ANALYSIS OF LIGHTWEIGHT STREAM CIPHERS

THÈSE N^{O} 4040 (2008)

PRÉSENTÉE LE 18 AVRIL 2008

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE DE SÉCURITÉ ET DE CRYPTOGRAPHIE
PROGRAMME DOCTORAL EN INFORMATIQUE, COMMUNICATIONS ET INFORMATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Simon FISCHER

M.Sc. in physics, Université de Berne
de nationalité suisse et originaire de Olten (SO)

acceptée sur proposition du jury:

Prof. M. A. Shokrollahi, président du jury
Prof. S. Vaudenay, Dr W. Meier, directeurs de thèse
Prof. C. Carlet, rapporteur
Prof. A. Lenstra, rapporteur
Dr M. Robshaw, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
Suisse
2008

Für Philomena

Abstract

Stream ciphers are fast cryptographic primitives to provide confidentiality of electronically transmitted data. They can be very suitable in environments with restricted resources, such as mobile devices or embedded systems. Practical examples are cell phones, RFID transponders, smart cards or devices in sensor networks. Besides efficiency, security is the most important property of a stream cipher. In this thesis, we address cryptanalysis of modern lightweight stream ciphers. We derive and improve cryptanalytic methods for different building blocks and present dedicated attacks on specific proposals, including some eSTREAM candidates. As a result, we elaborate on the design criteria for the development of secure and efficient stream ciphers. The best-known building block is the linear feedback shift register (LFSR), which can be combined with a nonlinear Boolean output function. A powerful type of attacks against LFSR-based stream ciphers are the recent algebraic attacks, these exploit the specific structure by deriving low degree equations for recovering the secret key. We efficiently determine the immunity of existing and newly constructed Boolean functions against fast algebraic attacks. The concept of algebraic immunity is then generalized by investigating the augmented function of the stream cipher. As an application of this framework, we improve the cryptanalysis of a well-known stream cipher with irregularly clocked LFSR's. Algebraic attacks can be avoided by substituting the LFSR with a suitable nonlinear driving device, such as a feedback shift register with carry (FCSR) or the recently proposed class of T-functions. We investigate both replacement schemes in view of their security, and devise different practical attacks (including linear attacks) on a number of specific proposals based on T-functions. Another efficient method to amplify the nonlinear behavior is to use a round-based filter function, where each round consists of simple nonlinear operations. We use differential methods to break a reduced-round version of eSTREAM candidate Salsa20. Similar methods can be used to break a related compression function with a reduced number of rounds. Finally, we investigate the algebraic structure of the initialization function of stream ciphers and provide a framework for key recovery attacks. As an application, a key recovery attack on simplified versions of eSTREAM candidates Trivium and Grain-128 is given.

Keywords: Cryptanalysis, Stream Cipher, Algebraic Attacks, T-functions, eSTREAM

Zusammenfassung

Stromchiffren sind schnelle kryptografische Verfahren, um die Vertraulichkeit von elektronisch übermittelten Daten zu gewährleisten. Sie können in Umgebungen mit eingeschränkten Ressourcen eingesetzt werden, etwa in mobilen Geräten oder in eingebetteten Systemen. Praktische Beispiele sind Mobiltelefone, RFID Transponder, Smartcards oder Geräte in Sensornetzwerken. Nebst der Effizienz ist die Sicherheit die wichtigste Eigenschaft einer Stromchiffre. In dieser Doktorarbeit behandeln wir die Kryptanalyse von modernen und leichtgewichtigen Stromchiffren. Wir entwickeln und verbessern kryptanalytische Methoden für verschiedene Bausteine, und präsentieren Angriffe auf spezifische Verfahren, einschliesslich einigen eSTREAM Kandidaten. Daraus ergeben sich diverse Kriterien für das Design und die Entwicklung von sicheren und effizienten Stromchiffren. Der am besten bekannte Baustein von Stromchiffren ist das lineare Schieberegister (LFSR), welches mit einer nichtlinearen Boolschen Filterfunktion kombiniert werden kann. Ein mächtiger Angriff gegen LFSR-basierte Stromchiffren sind die algebraischen Angriffe, welche die spezifische Struktur ausnutzen um tiefgradige Gleichungen zu erhalten und den geheimen Schlüssel zu rekonstruieren. Für bestehende und zukünftig konstruierte Boolsche Funktionen können wir effizient die Immunität gegen Schnelle Algebraische Angriffe bestimmen. Das Konzept der algebraischen Immunität kann auf die Erweiterte Funktion der Stromchiffre verallgemeinert werden. Die Methode wird dann erfolgreich auf eine bekannte Stromchiffre mit irregulär getakteten LFSR's angewendet. Im Allgemeinen können algebraische Angriffe verhindert werden, indem das LFSR durch einen geeigneten nichtlinearen Baustein ersetzt wird, etwa ein Schieberegister mit Speicher (FCSR) oder eine der kürzlich vorgeschlagenen T-funktionen. Wir untersuchen beide Bausteine im Hinblick auf die Sicherheit, und entwickeln diverse (insbesondere lineare) Angriffe gegen Stromchiffren, die auf T-funktionen basieren. Es gibt auch andere effiziente Konstruktionen, um das nichtlineare Verhalten einer Stromchiffre zu verstärken, etwa eine rundenbasierte Filterfunktion, wobei jede Runde aus einfachen nichtlinearen Operationen besteht. Wir verwenden differenzielle Methoden, um den eSTREAM Kandidaten Salsa20 für eine reduzierte Anzahl Runden zu brechen. Ähnliche Methoden können dann verwendet werden, um eine verwandte Kompressionsfunktion (ebenfalls mit einer reduzierten Anzahl Runden) zu brechen. Schliesslich untersuchen wir die algebraische Struktur der Initialisierungsfunktion von Stromchiffren, und stellen eine allgemeine Methode für die Rekonstruktion vom Schlüssel bereit. Als Anwendung präsentieren wir einen Angriff auf vereinfachte Versionen der eSTREAM Kandidaten Trivium und Grain-128.

Schlüsselwörter: Kryptanalyse, Stromchiffren, Algebraische Angriffe, T-Funktionen, eSTREAM

Acknowledgments

I would like to thank my supervisors Dr Willi Meier and Prof. Serge Vaudenay for offering me the chance of this valuable PhD position. After a demanding start for a physicist, I experienced a very fruitful supervision of Dr Willi Meier, and my benefit is not only of technical nature. I am deeply grateful for this. It is a honor for me to have Prof. Claude Carlet, Prof. Arjen Lenstra, Dr Matt Robshaw and Prof. Amin Shokrollahi in the thesis jury, thank you sincerely for investing your time. One of the best things in research is collaboration with motivated colleagues: I would like to express my gratitude to Dr Pascal Junod and Dr Frederik Armknecht. I was pleased when Jean-Philippe Aumasson, Shahram Khazaei and Reza Ebrahimi Atani later joined our team, working and discussing with them was very motivating. I would also like to thank Shahram for our great collaboration and for his brilliant ideas he shared with us. I cordially thank all members of the institutes IAST, IPSP and LASEC, I was very well integrated and enjoyed the pleasant ambiance; my work environment could not have been better. Many thanks to the directors of the institutes Prof. Heinz Burtscher and Prof. Rolf Gutzwiller. I gratefully acknowledge the support of this thesis by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322. Finally, I would like to thank my family and close friends for their help and friendship. My lovely mother constantly encouraged me to pursue my education. No words can express my gratitude to my wonderful wife Philomena, our love gives me more than anything else. This thesis is dedicated to her.

Contents

Chapter 1

Introduction

Motivation

Today's digital communication technologies require adequate security. Stream ciphers provide confidentiality of electronically transmitted data. Compared to other primitives, stream ciphers are competitive in software applications with exceptionally high speed, and in hardware applications with exceptionally small footprint. With the appearance of mobile devices and embedded systems (such as cell phones, RFID transponders, smart cards or devices in sensor networks), the latter becomes more significant and matches up with the concept of lightweight cryptography. However, the attacks found on well-known stream ciphers make it necessary to accomplish large efforts in the invention of new replacement schemes, and in return, to cryptanalyze the new schemes. Furthermore, it would be attractive to combine functionalities of primitives, *e.g.* authentication or integrity methods may be associated to stream ciphers. In this context, the ECRYPT project named eSTREAM has been initiated in 2004 to design and analyze new proposals of stream ciphers *"suitable for widespread adoption"* [104]. This project is a successor of the NESSIE project initiated in 2000, where no stream cipher was elected for the final portfolio. In contrast, the block cipher Rijndael was selected in 2001 to be the advanced encryption standard (AES). The AES is very popular and well studied. A modern stream cipher should be *"superior to the AES in at least one significant aspect"* [104], where we assume that the AES is used in some appropriate mode, *e.g.* counter mode.

Thesis Outline

In this thesis, we focus on the cryptanalysis of synchronous stream ciphers. The outline of this thesis is as follows: In Chapter 2, we present our preliminaries, including the formalism for stream ciphers and some important attacks and concepts. Our analysis begins in Chapter 3 with the well-known filter generators, which consist of a linear update and a nonlinear Boolean output function. To evaluate resistance against fast algebraic attacks, we present efficient algorithms and theoretical bounds. In Chapter 4, we are able to generalize the concept of algebraic immunity of stream ciphers by investigating the

augmented function. As an application of this framework, we improve the cryptanalysis of a well-known stream cipher with irregularly clocked LFSR's, see Chapter 5. Algebraic attacks are more difficult for nonlinear driving devices such as shift registers with carry (FCSR's). In Chapter 6, we investigate different representations for an FCSR-based stream cipher. Another recently proposed building block for stream ciphers are the so-called T-functions. In Chapter 7, we present a collection of practical attacks on stream ciphers based on T-functions. Instead of a nonlinear building block, one could use a round-based filter function, where each round consists of simple nonlinear operations. In Chapter 8, we use differential methods to break reduced-round versions of two stream ciphers and a related compression function. In Chapter 9, we investigate the algebraic structure of the initialization function of stream ciphers and provide a general framework for key recovery attacks. We finally draw our conclusions in Chapter 10. In the Appendix, we present a very simple and efficient attack on a specific stream cipher. We have cryptanalytic results for the following eSTREAM candidates: F-FCSR, Grain-128, MAG, Salsa20, Trivium, TSC-4. We have additional results for ASG, ChaCha, Rumba, TF-0, TF-0M, TSC-1, TSC-2, and for filter generators with different filter functions.

Reports and Publications

Here is a list of conference publications:

1. Simon Künzli, Pascal Junod, and Willi Meier. Distinguishing Attacks on T-functions. In the proceedings of Ed Dawson and Serge Vaudenay, editors, Progress in Cryptology - MyCrypt 2005, First International Conference on Cryptology in Malaysia, Kuala Lumpur, Malaysia, September 28-30, 2005. Volume 3715 of Lecture Notes in Computer Science, pages 2-15. Springer-Verlag, 2005. *Award for the best paper of the conference.*
2. Frederik Armknecht, Claude Carlet, Philippe Gaborit, Simon Künzli, Willi Meier, and Olivier Ruatta. Efficient Computation of Algebraic Immunity for Algebraic and Fast Algebraic Attacks. In the proceedings of Serge Vaudenay, editor, Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Saint Petersburg, Russia, May 28 - June 1, 2006. Volume 4004 of Lecture Notes in Computer Science, pages 147-164. Springer-Verlag, 2006.
3. Simon Fischer, Willi Meier, Côme Berbain, Jean-François Biasse, and Matthew Robshaw. Non-randomness in eSTREAM Candidates Salsa20 and TSC-4. In the proceedings of Rana Barua and Tanja Lange, editors, Progress in Cryptology - INDOCRYPT 2006, 7th International Conference on Cryptology in India, Kolkata, India, December 11-13, 2006. Volume 4329 of Lecture Notes in Computer Science, pages 2-16. Springer-Verlag, 2006.
4. Simon Fischer and Willi Meier. Algebraic Immunity of S-boxes and Augmented Functions. In the proceedings of Alex Biryukov, editor, Fast Software Encryption - FSE 2007, 14th International Workshop, Luxembourg City, Luxembourg, March

26-28, 2007. Volume 4593 of Lecture Notes in Computer Science, pages 366-381. Springer-Verlag, 2007.

5. Shahram Khazaei, Simon Fischer, and Willi Meier. Reduced Complexity Attacks on the Alternating Step Generator. In the proceedings of Carlisle M. Adams, Ali Miri and Michael J. Wiener, editors, Selected Areas in Cryptography - SAC 2007, 14th International Workshop, Ottawa, Canada, August 16-17, 2007. Volume 4876 of Lecture Notes in Computer Science, pages 1-16. Springer-Verlag, 2007.
6. Jean-Philippe Aumasson, Simon Fischer, Shahram Khazaei, Willi Meier, and Christian Rechberger. New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba. To appear in the proceedings of Kaisa Nyberg, editor, Fast Software Encryption - FSE 2008, 15th International Workshop, Lausanne, Switzerland, February 10-13, 2008. Full version available at `http://eprint.iacr.org/2007/472`. *Award of $1000 for the most interesting analysis of Rumba.*
7. Simon Fischer, Shahram Khazaei, and Willi Meier. Chosen IV Statistical Analysis for Key Recovery Attacks on Stream Ciphers. To appear in the proceedings of Serge Vaudenay, editor, AfricaCrypt 2008, Casablanca, Morocco, June 11-14, 2008. Also in SASC 2008 - The State of the Art of Stream Ciphers, Lausanne, Switzerland, February 13-14, 2008. Workshop Record, pages 33-42.

