedder. "GSM: Security, Services, and the SIM". *State of the Art in Applied Cryptography*. Lecture Notes in Computer Science, vol. 1528, Ed.: Bart Preneel, Vincent Rijmen, S. 224-240, Springer-Verlag 1998.

[Wob98] Reinhard Wobst. *Abenteuer Kryptologie*. 2. Auflage, Addison-Wesley 1998.