Here is a list of technical reports:

1. Simon Künzli and Willi Meier. Distinguishing Attack on MAG. In eSTREAM, ECRYPT Stream Cipher Project, Report 2005/053, 2005.
2. Simon Fischer, Willi Meier, and Dirk Stegemann. Equivalent Representations of the F-FCSR Keystream Generator. In SASC 2008 - The State of the Art of Stream Ciphers, Lausanne, Switzerland, February 13-14, 2008. Workshop Record, pages 87-96.
3. Frederik Armknecht, Claude Carlet, Simon Fischer, Philippe Gaborit, Willi Meier, and Olivier Ruatta. Efficient Computation of Algebraic Immunity for Algebraic and Fast Algebraic Attacks. Technical Report, 2008. *This is an extended version of the EUROCRYPT 2006 paper.*

On behalf of a consulting agreement with a Korean agency, we have evaluated the security and efficiency of a new stream cipher (in collaboration with Willi Meier, Peter Steigmeier, and Werner Witz). This work is not part of the thesis.

Chapter 2

Preliminaries

Cryptology is the science of information protection against unauthorized parties [98,119]. It can be split up into cryptography (design of cryptographic systems) and cryptanalysis (security analysis of cryptographic systems). The most important aspects of information protection are confidentiality (information is secret from unauthorized parties), authenticity (information originates from authorized party) and integrity (information is protected against malicious modification). Modern cryptographic problems also include electronic payment, electronic votes *etc.* Cryptographic algorithms can be classified into secret-key (or symmetric) algorithms and public-key algorithms. Finally, secret-key algorithms can be block ciphers or stream ciphers. We describe symmetric cryptosystems in general (providing confidentiality), and give an overview of designs and attacks on stream ciphers. This requires to introduce some notational conventions.

2.1 Notational Preliminaries

We denote by $\mathbb{F}$ the finite field GF(2) and by $\mathbb{F}^n$ the vector space of dimension n over $\mathbb{F}$. An element of $\mathbb{F}^n$ (*i.e.* a word of n bits) is denoted $x := (x_0, \ldots, x_{n-1})$ in vectorial notation, or $x := x_0 || \ldots || x_{n-1}$ in big-endian bitwise notation, where $||$ denotes concatenation. It can also be identified as an integer $x = \sum_{i=0}^{n-1} x_i 2^i$, where x_0 is the least significant bit (lsb), and x_{n-1} is the most significant bit (msb). The support of x is defined to be the set $\mathrm{supp}(x) := \{i | x_i = 1\}$ and the Hamming weight of x is $\mathrm{wt}(x) := |\mathrm{supp}(x)|$. For $x, y \in \mathbb{F}^n$, let $x \subseteq y$ be an abbreviation for $\mathrm{supp}(x) \subseteq \mathrm{supp}(y)$. Arithmetic operations like $+$, $-$, $\cdot$ are performed modulo 2^n. Boolean operations are performed on all n bits in parallel and are denoted by $\wedge$ (`AND`), $\vee$ (`OR`), and $\oplus$ (`XOR`). In addition, $\ll k$ (resp. $\gg k$) denotes a left (resp. right) shift by k positions (with zero-padding), and $\lll k$ (resp. $\ggg k$) denotes a cyclic left (resp. right) shift by k positions (*i.e.* a rotation). It will be clear from the context if $x = (x_0, \ldots, x_{m-1})$ denotes an element of $\mathbb{F}^{m \times n}$, *i.e.* a vector of m words of n bits each. In this case, a single bit j of word i is denoted $[x_i]_j$. A Boolean function f is a mapping from the set $\mathbb{F}^n$ to $\mathbb{F}$. One possibility to characterize f is its truth table $T(f) \in \mathbb{F}^{2^n}$. It is defined by $T(f) := (f(0), \ldots, f(2^n - 1))$. The Boolean function is balanced if $\mathrm{wt}(T(f)) = 2^{n-1}$. An alternative description of a Boolean function is its

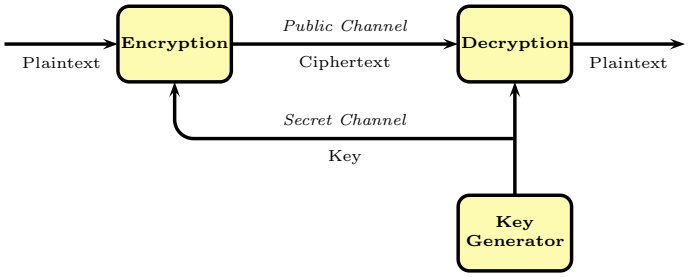

Figure 2.1: Symmetric encryption.

algebraic normal form. Let $\mathbb{F}[x_0, \ldots, x_{n-1}]$ be the ring of multivariate polynomials over $\mathbb{F}$ in the n unknowns $x_0, \ldots, x_{n-1}$. For a multi-index $\alpha \in \mathbb{F}^n$, we define the monomials by $x^\alpha := x_0^{\alpha_0} \cdot \ldots \cdot x_{n-1}^{\alpha_{n-1}}$. Then, any Boolean function $f : \mathbb{F}^n \to \mathbb{F}$ has a unique expression

$$f(x) = \bigoplus_\alpha f_\alpha \, x^\alpha, \quad f_\alpha \in \mathbb{F} \; . \tag{2.1}$$

Consequently, we define its coefficients vector $C(f) \in \mathbb{F}^{2^n}$ by $C(f) := (f_0, \ldots, f_{2^n-1})$ and its degree by $\deg(f) := \max\{|\alpha| : f_\alpha = 1\}$. We will also consider vectorial Boolean functions (or S-boxes) from $\mathbb{F}^n$ to $\mathbb{F}^m$.

2.2 Definition of a Cryptosystem

According to the communication model introduced by Shannon [113], there is a sender and a receiver with a public communication channel. The goal of the sender is to send some information (the plaintext) in a confidential way to the receiver. This can be achieved with a cryptosystem and an additional secure channel of low bandwidth. In symmetric cryptography, the secure channel can not be eavesdropped by an adversary, and it is used to transmit a secret key. Given the plaintext, the secret key and the cryptosystem, the sender can construct the ciphertext and send it to the receiver over the public communication channel. The receiver can then reconstruct the plaintext, given the ciphertext, the secret key and the cryptosystem, see Fig. 2.2. Formally, a cryptosystem is defined as follows:

Definition 1. *A cryptosystem consists of a plaintext space $\mathcal{P}$, a ciphertext space $\mathcal{C}$ and a key space $\mathcal{K}$. There is an encryption algorithm $\mathsf{Enc} : \mathcal{K} \times \mathcal{P} \to \mathcal{C}$ and a decryption algorithm $\mathsf{Dec} : \mathcal{K} \times \mathcal{C} \to \mathcal{P}$. For each $K \in \mathcal{K}$ and $p \in \mathcal{P}$, it is $\mathsf{Dec}(K, \mathsf{Enc}(K, p)) = p$.*

A cryptosystem is necessary to protect the information from eavesdropping of a third entity, which will be called the *adversary*. In an adversary model, the means and goals of an adversary are defined. According to Kerckhoff's Principle, an adversary knows the specification of the cryptosystem and has access to the ciphertext c. The goal of an adversary is to recover (part of) the plaintext, or to recover the secret key. Another reasonable scenario is the known-plaintext attack, where the adversary knows one or

more pairs of ciphertext with corresponding plaintext, and her goal is to decrypt other ciphertexts or to recover the key. In some situations, it may be reasonable to assume that the adversary has access to the physical device which contains an implementation of the cryptosystem. All kinds of physical emanations from the device (like power consumption, radiation, execution times *etc.*) can then potentially be used in a side-channel attack to recover the key. Given some adversary model, any attack is evaluated in terms of the required amount of data, time (number of basic operations) and memory. Consider the known-plaintext scenario, where the adversary knows a pair (p, c) and tries to recover the key K. An obvious attack is to try all possible keys K of the finite set $\mathcal{K}$, until the equation $\mathsf{Enc}(K, p) = c$ is verified. This is a basic brute-force attack, which is independent of the details of the underlying cryptosystem. Consequently, the size of the key space determines the maximum security of a cryptosystem, which should be related to the computational power of a strong adversary. If there exists no better attack than brute-force, the cryptosystem is computational secure. Otherwise, the cryptosystem is said to be broken (which does not mean that the attack is practical). For many cryptosystems, the problem of recovering the key can be seen as solving a huge system of nonlinear Boolean equations. Shannon claimed that breaking a good cipher should require *"as much work as solving a system of simultaneous equations in a large number of unknowns of a complex type"* (where the unknowns are the key bits). In general, such a problem is known to be NP-hard.

2.3 Stream Ciphers

Stream ciphers are an important class of symmetric encryption algorithms [111]. They encrypt individual symbols (usually binary digits) of a plaintext one at a time, using an encryption transformation which varies with time, see Fig. 2.2 and Fig. 2.3. Here is a formal definition:

Definition 2. *A synchronous stream cipher consists of an internal state $x \in \mathcal{X}$, an update function $L : \mathcal{X} \to \mathcal{X}$ and an output function $f : \mathcal{X} \to \mathcal{Z}$, where $\mathcal{Z}$ is the keystream alphabet. An output $z \in \mathcal{Z}$ at time t is produced according to $z^t = f(x^t)$, where $x^t = L^t(x)$ and x is the initial state. The initial state x is produced by an initialization function from the secret key K and an initialization vector IV denoted by V. The stream of outputs $z^0, z^1, \ldots$ is called the keystream. Each output symbol is then combined with the corresponding plaintext symbol to produce a ciphertext symbol.*

In a synchronous stream cipher, the keystream is independent of the plaintext and ciphertext. There is no error propagation, but both the sender and receiver must be synchronized. If synchronization is lost due to ciphertext digits being inserted or deleted during transmission, then decryption fails and can only be restored through *re-synchronization*. In the re-synchronization process, a public initialization vector (IV) is exchanged and loaded into the keyed internal state, without exchanging a new key. This way, the generator will produce a unique keystream (independent from other keystreams produced

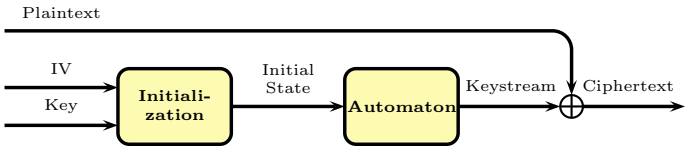

Figure 2.2: Stream cipher from a high level.

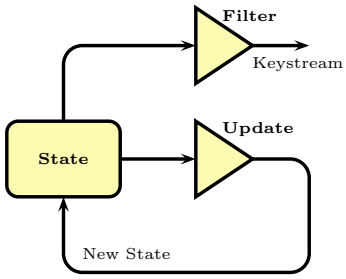

Figure 2.3: Automaton of a stream cipher.

by the same key) every time it is invoked, such that keystream blocks are never re-used. Additional methods for authentication and integrity are needful in many applications.

Compared to other cryptosystems, synchronous stream ciphers are advantageous in software applications with very high throughput requirements, or in hardware applications with restricted resources (such as limited storage, gate count, or power consumption). They are also appropriate when buffering is limited or when characters must be individually processed as they are received. Because they have no error propagation, synchronous stream ciphers may also be advantageous in situations where transmission errors are highly probable.

For a binary additive stream cipher it is $\mathcal{Z} = \{0, 1\}$, and the plaintext, ciphertext and keystream are binary streams of equal size. The encryption is then defined by $c = p \oplus z$, and decryption simply becomes $p = c \oplus z$. If the keystream z is a uniformly random stream which is used only once, then the binary additive stream cipher is called a *one-time pad*. According to Shannon [113], the one-time pad is *unconditionally secure*, which means no secret information can be obtained from the ciphertext even for an adversary with unlimited resources. However, this scheme is not efficient because it requires random keys of the same size as the plaintext. With a stream cipher according to Def. 2, one can use a small key to initialize an automaton and generate a *pseudo-random* keystream of length of the plaintext. In the standard adversary model of a stream cipher, it is assumed that the adversary knows some part of the keystream (corresponding to a known-plaintext attack) for chosen IV's, and her goal is to distinguish the keystream from a uniformly random stream, or to predict the keystream, or to recover the internal state (if the update function and the initialization function are invertible, then the key can be derived from the internal state).

Another class of stream ciphers are *self-synchronizing* stream ciphers, where the keystream is generated as a function of the key and a fixed number of previous ciphertext

digits. Self-synchronization is possible if ciphertext digits are deleted or inserted, because the decryption function depends only on a fixed number of preceding ciphertext digits. One of the rare examples is eSTREAM Phase 3 candidate Moustique. We do not further investigate this class. Finally, we remark that some stream ciphers have additional mechanisms for *message authentication* (but no eSTREAM candidate since Phase 3).

2.4 Other Cryptosystems

In this section, we briefly present two other important classes of cryptosystems. A *block cipher* is a symmetric cryptosystem with $\mathcal{P} = \mathcal{C} = \mathbb{F}^n$ for a block size n. For each key K, the encryption function $\mathsf{Enc}(K, p)$ is a *permutation*. In the most general case, the key space corresponds to the set of permutations of size $2^n!$, where a single key is represented by a table of size 2^n. It is reasonable to use only a subset of the permutations, which can be generated efficiently with a small key. To encrypt messages longer than the block size, a mode of operation is used. The *output feedback* (OFB) mode makes a block cipher into a synchronous stream cipher. Most block ciphers are constructed by repeatedly applying a simple round function, which consists of substitutions and permutations (*i.e.* realizing the concept of confusion and diffusion). Security of block ciphers is well studied, but block ciphers are typically less efficient compared to dedicated stream ciphers.

In symmetric cryptography, *key management* is a main concern. Each pair of participants must share a secret key, which gives a huge number of $N(N-1)/2$ keys for N participants. One solution is to use a trusted *key distribution center*, which shares a single key with each participant. Another solution is to use *public-key cryptosystem*. In a public-key cryptosystem, each participant has a secret key (for decryption) and a public key (for encryption). The public key can be distributed to all participants, using an authenticated (but not confidential) channel. It should be computationally difficult to compute the secret key given the public key, or to decrypt without the secret key (where the secret key is a trapdoor of a potential *one-way function*). Public-key algorithms are based on computationally hard problems, such as factorization. These algorithms are much less efficient than symmetric algorithms. Hence, public-key cryptosystems are often used to exchange a secret session key only, and are then replaced by efficient symmetric cryptosystems to secure the communication channel.

2.5 Cryptographic Hash Functions

A cryptographic hash function is a fixed (and unkeyed) transformation that takes an input of arbitrary size, and returns a string of fixed size n, which is called the *hash value*. The hash value is a concise representation of the (potentially large) input from which it was computed (*i.e.* a digital fingerprint). Hash functions are used in many cryptographic protocols, *e.g.* for message integrity checks and digital signatures. A hash function should behave as much as possible like a random function while still being deterministic and efficiently computable. It should have the following three security properties:

1. **Preimage resistance**: given $\mathsf{Hash}(x)$, the complexity to find an input x is not smaller than 2^n.
2. **Second preimage resistance**: given an input x, the complexity to find a second input y such that $\mathsf{Hash}(x) = \mathsf{Hash}(y)$ is not smaller than 2^n.
3. **Collision resistance**: The complexity to find any x and y such that $\mathsf{Hash}(x) = \mathsf{Hash}(y)$ is not smaller than $2^{n/2}$ using a serial birthday attack (see [122,20] for more advanced birthday attacks).

Most unkeyed hash functions are designed as iterative processes which hash arbitrary length inputs by processing successive fixed-size blocks of the input using a *compression function* f. This is known as the Merkle-Damgård construction, see *e.g.* [51]. A hash input x of arbitrary finite length is divided into fixed-length blocks x_i of r bits. This preprocessing typically involves appending extra bits (padding). Each block x_i then serves as input to the compression function f, which computes a new intermediate result h_i (the chaining variable) of bitlength n as a function of the previous intermediate result h_{i-1} and the next input block x_i. The initial chaining variable h_0 is a prespecified value or an IV, and the final chaining variable is the hash value (or an optional output transformation could be used). With this construction, collision resistance of the hash function can be reduced to the collision resistance of the compression function. Commonly used (iterative) hash functions are MD5 and SHA-1. In 2005, security flaws were identified in both algorithms, see *e.g.* [123]. The U.S. Institute of Standards and Technology (NIST) is initiating an effort to develop one or more additional hash algorithms through a public competition. In Chapter 8, we present a collision attack on a reduced-round compression function.

2.6 Designs of Stream Ciphers

We describe some well-known designs for stream ciphers, based on feedback shift registers and T-functions.

2.6.1 Feedback Shift Registers

Feedback shift registers, in particular *linear feedback shift registers*, are the basic components of many stream ciphers because they are well-suited for hardware implementations, and produce sequences having large periods and good statistical properties, see *e.g.* [111].

Definition 3. *A binary linear feedback shift register (LFSR) of size n is a finite state automaton with internal state of n bits. In each clock cycle, the update function L shifts the state by one position, where the input bit is a linear function of the previous bits. More precisely, let $x = (x_0, \ldots, x_{n-1})$ be the initial state. Then, the output sequence $X = (x_0, x_1, \ldots)$ is determined by the recursion $x_t = (c_1 x_{t-1} \oplus \ldots \oplus c_n x_{t-n})$ for $t \geq n$, where all c_i are fixed elements in $\{0, 1\}$.*

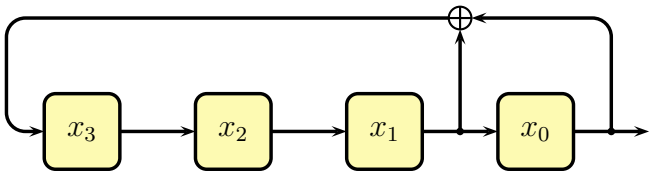

Figure 2.4: Example of a linear feedback shift register of size $n = 4$ and with feedback polynomial $C(D) = 1 + D^3 + D^4$.

The *connection polynomial* of the LFSR is defined by $C(D) := 1 \oplus c_1 D \oplus \ldots \oplus c_n D^n$. Let $S(D)$ be the formal power series $S(D) := x_0 \oplus x_1 D \oplus x_2 D^2 \oplus \ldots$, then the LFSR recursion is equivalent to $C(D)S(D) = P(D)$ for a polynomial $P(D)$ which is related to the initial state. If $C(D)$ is a primitive polynomial, then each of the $2^n - 1$ nonzero initial states of the corresponding LFSR produces an output sequence with maximum possible period $2^n - 1$. Hence, an LFSR with primitive connection polynomial is called a *maximum-length* LFSR. The output sequence of a maximum-length LFSR has good statistical properties, see [98].

Example 1. Consider an LFSR of size $n = 4$ and with primitive connection polynomial $C(D) = 1 \oplus D^3 \oplus D^4$, see Fig. 2.4. With the initial state $x = (1, 1, 0, 0)$, the output sequence becomes $X = (0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, \ldots)$ and is periodic with period 15. □

Because of the linear structure, an LFSRshould not be used by itself: the LFSR could be initialized with any n bits of the output to generate the remaining bits of the output (or to recover the initial state by solving a linear system). There are three general methodologies for destroying the linearity properties of an LFSR:

1. **Filter generators**: Use a nonlinear filter function on the contents of a single LFSR.
2. **Combination generators**: Use a nonlinear combining function (potentially with memory) on the outputs of several LFSR's.
3. **Clock-controlled generators**: Use the output of one (or more) LFSR's to control the clock of one (or more).

Nonlinear filters for designs of Type 1 will be analyzed in Chapters 3 and 4. Well-known examples of Type 2 are the Geffe Generator and the Summation Generator, a real-world application is E0 (which is used in the Bluetooth technology). Well-known examples of Type 3 are the Alternating Step Generator (see Chapter 5), the Shrinking Generator, and the Self-Shrinking Generator, a real-world application is A5 (which is used in the GSM technology for mobile communication). These three classical types of designs are not considered as antiquated: the structure is very simple, and serves as a basis for modern constructions, although many of the proposals are broken [26, 92].

More recent constructions use *nonlinear feedback shift registers* (NFSR's), where a nonlinear Boolean function serves as feedback function. Examples are the eSTREAM Phase 2 and 3 candidates Grain (filter generator with NFSR, LFSR and filter), ACHTERBAHN

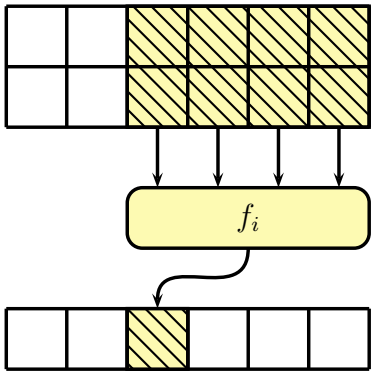

Figure 2.5: A T-function from two 6-bit words to one 6-bit word. For example, bit 4 of the output is determined by bits 1 to 4 of the two input words.

(combination generator with NFSR's) and MICKEY (clock-controlled generator with NFSR and LFSR). It should be noticed that the analysis of NFSR's is much more involved for both, the designer and the adversary. Jump registers can be used for efficient clock-controlled generators. A new type of feedback register was introduced by Klapper and Goresky [83] and is called a *feedback shift register with carry* (FCSR), which is equipped with auxiliary memory for storing the (integer) carry. An FCSR is similar to an LFSR, except that the contents of the tapped stages of the shift register are added as integers to the current content of the memory to form a sum. The least significant bit of the sum is then fed back into the first stage of the shift register, while the remaining higher order bits are retained as the new value of the memory. FCSR's can be conveniently analyzed using the algebra over the 2-adic numbers. An example of such a design is the eSTREAM Phase 3 candidate F-FCSR, which will be analyzed in Chapter 6.

2.6.2 T-functions

We have seen that LFSR's are simple primitives which are well understood, but the clean mathematical structure can also help an adversary to find an attack. In contrast to this tame approach, one could also use crazy compositions of operations, hoping that neither the designer nor the adversary will be able to analyze the mathematical behavior of the scheme. This wild approach is often preferred in real-world designs. Recently, triangular functions (*T-functions*) have been introduced by Klimov and Shamir, see [85,86,87,84]. In a T-function, information does not propagate from left to right. T-functions are semi-wild: they can look like crazy combinations of nonlinear Boolean and arithmetic operations, but have many analyzable mathematical properties. Here is a definition:

Definition 4. *A (multiword) T-function is a mapping from k n-bit words to l n-bit words, in which each bit i of any of the outputs ($0 \leq i \leq n-1$) can depend only on bits $0, \ldots, i$ of the inputs.*

All the Boolean operations and most of the arithmetic operations (such as addition and multiplication, but not right shift and circular shift) in modern processors are T-functions,

and can be executed in one clock cycle. Compositions of T-functions are also T-functions, which allows to design many T-functions with very efficient software implementation. A T-function with $k = l$ words could be used iteratively as an update function in a stream cipher. From an efficiency point of view, a composition of a small number of basic T-functions would be desirable. From the security point of view, the composition should be a mixture of Boolean and arithmetic operations, including some nonlinear subexpressions such as squaring. Klimov and Shamir developed tools to analyze invertibility and the cycle structure of T-functions. *Invertibility* is important in a stream cipher, because if we repeatedly apply an update function to the internal state, we want to prevent an incremental loss of entropy. The cycle structure of an invertible T-function is also important, since we do not want the sequence of generated states to be trapped in a short cycle. A T-function has the *single-cycle* property, if its repeated application to any initial state goes through all the 2^{kn} possible states. For a single-cycle T-function used as update function, no weak initial states are possible. It can be seen as replacement schemes of maximum-length LFSR's. However, the period of the least significant bits (or bit-slices) of T-functions is small by construction: The period for bit(-slice) i is at most 2^{ki} for a state of k words. Consequently, one should only use the most significant bits in an additional filter function, or mix lower and higher bits with cyclic shifts.

Example 2. The mapping $x \mapsto x+(x^2 \vee 5) \mod 2^n$ is an invertible T-function with a single cycle. For $n = 4$ and the initial state $x = (1, 1, 0, 0)$, the sequence of the most-significant bits becomes $X = (1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, \ldots)$ and is periodic with period 16. □

Most processors have either 32 or 64 bit words, and thus univariate mappings with $k = 1$ are not sufficient. Klimov and Shamir constructed invertible T-functions on multiword states whose iteration is guaranteed to yield a single cycle. One of their proposals was used in eSTREAM Phase 1 candidate Mir-1. Another class of single-cycle T-functions on multiword states named TSC was proposed by Hong *et al.* in [76], one of their construction was optimized for hardware implementations. In Chapter 7, we present powerful attacks on different proposals, and some cryptanalysis of eSTREAM Phase 2 candidate TSC-4.

2.6.3 Alternative Designs

There are also alternative and unique designs of stream ciphers such as RC4, which is used in the security applications WEP and SSL. The eSTREAM Phase 3 candidates Trivium and Salsa20 can also be mentioned here, although their design is strongly influenced by block ciphers. We present some cryptanalysis on these two eSTREAM candidates in Chapters 3, 8 and 9.

2.7 Attacks on Stream Ciphers

We describe some well-known attacks on stream ciphers, mainly on LFSR-based designs. Many of these attacks result in necessary conditions for the design of secure stream ciphers.

2.7.1 LFSR-synthesis

The *linear complexity* C of any binary sequence is defined by the length of the shortest LFSR that generates the sequence. Given at least $2C$ bits of a binary sequence with linear complexity C, the *Berlekamp-Massey* algorithm [93] determines an LFSR of length C in $\mathcal{O}(C^2)$ time. In the case of a filter generator with an LFSR of size n, the linear complexity of the keystream is at most $D_l := \sum_{i=0}^{l} \binom{n}{i}$, where $l := \deg(f)$ is the algebraic degree of the Boolean filter function f. With $2D_l$ bits of keystream, the remaining keystream can be synthesized in $\mathcal{O}(D_l^2)$. As a consequence, linear complexity of a stream cipher should be large.

2.7.2 Algebraic Attacks

Any stream cipher can be expressed as a system of multivariate algebraic equations, depending on the secret key and on the known keystream. The observed keystream can be substituted in this system, and the system can be solved to recover the secret key. These two steps (find equations and solve the system) are the principle of *algebraic attacks* [45, 95]. If the system corresponds to simultaneous equations in a large number of unknowns and of a complex (nonlinear) type, then solving the system is difficult. An overdefined nonlinear system could be linearized (where each monomial is replaced by a new variable) and solved by Gaussian elimination. The efficiency of the method depends on the algebraic degree of the equations.

Example 3. Consider the system of equations $x_0x_1 = 0$, $x_0 \oplus x_0x_1 = 1$, $x_1 \oplus x_0x_1 = 0$. With the new variables $y_0 = x_0$, $y_1 = x_1$, $y_2 = x_0x_1$, one obtains the linearized system $y_2 = 0$, $y_0 \oplus y_2 = 1$, $y_1 \oplus y_2 = 0$. Gaussian elimination yields $y_0 = 1$, $y_1 = 0$, $y_2 = 0$, which corresponds to the solution $x_0 = 1$, $x_1 = 0$. □

One could also use more sophisticated methods derived from Buchenberger's algorithm to compute a *Gröbner basis* of a polynomial ideal. The most efficient algorithm is Faugère's F_5 [56], an alternative method is XL [44]. More recently, SAT-solver have been used [12] (such as MiniSAT). However, the computational cost of these approaches is difficult to evaluate and strongly depends on the structure of the system, see [9].

In the case of filter generators, the basic equations are $z^t = f(L^t(x))$ with $x = (x_0, \ldots, x_{n-1})$ and for $t = 0, 1, 2, \ldots$. Notice that $L(x)$ and also $L^t(x)$ is linear in x for any t. Consequently, the degree of $f(L^t(x))$ corresponds to $l := \deg(f) \leq n$ for any t. The number of monomials of degree i is $\binom{n}{i}$, so the overall number of monomials of degree between 0 and l is $D_l := \sum_{i=0}^{l} \binom{n}{i}$ (which is about $\binom{n}{l}$ for $l < n/2$). If we use linearization, the number of variables is at most D_l, and time complexity to solve this system is about $\mathcal{O}(D_l^3)$ (where the exponent 3 is taken for matrix inversion). This is worse compared to LFSR-synthesis. The crucial idea of algebraic attacks on filter generators is to reduce the degree of the equations. Let us first recall the definition of an *annihilator*: a function g is called an annihilator of f, if $fg = 0$. Here is a simple example:

Example 4. The function $g = x_0 \oplus x_1$ of degree 1 is an annihilator of $f = x_0x_1$ of degree 2, hence $fg = 0$. □

Let g be an annihilator for f or $f \oplus 1$ of low degree d. In the case $fg = 0$, one can multiply $z^t = f(L^t(x))$ by $g(L^t(x))$ and obtains $g(L^t(x)) \cdot z^t = 0$. For $z^t = 1$, this is an equation of degree d. Similarly, for $(f \oplus 1)g = 0$, one obtains $g(L^t(x)) \cdot (z^t \oplus 1) = 0$ of degree d. The complexity of algebraic attacks can be summarized by:

1. **Relation search step**. Finding annihilators g of f or $f \oplus 1$ with low degree d (if these exist at all). The complexity of this step is roughly in $\mathcal{O}(D_k^3)$, for $D_k := \sum_{i=0}^{d} \binom{k}{i}$ and where $k \leq n$ is the fixed number of input variables to the filter function.
2. **Solving step**. With R linearly independent annihilators of degree d for f or $f \oplus 1$, a single output bit z^t can be used to set up (in average) $R/2$ equations in x at time t. The number of monomials in these equations is at most $D := \sum_{i=0}^{d} \binom{n}{i}$, hence by linearization, data complexity of conventional algebraic attacks becomes about $2D/R$, and time complexity $\mathcal{O}(D^3)$.

In general, the *algebraic immunity* $\mathcal{AI}$ of a Boolean function f is defined by the minimum degree d of an annihilator for f or $f \oplus 1$. In [45] it has been shown that for any function f with k-bit input vector, functions $g \neq 0$ and h exist, with $fg = h$ such that e and d are at most $\lceil k/2 \rceil$. This implies that $\mathcal{AI}(f) \leq \lceil k/2 \rceil$. For a function with maximum algebraic immunity, time complexity of algebraic attacks is only about the square root of simple linearization; data complexity is about the square root of simple linearization and of LFSR-synthesis. As a consequence, $\mathcal{AI}$ of a stream cipher should be large. There are sophisticated algorithms to determine $\mathcal{AI}$ of an arbitrary Boolean function [3, 53]. Recently, some theoretical work on constructions of Boolean functions with maximum $\mathcal{AI}$ was presented [48, 50, 29, 35]. Algebraic Attacks can also be applied to other LFSR-based designs. However, these attacks are more difficult if the update function L is nonlinear (*e.g.* for some T-functions constructions), as the degree of equations is increasing with t. In this work, an improvement of algebraic attacks is presented in Chapter 4.

2.7.3 Fast Algebraic Attacks

Fast algebraic attacks were introduced by Courtois in [41]. They were confirmed and improved later by Armknecht in [2] and Hawkes and Rose in [72]. A prior aim of fast algebraic attacks is to find a relation $fg = h$ with $e := \deg g$ small and $d := \deg h$ larger. The equation $z^t = f(L^t(x))$ of filter generators is multiplied by g such that $g(L^t(x)) \cdot z^t = h(L^t(x))$. In classical algebraic attacks, the degree d would necessarily lead to considering a number of unknowns of the order of $D := \sum_{i=0}^{d} \binom{n}{i}$. In fast algebraic attacks, one considers that the sequence of the functions $h(L^t(x))$ can be obtained as an LFSR with linear complexity D. One could use then the Berlekamp-Massey algorithm to eliminate all monomials of degree superior to e in the equations, such that eventually one only needs to solve a system in $E := \sum_{i=0}^{e} \binom{n}{i}$ unknowns. More precisely, one can precompute a linear combination $\bigoplus_{i=0}^{D} c_i \cdot h(L^{t+i}(x)) = 0$ with fixed coefficients $c_i \in \{0, 1\}$

for all t. This gives $\bigoplus_{i=0}^{D} c_i \cdot g(L^{t+i}(x)) \cdot z^{t+i} = 0$ of lower degree e. The complexity of fast algebraic attacks can be summarized in these four steps:

1. **Relation search step**. One searches for functions g and h of low degrees such that $fg = h$. For g and h of degrees e and d respectively, with associated values $D_k := \sum_{i=0}^{d} \binom{k}{i}$ and $E_k := \sum_{i=0}^{e} \binom{k}{i}$, such g and h can be found when they exist by solving a linear system with $D_k + E_k$ equations, and with complexity $\mathcal{O}((D_k + E_k)^3)$. Usually one considers $e < d$.
2. **Precomputation step**. In this step, one searches for particular linear relations which permit to eliminate monomials with degree greater than e in the equations. This step needs a sequence of $2D$ bits of stream and has a complexity of $\mathcal{O}(D \log^2(D))$ using a direct method presented in [72].
3. **Substitution step**. At this step, one eliminates the monomials of degrees greater than e. This step has a natural complexity in $\mathcal{O}(E^2 D)$ but using discrete Fourier transform, it is claimed in [72] that a complexity $\mathcal{O}(ED \log(D))$ can be obtained.
4. **Solving step**. One solves the system with E linear equations in $\mathcal{O}(E^3)$. Each equation requires D bits of keystream, hence data complexity is about $C_D = D + E$. This is not much larger than in algebraic attacks (with the same asymptotic complexity).

Notice that, for arbitrary non-zero functions f, g, h, the relation $fg = h$ implies $fh = h$, thus we have $d \geq \mathcal{AI}(f)$ and we can restrict to values e with $e \leq \mathcal{AI}(f)$. Fast algebraic attacks are always more efficient than conventional algebraic attacks if $d = \mathcal{AI}(f)$ and $e < d$. In case that e turns out to be large for this d, it is of interest to determine the minimum e where d is slightly larger than $\mathcal{AI}(f)$. In Chapter 3, we give efficient algorithms for the relation search step. A powerful variant of algebraic attacks on the filter generator was presented recently in [108]. In this work, a precomputation of complexity $\mathcal{O}(D_l \log^2(D_l))$ is used to find linear equations in the initial state variables (similar to the precomputation of FAA's). The initial state can be recovered after observing about D_l keystream bits with a complexity of $\mathcal{O}(D_l)$.

2.7.4 Correlation and Linear Attacks

In correlation and linear attacks one considers an overdefined system of linear input-output relations of some correlation (*i.e.* some noisy equations). In contrast, algebraic attacks deal with exact equations.

Correlation Attacks. The main scenario of correlation attacks are combination generators, assuming that the keystream bit z^t is correlated to one individual LFSR output sequence x_t due to the combining function, hence $\Pr(x_t = z^t) = p \neq 0.5$. Here is an example of bit-correlation:

Example 5. Consider a combination generator with 3 LFSR's and with combination function $f(x) = x_0x_1 \oplus x_0x_2 \oplus x_1x_2$. Then, the correlation of the output sequence of any LFSR to the keystream is $p = 0.75$. □

With a *divide-and-conquer* strategy, one can determine the initial state of the target LFSR first, given N bits of keystream. In the original correlation attack proposed by Siegenthaler [115], one checks all possible initial states of the target LFSR, and chooses the initial state with the minimum number of deviations to the observed keystream (assuming $p > 0.5$). The attack by Siegenthaler can be prevented by using a *correlation-immune* combining function. In this case, the keystream is statistically independent of the output of each constituent LFSR; any correlation attack should then consider several LFSR's together.

Fast Correlation Attacks. In [96], *fast correlation attacks* have been developed. These attacks are significantly faster than exhaustive search over all initial states of the target LFSR. Let us first review the statistical model: it is assumed that a linear combination of some LFSR outputs is correlated to the keystream. This linear combination corresponds to the output of a unique LFSR of size L. The output of this LFSR is a codeword of size N of the linear code of dimension L defined by the feedback polynomial. The keystream can be seen as the result of the transmission of this codeword through the binary symmetric channel with error probability $1 - p$, or $\Pr(x_t = z^t) = p \neq 0.5$. The attack aims at recovering L bits of the codeword from the knowledge of N bits of the keystream. With information theoretic arguments, the minimum number of required keystream bits to recover the initial state is $N = L/(1 - h(p))$, where h is the binary entropy function, but no efficient decoding algorithm is known for achieving this bound. The idea of the attack in [96] is to use parity-check equations (of degree $\leq N$) for the linear code, which can be derived from the feedback polynomial. Every bit x_t satisfies several parity-check equations, and by substituting the corresponding keystream digits z^t in these equations, we obtain relations for each bit z^t which may or may not hold. The probability bias of these equations is determined by p and by the weight w of the parity-check equations. To test whether $x_t = z^t$, one counts the number of all equations which hold for z^t. Under favorable conditions, one can find some bits z^t which have a high probability of being correct, which means that only slight modifications of the estimate are necessary to determine all bits of the LFSR. This main idea can be combined with a partial exhaustive search, and the attack can be extended to the situation where w is large: the precomputation step then consists in generating parity-check equations of low weight (*i.e.* finding polynomial multiples of the feedback polynomial of low weight) but large degree N, see *e.g.* [34,37]. Another idea is to use iterative decoding methods for the keystream bits z^t: assign to each bit z^t a new probability for $x_t = z^t$ conditioned on the number of relations satisfied. This procedure can be iterated, and after a few rounds, those bits z^t with small conditional probability are complemented, until we end up with the original LFSR sequence. The attacks were improved by a series of variant attacks [80].

Linear Attacks. The bit-correlations of correlation attacks can be viewed as a special case of *linear cryptanalysis* [94], which tries to take advantage of high probability occurrences of linear relations involving keystream bits and initial state bits. In general, the starting point is a system of linear relations in some of the initial state bits x which hold with

probabilities different from 1/2 for the observed keystream. The most likely solution for x is the one that satisfies the most relations (assuming that all relations have $p > 0.5$), see [15] for a recent application on eSTREAM candidate Grain-v0. Another approach is to use low-weight parity checks for the linear relations and apply iterative methods [64]. In Chapter 7, we present linear attacks on T-functions. An important measure for correlation and linear attacks is the *nonlinearity* of a Boolean function. In Chapter 4, we present an efficient method to find relations with small nonlinearity.

2.7.5 Differential Attacks

Differential cryptanalysis [24] is a general method of cryptanalysis that is applicable primarily to block ciphers, but also to stream ciphers and cryptographic hash functions. One investigates how a difference in the input of the cipher affects the difference in the output (requiring chosen plaintext). The difference is traced through the network of transformations F, discovering where the cipher exhibits non-random behavior. The goal is to find a suitable differential, *i.e.* a fixed input-difference Δ_x and a fixed output-difference Δ_z such that $\Delta_z = F(x) \oplus F(x \oplus \Delta_x)$ with high probability for a random input x. The differential can be exploited to distinguish the output with statistical methods (or to recover the key using more sophisticated variants). The statistical properties of the differential mainly depend on the nonlinear part of the cipher. Note that a fixed differential of first order reduces the algebraic degree of the output function by one. One method of differential attack on a stream cipher is to find a high-probability differential for the output function [23]. The known keystream allows computation of the output-difference, and the inputs of the initialization function (*i.e.* the key and the IV) should be chosen such that two states with the desired input-difference are produced. There are many specialized types of differential attacks. Collisions in hash functions correspond to differentials with zero output-difference.

2.7.6 Tradeoff Attacks

Tradeoff attacks are generic attacks, where a tradeoff in time, memory and data can be achieved to attack the stream cipher. During the precomputation phase, which requires P steps, the adversary explores the general structure of the stream cipher and summarizes her findings in large tables, requiring memory of size M. During the realtime phase, which requires T steps, the adversary is given D frames (*i.e.* data which corresponds to D different keystreams produced by unknown keys and IV's), and her goal is to use the tables to find the key of one frame as fast as possible. In [10], Babbage concludes that the internal state of the stream cipher should be at least twice as large as the key. Biryukov and Shamir presented some improved TMD tradeoffs in [25]. In [77, 52] they conclude that a TMD tradeoff attack can be mounted with P, D, T, M smaller than exhaustive key search, if the IV size is smaller than half the key size. Furthermore, an attack can be mounted with D, T, M smaller than exhaustive key search, but without restrictions on P, if the IV size is smaller than the key size.

2.8 Statistical Tests

In this section, we briefly describe optimal distinguishers and statistical key recovery attacks, and we describe the χ^2 test.

2.8.1 Optimal Distinguishers

Binary hypothesis testing is a formal way for distinguishing between two distributions. This is a frequently encountered problem in cryptanalysis. We use the well-known approach by Neyman-Pearson, and some advanced methods described in [11]. Let $X^N := (X_0, X_1, \ldots, X_{N-1})$ denote N *i.i.d.* random variables where each $X_i \in \mathcal{X}$ and $\mathcal{X}$ has cardinality m. We assume that the distribution of the random variables is either D_0 (the null hypothesis) or D_1 (the alternative hypothesis). Both distributions are assumed to be known. Let $x^N := (x_0, x_1, \ldots, x_{N-1})$ be a realization of the N random variables. Given x^N, the goal is to decide if the random variables have distribution D_0 or D_1. A (possibly computationally unbounded) algorithm $\mathcal{D}$ which takes as input a sequence of N realizations x^N distributed according to D_0 or D_1, and outputs 0 or 1 according to its decision, is called a *distinguisher*. It can be fully determined by an acceptance region $\mathcal{A} \subset \mathcal{X}$ such that $\mathcal{D}(x^N) = 1$ iff $x^N \in \mathcal{A}$. Note that $\mathcal{D}(X^N)$ is a derived random variable. The ability to distinguish a distribution from another is usually measured in terms of the *advantage* of the distinguisher and is defined by

$$\mathrm{Adv}_{\mathcal{D}} := \left|\Pr(\mathcal{D}(X^N) = 0|D_0) - \Pr(\mathcal{D}(X^N) = 0|D_1)\right| \ . \tag{2.2}$$

Hence, the distinguisher can make two types of errors: it can either output 0 when the distribution is D_1 (which is a false alarm if H_0 is the interesting event) or 1 when the distribution is D_0 (which is a non-detection); we will denote these respective error probabilities by $p_\alpha := \Pr(\mathcal{D}(X^N) = 0|D_1)$ and $p_\beta := \Pr(\mathcal{D}(X^N) = 1|D_0)$, and the overall error probability is defined as $p_e := \frac{1}{2}(p_\alpha + p_\beta)$. It is linked to the advantage by the simple relation $\mathrm{Adv}_{\mathcal{D}} = 1 - 2p_e$. The Neyman-Pearson Lemma derives an optimal test $\mathcal{D}$, *i.e.* a test which minimizes the error p_e for given N. It is based on the likelihood ratio LR with acceptance region

$$\mathcal{A} = \{x : \ \mathrm{LR}(x^N) \geq 1\} \text{ with } \mathrm{LR}(x^N) = \frac{\Pr(X^N = x^N|D_0)}{\Pr(X^N = x^N|D_1)} \ . \tag{2.3}$$

Let us now assume that the distributions D_0 and D_1 are close to each other, *i.e.* $\Pr(X = x|D_0) = \mu_x$ and $\Pr(X = x|D_1) = \mu_x + \varepsilon_x$ with probability bias $|\varepsilon_x| \ll \mu_x$ for all $x \in \mathcal{X}$. Baignères *et al.* introduced the distance $\Delta(D_0, D_1)$ between two close distributions. This measure is directly linked to the number of samples needed to distinguish both probability distributions with a good success probability. It is defined by

$$\Delta(D_0, D_1) := \sum_{x \in \mathcal{X}} \frac{\varepsilon_x^2}{\mu_x} \ . \tag{2.4}$$

The data complexity of an optimal distinguisher becomes $N = d/\Delta(D_0, D_1)$, where the real number d controls the overall probability of error. If Φ denotes the distribution function of the standard normal distribution, it is approximately $p_e \approx \Phi(-\sqrt{d}/2)$. In the case where D_1 is the uniform distribution, we use the notation $\Delta(D_0)$ instead of $\Delta(D_0, D_1)$ and have $\Delta(D_0) = |\mathcal{X}| \sum_x \varepsilon_x^2$. This measure is called the *squared Euclidean imbalance*. In the case $\mathcal{X} = \{0, 1\}$ we have $\varepsilon := \varepsilon_0 = -\varepsilon_1$ and one can see that $\Delta(D_0) = 4\varepsilon^2$ and N is proportional to ε^{-2}. It is a well accepted fact that the complexity of linear cryptanalysis is linked to the inverse of the square of the bias.

2.8.2 Key Recovery

According to [11], the framework of optimal distinguishers can be adapted to key recovery. Assume that one observes 2^n realizations x of size N each, where $2^n - 1$ realizations have uniform distribution D_1, and only one realization has non-uniform distribution D_0. The goal is then to identify the realization with distribution D_0. Such a situation could appear when the adversary wants to distinguish the correct subkey of n bits (assuming that only the correct subkey gives a distribution D_0). We consider the simple key ranking method where the rank of a subkey corresponds to the grade $\mathrm{LR}(x^N)$. The correct subkey has a high expected rank if

$$N = \frac{4n \log 2}{\Delta(D_0)} \ . \tag{2.5}$$

Then, the expected rank (starting by 1) becomes $1 + (2^n - 1)\Phi(-\sqrt{n\Delta(D_0)/2})$. This is a guess-and-determine attack: a subkey is guessed, and the correct one can be determined. The remaining bits of the key can then be found with a partial search (verify the observed keystream).

2.8.3 The Chi-Squared Test

The χ^2 test is used to distinguish an unknown distribution D_0 from the uniform distribution D_1, see *e.g.* [82]. Again, let $X^N := (X_0, X_1, \ldots, X_{N-1})$ denote N *i.i.d.* random variables where each $X_i \in \mathcal{X}$ and $\mathcal{X}$ has cardinality m. We assume that the distribution of the random variables is either D_0 (the null hypothesis) or the uniform distribution D_1 (the alternative hypothesis). Let $x^N = (x_0, x_1, \ldots, x_{N-1})$ be a realization, and N_x the number of observations x in x^N. Then, the χ^2 statistic is a random variable defined by

$$\chi^2 := \sum_x \frac{(N_x - N/m)^2}{N/m} \ . \tag{2.6}$$

For large N, the χ^2 statistic is compared with the threshold of the $\chi^2_{\alpha,m-1}$ distribution having $m - 1$ degrees of freedom and significance level α. Consequently, a χ^2 test can be defined by a threshold T, such that the alternative hypothesis is accepted if $\chi^2 < T$. If the random variables in X^N have uniform distribution D_1, then the expectation of χ^2 becomes $\mathrm{E}(\chi^2) = m - 1$. Now assume that the random variables in X^N have a non-uniform distribution D_0 with $\Pr(X = x|D_0) = 1/m + \varepsilon_x$, such that $|\varepsilon_x| \ll 1/m$ for all

$x \in \mathcal{X}$. With the definitions $\varepsilon^2 := \sum_x \varepsilon_x^2$ and $c := N\varepsilon^2$, the expectation of χ^2 becomes about $\mathrm{E}(\chi^2) = (c+1)m - 1$. The difference between the expectations becomes significant, if $c = \mathcal{O}(1)$. Consequently, about $N = 1/\varepsilon^2$ samples are required to distinguish a source with distribution D_1 from a source with uniform distribution D_0. Note that ε^2 differs from $\Delta(D_0)$ only by a factor of $|\mathcal{X}|$. It is well-known that a χ^2 cryptanalysis needs about $1/\Delta(D_0)$ queries to succeed, which is not worse (up to a constant term) than an optimal distinguisher. In fact, the χ^2 statistical test is asymptotically equivalent to a likelihood ratio test. Consequently, if one distribution is unknown, the best practical alternative to an optimal distinguisher seems to be a χ^2 attack.

Chapter 3

Algebraic Immunity against Fast Algebraic Attacks

In this chapter we propose several efficient algorithms for assessing the resistance of Boolean functions against fast algebraic attacks when implemented in LFSR-based stream ciphers. An efficient generic algorithm is demonstrated to be particularly efficient for symmetric Boolean functions. As an application, it is shown that large classes of symmetric functions are very vulnerable to fast algebraic attacks despite their proven resistance against conventional algebraic attacks.

3.1 Introduction

Resistance against fast algebraic attacks is not fully covered by algebraic immunity, as has been demonstrated, *e.g.* by a fast algebraic attack on the eSTREAM Phase 2 candidate SFINKS [42]. We will later give examples of functions which have optimal algebraic immunity but are very vulnerable against fast algebraic attacks. It seems therefore relevant to be able to efficiently determine the immunity of existing and newly constructed Boolean functions against fast algebraic attacks. For determining immunity against fast algebraic attacks, we give a new algorithm that compares favorably with the known algorithms [95, 53].

The algorithm is applied to several classes of Boolean functions with optimal algebraic immunity, including symmetric Boolean functions like the majority functions. Symmetric functions are attractive as the hardware complexity grows only linearly with the number of input variables. However, it is shown in this chapter that the specific structure of these functions can be exploited in a much refined algorithm. It is concluded that large classes of symmetric functions are very vulnerable to fast algebraic attacks despite their optimal algebraic immunity. A symmetric function would not be implemented by itself but rather in combination with other nonlinear components in stream ciphers. It seems nevertheless essential to know the basic cryptographic properties of each component used.

3.2 Efficient Computation of Immunity

Given a Boolean function $f(x) = \bigoplus_\alpha f_\alpha x^\alpha$ with n input variables, the goal is to decide whether $g(x) = \bigoplus_\beta g_\beta x^\beta$ of degree e and $h(x) = \bigoplus_\gamma f_\gamma x^\gamma$ of degree d exist, such that $fg = h$. The known function f is represented preferably by the truth table $T(f)$, which allows to efficiently access the required elements, and the unknown functions g and h are represented by the coefficient vectors $C(g)$ and $C(h)$, which leads to the simple side conditions $g_\beta = 0$ for $|\beta| > e$ and $h_\gamma = 0$ for $|\gamma| > d$. In order to decide if g and h exist, one has to set up a number of linear equations in g_β and h_γ. Such equations are obtained *e.g.* by evaluation of $f(x) \cdot \bigoplus_\beta g_\beta x^\beta = \bigoplus_\gamma h_\gamma x^\gamma$ for some values of x. There are $D + E$ variables with $D := \sum_{i=0}^{d} \binom{n}{i}$ and $E := \sum_{i=0}^{e} \binom{n}{i}$, so one requires at least the same number of equations. The resulting system of equations can be solved by Gaussian elimination with time complexity $\mathcal{O}((D+E)^3) = \mathcal{O}(D^3)$. If any $D + E$ equations are linearly independent, then no nontrivial g and h of corresponding degree exist. Otherwise, one may try to verify a nontrivial solution.

Certainly, there are more sophisticated algorithms, namely we are able to express a single coefficient h_γ as a linear combination of coefficients g_β. If these relations hold for any value of γ, one may choose γ with $|\gamma| > d$ such that $h_\gamma = 0$, in order to obtain relations in g_β only. Consequently, equations for coefficients of g can be completely separated from equations for coefficients of h. As there are only E variables g_β, one requires at least E equations, and the system of equations can be solved in $\mathcal{O}(E^3)$. Depending on the parameters n, d, e and on the structure of f, there are different strategies how to efficiently set up equations.

3.2.1 Setting up Equations

In this section, we consider the product $fg = h$ where f, g and h are arbitrary Boolean functions in n variables. Recall that $\alpha \subseteq \beta$ is an abbreviation for $\mathrm{supp}(\alpha) \subseteq \mathrm{supp}(\beta)$. In addition $\beta - \alpha$ denotes integer subtraction (which is equivalent to bitwise subtraction if $\alpha \subseteq \beta$). There is a well-known relation between elements of the truth table $T(f)$ and the coefficients vector $C(f)$ of a Boolean function f, which requires introduction of the following matrix:

Definition 5. *The Hadamard matrix H_N is an $N \times N$ matrix in $\mathbb{F}$, where the element of row i and column j (counting from 0) is defined by $\binom{i}{j} \mod 2$.*

Notice that H_N is a lower-triangular and self-inverse matrix. The following relation is given without proof:

Proposition 1. *For any Boolean function f with n input variables, one has $T(f) = H_{2^n} \cdot C(f)$ and $C(f) = H_{2^n} \cdot T(f)$.*

Prop. 1 can be replaced by an expression that is computed more efficiently.

Lemma 1. *Consider a Boolean function $f(x) = \bigoplus_\alpha f_\alpha x^\alpha$. Then it is $f(k) = \bigoplus_{\alpha \subseteq k} f_\alpha$ and $f_k = \bigoplus_{\alpha \subseteq k} f(\alpha)$.*

PROOF. Prop. 1 can be expressed as $f(k) = \bigoplus_\alpha \binom{k}{\alpha} f_\alpha$ and $f_k = \bigoplus_\alpha \binom{k}{\alpha} f(\alpha)$, where binomial coefficients are in modulo 2. Lucas' Theorem says that $\binom{\alpha}{\beta} = \prod \binom{\alpha_i}{\beta_i} \mod 2$, where α_i and β_i are the binary expression of α and β, respectively. Consequently, if $\binom{\alpha}{\beta} = 1 \mod 2$, then $\beta_i = 1 \Rightarrow \alpha_i = 1$ for all i, which is equivalent to $\beta \subseteq \alpha$. □

With the following theorem, we are able to express a single coefficient h_γ as a linear combination of coefficients g_β, where the linear combination is computed either with elements of the truth table $T(f)$, or with elements of the coefficient vector $C(f)$.

Theorem 1. *Let* $f(x) = \bigoplus_\alpha f_\alpha x^\alpha$ *and* $g(x) = \bigoplus_\beta g_\beta x^\beta$. *Set* $h(x) = \bigoplus_\gamma h_\gamma x^\gamma := f(x) \cdot g(x)$. *Then, for* $A_{\gamma,\beta} \in \mathbb{F}$, *we have for each* γ

$$h_\gamma = \bigoplus_{\beta \subseteq \gamma} A_{\gamma,\beta} \cdot g_\beta \tag{3.1}$$

$$A_{\gamma,\beta} := \bigoplus_{\beta \subseteq \alpha \subseteq \gamma} f(\alpha) = \bigoplus_{\gamma - \beta \subseteq \alpha \subseteq \gamma} f_\alpha \ . \tag{3.2}$$

PROOF. From Prop. 1 we have $f(k) = \bigoplus_{\alpha \subseteq k} f_\alpha$ and $f_k = \bigoplus_{\alpha \subseteq k} f(\alpha)$. We obtain the relation $h_\gamma = \bigoplus_{\alpha \subseteq \gamma} h(\alpha) = \bigoplus_{\alpha \subseteq \gamma} f(\alpha) g(\alpha)$. With $g(\alpha) = \bigoplus_{\beta \subseteq \alpha} g_\beta$, this becomes

$$h_\gamma = \bigoplus_{\alpha \subseteq \gamma} \left(\bigoplus_{\beta \subseteq \alpha} g_\beta f(\alpha) \right) = \bigoplus_{(\alpha,\beta):\beta \subseteq \alpha \subseteq \gamma} f(\alpha) g_\beta = \bigoplus_{\beta \subseteq \gamma} \left(\bigoplus_{\beta \subseteq \alpha \subseteq \gamma} f(\alpha) \right) g_\beta \ .$$

For the second expression, first observe that by definition we have $x^\alpha := x_0^{\alpha_0} \cdots x_{n-1}^{\alpha_{n-1}} = \prod_{i \in \mathrm{supp}(\alpha)} x_i$. As we can replace x_i^e by x_i for any $e \geq 1$, it holds $x^\alpha x^\beta = \prod_{i \in \mathrm{supp}(\alpha) \cup \mathrm{supp}(\beta)} x_i$. Then $h_\gamma x^\gamma = \bigoplus_{\mathcal{S}} f_\alpha x^\alpha g_\beta x^\beta$ with $\mathcal{S} = \{(\alpha, \beta) : \mathrm{supp}(\alpha) \cup \mathrm{supp}(\beta) = \mathrm{supp}(\gamma)\}$ and hence $h_\gamma = \bigoplus_{\beta \subseteq \gamma} \left(\bigoplus_{\mathcal{S}'} f_\alpha \right) g_\beta$ with $\mathcal{S}' = \{\alpha : \mathrm{supp}(\alpha) \cup \mathrm{supp}(\beta) = \mathrm{supp}(\gamma)\} = \{\alpha : \mathrm{supp}(\gamma) \setminus \mathrm{supp}(\beta) \subseteq \mathrm{supp}(\alpha) \subseteq \mathrm{supp}(\gamma)\}$, which is $\mathcal{S}' = \{\alpha : \mathrm{supp}(\gamma - \beta) \subseteq \mathrm{supp}(\alpha) \subseteq \mathrm{supp}(\gamma)\}$ as no carries occur in the subtraction for $\beta \subseteq \gamma$. We finally have $\mathcal{S}' = \{\alpha : \gamma - \beta \subseteq \alpha \subseteq \gamma\}$.□

Example 6. Let us set up an equation for $\gamma = (101)_2$. Eq. 3.1 brings out

$$\begin{aligned} h_{(101)} = {} & A_{(101),(000)} \cdot g_{(000)} \\ & \oplus A_{(101),(100)} \cdot g_{(100)} \\ & \oplus A_{(101),(001)} \cdot g_{(001)} \\ & \oplus A_{(101),(101)} \cdot g_{(101)} \ . \end{aligned}$$

If we use the Eq. 3.2 in order to determine the coefficients $A_{i,j}$, we find

$$\begin{aligned} A_{(101),(000)} &= f_{(101)} = f(000) \oplus f(100) \oplus f(001) \oplus f(101) \\ A_{(101),(100)} &= f_{(001)} \oplus f_{(101)} = f(100) \oplus f(101) \\ A_{(101),(001)} &= f_{(100)} \oplus f_{(101)} = f(001) \oplus f(101) \\ A_{(101),(101)} &= f_{(000)} \oplus f_{(100)} \oplus f_{(001)} \oplus f_{(101)} = f(101) \ . \end{aligned}$$

□

Remark 1. Let us introduce the vector $B_{\gamma,\beta} \in \mathbb{F}^{2^n}$, where the i'th element of $B_{\gamma,\beta}$ is defined by $\binom{\gamma}{i}\binom{i}{\beta}$. Then, according to Lucas' Theorem, the coefficient $A_{\gamma,\beta}$ can also be written as a scalar product $A_{\gamma,\beta} = B_{\gamma,\beta} \cdot T(f) = B_{\gamma,\gamma-\beta} \cdot C(f)$.

3.2.2 Determining the Existence of Solutions

Based on Th. 1, we propose Alg. 1 to determine if g and h exist, given f and the corresponding degrees e and d.

Algorithm 1 Determine the existence of g and h for any f

Input: A Boolean function f with n input variables and two integers $0 \leq e \leq \mathcal{AI}(f)$ and $\mathcal{AI}(f) \leq d \leq n$.

Output: Determine if g of degree at most e and h of degree at most d exist such that $fg = h$.

1: Initialize an $E \times E$ matrix G, and let each entry be zero.
2: Compute an ordered set $\mathcal{I} \leftarrow \{\beta : |\beta| \leq e\}$.
3: **for** i from 1 to E **do**
4: Choose a random γ with $|\gamma| = d + 1$.
5: Determine the set $\mathcal{B} \leftarrow \{\beta : \beta \subseteq \gamma,\ |\beta| \leq e\}$.
6: **for all** β in $\mathcal{B}$ **do**
7: Let the entry of G in row i and column β (in respect to $\mathcal{I}$) be $A_{\gamma,\beta}$
8: **end for**
9: **end for**
10: Solve the linear system of equations, and output `no` g `and` h `of corresponding degree` if there is only a trivial solution.

Let us discuss functionality and complexity of Alg. 1. According to Eq. 3.1, for each choice of γ one can set up an equation that depends on a linear combination of h_γ and $\{g_\beta : \beta \subseteq \gamma\}$. We make use of the side conditions $h_\gamma = 0$ for $|\gamma| > d$, and $g_\beta = 0$ for $|\beta| > e$. This can be used to simplify the equations: with a choice of γ such that $|\gamma| > d$, one obtains an equation that depends only on a linear combination of $\{g_\beta : \beta \in \mathcal{B}\}$ with $\mathcal{B} := \{\beta : \beta \subseteq \gamma,\ |\beta| \leq e\}$. In order to compute the linear combination, one has to compute the coefficients $A_{\gamma,\beta}$ for each $\beta \in \mathcal{B}$. Notice that with Eq. 3.2, $A_{\gamma,\beta}$ has $2^{|\gamma|-|\beta|}$ addends $f(\alpha)$, where $\{\beta : \beta \subseteq \gamma,\ |\beta| = b\}$ has cardinality $\binom{|\gamma|}{b}$. Consequently, a single equation is set up in $\sum_{b=0}^{e}\binom{|\gamma|}{b}2^{|\gamma|-b}$ steps, assuming that $f(\alpha)$ can be accessed in negligible time. As there are E variables g_β, one requires at least E equations in order to make a statement about the (non-) existence of nontrivial g (the equations are linearly independent with high probability). The coefficients $A_{\gamma,\beta}$ of the equations are stored in an $E \times E$ matrix G, initialization takes $\mathcal{O}(E^2)$ time and memory. This requires an ordered set $\mathcal{I} \leftarrow \{\beta : |\beta| \leq e\}$, indicating the order of g_β in G; it can be precomputed in $\mathcal{O}(E)$ time. The complexity to set up a single equation increases with $|\gamma|$ (where we required $|\gamma| > d$), so one may choose $|\gamma|$ as small as possible. Let $|\gamma| = d + 1$ for each equation, which allows to set up a number of $\binom{n}{d+1}$ equations. In the case of $e \ll d$ and $d \approx n/2$ (which is the typical scope of fast algebraic attacks), one has $E < \binom{n}{d+1}$, so it is sufficient

to set up equations with $|\gamma| = d + 1$ only. The overall time complexity to set up E equations becomes

$$C'_T = E \sum_{b=0}^{e} \binom{d+1}{b} 2^{d+1-b} \; . \tag{3.3}$$

Time complexity of the final step of Alg. 1 is $\mathcal{O}(E^3)$, hence overall time complexity of Alg. 1 is $C_T = C'_T + \mathcal{O}(E^3)$. For the specified range of parameters, one has $C'_T < E(e+1)\binom{d+1}{e}2^{d+1}$. Furthermore, $D \approx 2^{n-1} \approx 2^{2d-1}$, or $\sqrt{D} \approx 2^d$ and hence $C'_T \in \mathcal{O}(e\binom{d+1}{e}\sqrt{D}E)$.

Proposition 2. *The arithmetic complexity of Alg. 1 to determine the existence of g and h for any f corresponds to $C_T \in \mathcal{O}(e\binom{d+1}{e}\sqrt{D}E + E^3)$, provided that $e \ll d$ and $d \approx n/2$.*

Compared to the complexity $\mathcal{O}(D^3)$ of Alg. 2 in [95], Alg. 1 is very efficient for g of low degree, see Fig. 3.1 and the following example.

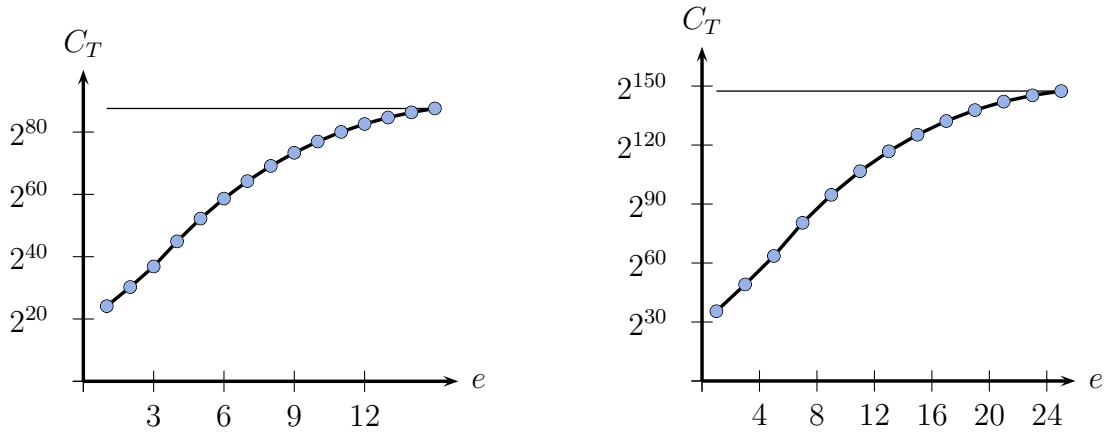

Figure 3.1: Time complexity of Alg. 1 for $n = 30$ (left) and $n = 50$ (right) with $d = n/2$. The solid line indicates the complexity D^3 of the trivial algorithm.

Example 7. Consider the majority function f with $n = 5$ inputs. As $\mathcal{AI}(f) = 3$, it may be interesting to find g and h with $e = 1$ and $d = 3$. The function g has $E = 6$ coefficients g_β with $|\beta| \leq e$, and we will setup equations for all γ with $|\gamma| > d$:

$$\begin{aligned} \beta \in &\{(00000), (00001), (00010), (00100), (01000), (10000)\} \\ \gamma \in &\{(11110), (11101), (11011), (10111), (01111), (11111)\} \; . \end{aligned}$$

To construct the matrix G, one has to compute the coefficients $A_{\gamma,\beta}$ for all combinations of γ and β (if $\beta \subseteq \gamma$), see also Ex. 6. One obtains (in respect to the above order of β):

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \; .$$

The kernel space has dimension 4, a possible basis of the kernel are the vectors

$$\{(0,1,1,0,0,0),(0,1,0,1,0,0),(0,1,0,0,1,0),(0,1,0,0,0,1)\}\ .$$

This corresponds to the monomials $x_0 \oplus x_1$, $x_0 \oplus x_2$, $x_0 \oplus x_3$, $x_0 \oplus x_4$ respectively. Any linear combination of these monomials is a solution for g. For example, with $g(x) = x_0 \oplus x_1$, and $f(x) = x_0x_1x_2 \oplus x_0x_1x_3 \oplus x_0x_1x_4 \oplus x_0x_2x_3 \oplus x_0x_2x_4 \oplus x_0x_3x_4 \oplus x_1x_2x_3 \oplus x_1x_2x_4 \oplus x_1x_3x_4 \oplus x_2x_3x_4 \oplus x_0x_1x_2x_3 \oplus x_0x_1x_2x_4 \oplus x_0x_1x_3x_4 \oplus x_0x_2x_3x_4 \oplus x_1x_2x_3x_4$ one has indeed $f(x) \cdot g(x) = (x_0 \oplus x_1) \cdot (x_2x_3 \oplus x_2x_4 \oplus x_3x_4)$, which is of degree 3. □

This interesting example will be investigated in more detail in Sect. 3.3 for symmetric functions.

3.2.3 Experimental Results

In [48], a class of (non-symmetric) Boolean functions f with maximum algebraic immunity is presented; these functions will be referred here as DGM functions. Application of Alg. 1 on their examples for $n = 5, 6, 7, 8, 9, 10$ reveals that h and g exist with $d = \mathcal{AI}(f) = \lceil n/2 \rceil$ and $e = 1$. We point out that this is the most efficient situation for a fast algebraic attack. Explicit functions g with corresponding degree are also obtained by Alg. 1, see Tab. 3.1 (where we show one single example of g, and dim denotes the dimension of the solution space for g of degree e). A formal expansion of $f(x) \cdot g(x)$ was performed to verify the results. A reaction on this attack is presented in [49].

Table 3.1: Degrees of the functions h and g for DGM functions f with n input variables.

n	$\deg f$	$\deg h$	$\deg g$	g	dim
5	4	3	1	$1 \oplus x_3$	4
6	4	3	1	$1 \oplus x_5$	4
7	5	4	1	$1 \oplus x_3 \oplus x_4$	1
8	5	4	1	$1 \oplus x_4 \oplus x_5$	1
9	8	5	1	$x_3 \oplus x_4 \oplus x_5 \oplus x_6$	1
10	8	6	1	$x_4 \oplus x_5 \oplus x_6 \oplus x_7$	1

3.3 Immunity of Symmetric Functions

In this section, we present a general analysis of the resulting system of equations for symmetric functions and propose a generic and a specific algorithm in order to determine the existence of g and h of low degrees.

3.3.1 Setting up Equations

Consider the case that $f(x)$ is a symmetric Boolean function. This means that $f(x) = f(x_0, \ldots, x_{n-1})$ is invariant under changing the order of the variables x_i. Therefore, we have $f(y) = f(y')$ if $|y| = |y'|$ and we can identify f with its (abbreviated) truth table $T^s(f) := (f^s(0), \ldots, f^s(n)) \in \mathbb{F}^{n+1}$ where $f^s(i) := f(y)$ for a y with $|y| = i$. Let $\sigma_i(x) := \bigoplus_{|\alpha|=i} x^\alpha$ denote the elementary symmetric polynomial of degree i. Then, each symmetric function f can be expressed by $f(x) = \bigoplus_i f_i^s \sigma_i(x)$ with $f_i^s \in \mathbb{F}$. Similarly to the non-symmetric case, f can be identified with its (abbreviated) coefficient vector $C^s(f) := (f_0^s, \ldots, f_n^s) \in \mathbb{F}^{n+1}$. Again, a relationship between $C^s(f)$ and $T^s(f)$ does exist:

Proposition 3. *For any symmetric Boolean function f with n input variables, one has $T^s(f) = H_{n+1} \cdot C^s(f)$ and $C^s(f) = H_{n+1} \cdot T^s(f)$.*

One can derive a much simpler relation for the coefficients h_γ in the case of symmetric functions f. Notice that in general, g and h are not symmetric.

Corollary 1. *Let $f(x) = \bigoplus_{i=0}^n f_i^s \sigma_i(x)$ be a symmetric function and $g(x) = \bigoplus_\beta g_\beta x^\beta$. Set $h(x) = \bigoplus_\gamma h_\gamma x^\gamma := f(x) \cdot g(x)$. Then, for $A_{i,j}^s \in \mathbb{F}$, we have for each γ*

$$h_\gamma = \bigoplus_{\beta \subseteq \gamma} A^s_{|\gamma|,|\beta|} \cdot g_\beta \tag{3.4}$$

$$A^s_{i,j} = \bigoplus_k \binom{i-j}{i-k} f^s(k) = \bigoplus_k \binom{j}{i-k} f_k^s \ . \tag{3.5}$$

Proof. Notice that Th. 1 holds for any function f, including symmetric functions. As $f(\alpha) = f(\alpha')$ for $|\alpha| = |\alpha'|$, we can rewrite the expressions from Th. 1 to

$$h_\gamma = \bigoplus_{\beta \subseteq \gamma} \left(\bigoplus_{\beta \subseteq \alpha \subseteq \gamma} f(\alpha) \right) g_\beta = \bigoplus_{\beta \subseteq \gamma} \left(\bigoplus_k \bigoplus_{\beta \subseteq \alpha \subseteq \gamma, |\alpha|=k} f^s(k) \right) g_\beta \ ,$$

$$h_\gamma = \bigoplus_{\beta \subseteq \gamma} \left(\bigoplus_{\gamma - \beta \subseteq \alpha \subseteq \gamma} f_\alpha \right) g_\beta = \bigoplus_{\beta \subseteq \gamma} \left(\bigoplus_k \bigoplus_{\gamma - \beta \subseteq \alpha \subseteq \gamma, |\alpha|=k} f_k^s \right) g_\beta \ .$$

Thus, to prove the claim it is enough to show that $|\{\alpha : \beta \subseteq \alpha \subseteq \gamma, |\alpha| = k\}| = \binom{|\gamma|-|\beta|}{|\gamma|-k}$ and $|\{\alpha : \gamma - \beta \subseteq \alpha \subseteq \gamma, |\alpha| = k\}| = \binom{|\beta|}{|\gamma|-k}$. For the first claim, observe that for any α with $\beta \subseteq \alpha \subseteq \gamma$ it holds that

$$\alpha_i = \begin{cases} 0, & \gamma_i = 0 \\ 1, & \gamma_i = 1, \beta_i = 1 \\ *, & \gamma_i = 1, \beta_i = 0 \end{cases}$$

where $'*'$ denotes that the value can be 0 or 1. This shows that $|\gamma| - |\beta|$ entries of α can be freely chosen if α can have an arbitrary weight. Because of $\beta \subseteq \alpha$, it holds that

$|\alpha| \geq |\beta|$. To ensure that $|\alpha| = k$, exactly $k - |\beta|$ of the freely selectable entries must be equal to 1. Thus, the number of combinations is $\binom{|\gamma|-|\beta|}{k-|\beta|} = \binom{|\gamma|-|\beta|}{|\gamma|-k}$ which shows the first claim. The proof for the second claim is similar. To fulfill $\gamma - \beta \subseteq \alpha \subseteq \gamma$, it must holds for the entries of α that

$$\alpha_i = \begin{cases} 0, & \gamma_i = 0 \\ *, & \gamma_i = 1, \beta_i = 1 \\ 1, & \gamma_i = 1, \beta_i = 0 \end{cases} .$$

Because of $\beta \subseteq \gamma$ by assumption, exactly $|\beta|$ entries of α are not fixed. To ensure that $|\alpha| = k$, exactly $k-(|\gamma|-|\beta|)$ of the freely selectable entries must be equal to 1. Therefore, the number of combinations is $\binom{|\beta|}{k-(|\gamma|-|\beta|)} = \binom{|\beta|}{|\gamma|-k}$ which shows the second claim. □

Example 8. Let us set up an equation for $\gamma = (101)_2$. It is $|\gamma| = 2$, and Eq. 3.4 brings out

$$\begin{aligned} h_{(101)} &= A^s_{2,0} \cdot (g_{(000)}) \\ &\oplus A^s_{2,1} \cdot (g_{(100)} \oplus g_{(001)}) \\ &\oplus A^s_{2,2} \cdot (g_{(101)}) \ . \end{aligned}$$

If we use Eq. 3.5 in order to determine the coefficients $A^s_{i,j}$, we find

$$\begin{aligned} A^s_{2,0} &= f^s_2 = f^s(0) \oplus f^s(2) \\ A^s_{2,1} &= f^s_1 \oplus f^s_2 = f^s(1) \oplus f^s(2) \\ A^s_{2,2} &= f^s_0 \oplus f^s_2 = f^s(2) \ . \end{aligned}$$ □

Remark 2. Let us introduce the vector $B^s_{i,j} \in \mathbb{F}^{n+1}$, where the k'th element of $B^s_{i,j}$ is defined by $\binom{i-j}{i-k}$. Then, the coefficient $A^s_{i,j}$ can also be written as a scalar product $A^s_{i,j} = B^s_{i,j} \cdot T^s(f) = B^s_{i,i-j} \cdot C^s(f)$.

3.3.2 Determining the Existence of Solutions

Given a symmetric function f, the existence of g and h with corresponding degrees can be determined by an adapted version of Alg. 1 (which will be referred as Alg. 1S): in step 7, the coefficient $A_{\gamma,\beta}$ is replaced by $A^s_{|\gamma|,|\beta|}$. The discussion of this slightly modified algorithm is similar to Sect. 3.2.2. However, computation of $A^s_{i,j}$ requires only $n+1$ evaluations of the function f, which can be neglected in terms of complexity. Consequently, time complexity to set up equations is only about $\mathcal{O}(E^2)$, and overall complexity of Alg. 1S becomes $\mathcal{O}(E^3)$.

Next, we will derive a method of very low (polynomial) complexity to determine the existence of g and h of low degree for a symmetric function f, but with the price that the method uses only sufficient conditions (*i.e.* some solutions may be lost). More precisely, we constrict ourselves to homogeneous functions g of degree e (*i.e.* g contains monomials of degree e only, $g_\beta = 0$ for all β with $|\beta| \neq e$), and Eq. 3.4 becomes

$$h_\gamma = A^s_{|\gamma|,e} \bigoplus_{|\beta|=e;\beta\subseteq\gamma} g_\beta \ . \tag{3.6}$$

Remember that $h_\gamma = 0$ for $|\gamma| > d$, so the homogeneous function g is determined by the corresponding system of equations for all γ with $|\gamma| = d+1, \ldots, n$. In this system, the coefficient $A^s_{|\gamma|,e}$ is constant for $\binom{n}{|\gamma|}$ equations. If $A^s_{|\gamma|,e} = 0$, then all these equations are linearly dependent (*i.e.* of type $0 = 0$). On the other hand, if $A^s_{|\gamma|,e} = 1$, then a number of $\binom{n}{|\gamma|}$ additional equations is possibly linearly independent. Consequently, if the sum of all possibly linearly independent equations for $|\gamma| = d+1, \ldots, n$ is smaller than the number of variables $\binom{n}{e}$, Eq. 3.6 gives an underdefined system of linear equations which must have a nontrivial (and homogeneous) solution g. This sufficient criterion is formalized by

$$\sum_{i=d+1}^{n} A^s_{i,e} \cdot \binom{n}{i} < \binom{n}{e} . \tag{3.7}$$

Given some degree e, the goal is to find the minimum value of d such that Eq. 3.7 holds. This can be done incrementally, starting from $d = n$. We formalized Alg. 2 of polynomial complexity $\mathcal{O}(n^3)$. This algorithm turned out to be very powerful (but not necessarily optimal) in practice, see Sect. 3.3.4 for some experimental results.

Algorithm 2 Determine the degrees of g and h for symmetric f

Input: A symmetric Boolean function f with n input variables.
Output: Degrees of specific homogeneous functions g and h such that $fg = h$.

1: **for** e from 0 to $\lceil n/2 \rceil$ **do**
2: Let $d \leftarrow n$, number of equations $\leftarrow 0$, number of variables $\leftarrow \binom{n}{e}$.
3: **while** number of equations $<$ number of variables **and** $d + 1 > 0$ **do**
4: Compute $A \leftarrow A^s_{d,e}$.
5: Add $A \cdot \binom{n}{d}$ to the number of equations.
6: $d \leftarrow d - 1$.
7: **end while**
8: Output $\deg g = e$ and $\deg h = d + 1$.
9: **end for**

For a specified class of symmetric Boolean functions f, it is desirable to prove some general statements concerning the degrees of g and h for any number of input variables n. In the next section, we apply technique based on Alg. 2 in order to prove a theorem for the class of majority functions.

3.3.3 Fast Algebraic Attacks on the Majority Function

We denote by f the symmetric Boolean majority function with $n \geq 2$ input variables, defined by $f^s(i) := 0$ if $i \leq \lfloor n/2 \rfloor$ and $f^s(i) := 1$ otherwise. For example, $T^s(f) := (0, 0, 1)$ for $n = 2$, and $T^s(f) := (0, 0, 1, 1)$ for $n = 3$. The algebraic degree of this function is $2^{\lfloor \log_2 n \rfloor}$. In [29] and [50], it could be proven independently that f has maximum algebraic immunity.[1] First, we show that the coefficients $A^s_{|\gamma|,e}$ from Corollary 1 have a simple form in the case of the majority function.

[1] It has been proved in [107] that the majority function is the only symmetric function with maximum $\mathcal{AI}$ in odd number of variables.

Lemma 2. *Let f be the majority function in n inputs and $d = \lfloor n/2 \rfloor + 1$. Then for $d' < d$ one has $A^s_{d',e} = 0$, and for $d' \geq d$, one has*

$$A^s_{d',e} = \binom{d'-e-1}{d-e-1} . \tag{3.8}$$

Proof. Recall that $f(k) = 0$ for $k < d$. Hence, due to Corollary 1 it is

$$A^s_{d',e} \overset{\text{Cor. } 1}{=} \bigoplus_{k=0}^{n} \binom{d'-e}{k-e} \cdot f(k) = \bigoplus_{k=d}^{n} \binom{d'-e}{k-e} = \bigoplus_{k=d}^{d'} \binom{d'-e}{k-e} .$$

The last equation is true because of $\binom{a}{b} = 0 \mod 2$ for $a < b$. This shows that $A^s_{d',e} = 0$ for $d' < d$. Now, let $d' \geq d \geq 1$. Then, $A^s_{d',e}$ can be simplified to

$$\begin{aligned}
A^s_{d',e} &= \bigoplus_{k=d}^{d'} \binom{d'-e}{k-e} = \bigoplus_{k=d}^{d'} \binom{d'-e-1}{k-e} \oplus \bigoplus_{k=d}^{d'} \binom{d'-e-1}{k-e-1} \\
&= \bigoplus_{k=d}^{d'-1} \binom{(d'-1)-e}{k-e} \oplus \bigoplus_{k=d}^{d'} \binom{(d'-1)-e}{(k-1)-e} \\
&= A^s_{d'-1,e} \oplus \bigoplus_{k=d-1}^{d'-1} \binom{(d'-1)-e}{k-e} \\
&= 2 \cdot A^s_{d'-1,e} \oplus \binom{d'-e-1}{d-e-1} = \binom{d'-e-1}{d-e-1} .
\end{aligned}$$

□

The main goal of this section is to prove the following theorem, which discloses the properties of f (and related functions) with respect to fast algebraic attacks.

Theorem 2. *Consider the majority function f with any $n \geq 2$ input variables, defined by $f^s(i) := 0$ if $i \leq \lfloor n/2 \rfloor$ and $f^s(i) := 1$ otherwise. Then there exist Boolean functions g and h such that $fg \equiv h$, where $\deg h = \lfloor n/2 \rfloor + 1$ and $\deg g = d - 2^j$, and where $j \in \mathbb{N}^0$ is maximum so that $\deg g > 0$. If n is even, the vector space of solutions g has a dimension $\geq \binom{n}{e} - \binom{n}{e-2}$, and in the case of n odd a dimension $\geq \binom{n}{e} - \binom{n}{e-1}$.*

Proof. Consider $fg = h$, where f is the majority function, $\deg g = e$ and $\deg h = d$. Our strategy is to set e and d to values which guarantee that Eq. 3.7 is satisfied, so we have to analyze the number of non-zero coefficients $A^s_{d',e} = \binom{d'-e-1}{d-e-1} \mod 2$ from Lemma 2 for these values, and determine the size of the left side in Eq. 3.7. Now, let $e := d - 2^j$ where j is chosen maximum such that $e \geq 1$. Observe that $d - e - 1 = 2^j - 1 = \sum_{i=0}^{j-1} 2^i$. For $a = \sum a_i 2^i$, by Lucas' Theorem, it holds that $\binom{a}{d-e-1} = \prod_{i=0}^{j-1} \binom{a_i}{1} \cdot \prod_{i=j}^{n} \binom{a_i}{0} \mod 2$. Note that $\binom{a}{b} = 0 \mod 2$ iff at least one term $\binom{0}{1} = 0$ exists in the product. This shows that $\binom{a}{d-e-1} \mod 2 = 1$ if and only if $a_i = 1$ for $i = 0, \ldots, j-1$, or, equivalently,

$$\begin{aligned} a &= \left(\sum_{i=0}^{j-1} 2^i\right) + \left(\sum_{i=j}^{n} a_i \cdot 2^i\right) = \left(2^j - 1\right) + \left(\sum_{i=j}^{n} a_i \cdot 2^i\right) \\ &= \left(2^j - 1\right) + 2^j \cdot \left(\sum_{i=j}^{n} a_i \cdot 2^{i-j}\right) . \end{aligned}$$

Hence, $A^s_{d+i,e} = \binom{d-e-1+i}{d-e-1} \mod 2 = \binom{(2^j-1)+i}{2^j-1} \mod 2 = 1$ if and only if i is a multiple of $2^j = d - e$. In other words, if $d = \lfloor n/2 \rfloor + 1$ and if $e = \lfloor n/2 \rfloor + 1 - 2^j$, only equations in Eq. 3.6 with $d' = d + k \cdot (d - e)$ and $k \geq 1$ impose conditions on the coefficients g_β, whereas the others are necessarily equal to zero. For $k = 1$, we have $A_{2d-e,e} = 1$. We will prove now that $d + k \cdot (d - e) > n$ for $k = 2$. The consequence is that only the coefficients $A_{d',e}$ for $d' = 2d - e$ are equal to 1. By the definition of j, it holds that

$$\lfloor n/2 \rfloor + 1 - 2^{j+1} \leq 0 \Leftrightarrow 2^{j+1} \geq \lfloor n/2 \rfloor + 1 \Leftrightarrow 2(d - e) \geq d \Leftrightarrow d - e \geq e \ .$$

Notice that $2d - e = 2\lfloor n/2 \rfloor + 2 - e$, which is $n + 2 - e$ for n even, and $n + 1 - e$ for n odd (and which is at least $d + 1$). With $2d - e \geq n + 1 - e$ and $d - e \geq e$, we find $d + 2 \cdot (d - e) = (2d - e) + (d - e) \geq (n + 1 - e) + e > n$. Altogether, the number of non-trivial equations in Eq. 3.6 is $\binom{n}{2d-e}$ which is equal to $\binom{n}{e-2}$ for n even and $\binom{n}{e-1}$ for n odd. In both cases, this value is less than $\binom{n}{e}$, the number of coefficients g_β. Consequently, the system of equations of Eq. 3.6 is underdefined and non-trivial solutions for g exist. Further on, one sees that the dimension of the solution space is at least $\binom{n}{e} - \binom{n}{e-2}$ for n even and at least $\binom{n}{e} - \binom{n}{e-1}$ for n odd. □

Algebraic and fast algebraic attacks are invariant with regard to binary affine transformations in the input variables. Consequently, Th. 2 is valid for all Boolean functions which are derived from the majority function by means of affine transformations. We notice that such a class of functions was proposed in a recent paper, discussing design principles of stream ciphers [27, 28]. For values $n = 2, 3, 4, 6$ only, Th. 2 is not meaningful. A very interesting subcase is $n = 2^{j+1}$ and $n = 2^j + 1$ for $j \geq 2$, for which Boolean functions g with $\deg g = 1$ exist.

Table 3.2: Degrees of the functions g and h (from $fg = h$) for dimension n, according to Th. 2.

n	5	6	7	8	9	10	11	12	13	14	15	16
$\deg g$	1	2	2	1	1	2	2	3	3	4	4	1
$\deg h$	3	4	4	5	5	6	6	7	7	8	8	9

3.3.4 Experimental Results

Application of Alg. 1S reveals that Th. 2 is optimal for the majority function where $d = \lfloor n/2 \rfloor + 1$ (verification for $n = 5, 6, \ldots, 16$). An explicit homogeneous function g can be constructed according to $g(x) = \prod_{i=0}^{e-1}(x_{2i} + x_{2i+1})$. We verified that Alg. 2 can discover the solutions of Th. 2.

In [29], a large pool of symmetric Boolean functions with maximum algebraic immunity is presented (defined for n even). One of these functions is the majority function, whereas the other functions are nonlinear transformations of the majority function. Application of Alg. 2 brings out that Th. 2 is valid for all functions f (verification for $n = 6, 8, \ldots, 16$). For some functions f, Alg. 2 finds better solutions than predicted by Th. 2 (*e.g.* for $T^s(f) := (0, 0, 0, 1, 1, 0, 1)$ where $d = 3$ and $e = 1$), which means that Th. 2 is not optimal for all symmetric functions. All solutions found by Alg. 2 can be constructed according to the above equation. Furthermore, Alg. 1S finds a few solutions which are (possibly) better than predicted by Alg. 2 (*e.g.* for $T^s(f) := (0, 0, 0, 1, 1, 1, 0)$ where $d = 3$ and $e = 2$), which means that Alg. 2 is not optimal for all symmetric functions.

3.4 Summary

In this chapter, several efficient algorithms are derived to compute the algebraic immunity of Boolean functions against fast algebraic attacks. Here, we focus on fast algebraic attacks related to filtered registers, although the algorithms may be used in different contexts. We described very fast algorithms for symmetric functions and proved that symmetric functions do not seem to be very secure in the context of a filtered register. For non-symmetric functions, our algorithm is very efficient when the degree of g is small.

Chapter 4

Algebraic Immunity of Augmented Functions

In this chapter, the algebraic immunity of S-boxes and augmented functions of stream ciphers is investigated. Augmented functions are shown to have some algebraic properties that are not covered by previous measures of immunity.

4.1 Introduction

In the previous chapter, it turned out in some cases that large $\mathcal{AI}$ did not help to prevent fast algebraic attacks (FAA's). It is an open question if immunity against FAA's is a sufficient criterion for any kind of algebraic attacks on stream ciphers. In the case of block ciphers, the algebraic immunity of S-boxes is a measure for the complexity of a very general type of algebraic attacks, considering implicit or conditional equations [46, 4]. Present methods for computation of $\mathcal{AI}$ of S-boxes are not very efficient, only about $n = 20$ variables are computationally feasible (except for power mappings, see [103, 43]).

In this chapter, we integrate the general approach for S-boxes in the context of stream ciphers and generalize the concept of algebraic immunity of stream ciphers, see Open Problem 7 in [33]. More precisely, we investigate conditional equations for augmented functions of stream ciphers and observe some algebraic properties (to be used in an attack), which are not covered by the previous definitions of $\mathcal{AI}$. As a consequence, immunity against FAA's is not sufficient to prevent any kind of algebraic attack: Depending on the Boolean functions used in a stream cipher, we demonstrate that algebraic properties of the augmented function allow for attacks which need much less known output than established algebraic attacks. This induces some new design criteria for stream ciphers. Time complexity of our attacks is derived by intrinsic properties of the augmented function. Our framework can be applied to a large variety of situations. We present two applications (which both have been implemented). First, we describe efficient attacks on some filter generators. For example, we can efficiently recover the state of a filter generator based on certain Boolean functions when an amount of output data is available which is only linear in the length of the driving LFSR. This should be compared to the data complexity of conventional algebraic attacks, which is about $\binom{n}{e}$, where n is the length of the LFSR and e equals the algebraic immunity of the filter function. Our investigation of the

augmented function allows to contribute to open problems posed in [57], and explains why algebraic attacks using Gröbner bases against filter generators are in certain cases successful even for a known output segment only slightly larger than the LFSR length. In a second direction, a large scale experiment carried out with the eSTREAM Phase 3 candidate Trivium suggests some immunity of this cipher against algebraic attacks on augmented functions. This experiment becomes feasible as for Trivium with its 288-bit state one can find preimages of 144-bit outputs in polynomial time. Finally, we investigate conditional correlations based on this framework.

Augmented functions of LFSR-based stream ciphers have previously been studied, *e.g.* in [1, 61, 91], where it had been noticed that the augmented function can be weaker than a single output function, with regard to (conditional) correlation attacks as well as to inversion attacks. However, for the first time, we analyze the $\mathcal{AI}$ of sometimes quite large augmented functions. Surprisingly, augmented functions did not receive much attention in this context yet.

4.2 Algebraic Properties of S-boxes

Let $\mathbb{F}$ denote the finite field GF(2), and consider the vectorial Boolean function (or S-box) $S : \mathbb{F}^n \to \mathbb{F}^m$ with $S(x) = z$, where $x := (x_0, \ldots, x_{n-1})$ and $z := (z_0, \ldots, z_{m-1})$. In the case of $m = 1$, the S-box reduces to a Boolean function, and in general, the S-box consists of m Boolean functions $S_i(x)$. These functions give rise to the explicit equations $S_i(x) = z_i$. Here, we assume that z is known and x is unknown.

4.2.1 Implicit Equations

The S-box can hide implicit equations, namely $F(x, z) = 0$ for each $x \in \mathbb{F}^n$ and with $z = S(x)$. The algebraic normal form of such an equation is denoted $F(x, z) = \bigoplus c_{\alpha,\beta} x^\alpha z^\beta = 0$, with coefficients $c_{\alpha,\beta} \in \mathbb{F}$ and multi-indices $\alpha, \beta \in \mathbb{F}^n$ (which can likewise be identified by their integers). In the context of algebraic attacks, it is of interest to focus on implicit equations with special structure, *e.g.* on sparse equations or equations of small degree. Let the degree in x be $d := \max\{|\alpha|, c_{\alpha,\beta} = 1\} \leq n$ with the weight $|\alpha|$ of α, and consider an unrestricted degree for the known z, hence $\max\{|\beta|, c_{\alpha,\beta} = 1\} \leq m$. The maximum number of monomials (or coefficients) in an equation of degree d corresponds to $2^m D$, where $D := \sum_{i=0}^{d} \binom{n}{i}$. In order to determine the existence of an implicit equation of degree d, consider a matrix M in $\mathbb{F}$ of size $2^n \times 2^m D$. Each row corresponds to an input x, and each column corresponds to an evaluated monomial (with some fixed order). If the number of columns in M is larger than the number of rows, then linearly dependent columns (*i.e.* monomials) exist, see [40, 46]. The rank of M determines the number of linearly independent (but potentially not algebraically independent) solutions, and the solutions correspond to the kernel of M^T. Any non-zero implicit equation (which holds for each input x) may then depend on x and z, or on z only. If it depends on x and z, then the equation may degenerate for some values of z. For example, $x_0 z_0 \oplus x_1 z_0 = 0$ degenerates for $z_0 = 0$.

Table 4.1: Theoretical block size m_0 for different parameters n and d.

$_d{}^n$	16	18	20	32	64	128
1	12	14	16	27	58	121
2	9	11	13	23	53	115
3	7	9	10	20	49	110

4.2.2 Conditional Equations

As the output is assumed to be known, one could investigate equations which are conditioned by the output z, hence $F_z(x) = 0$ for each preimage $x \in S^{-1}(z)$ and of degree d in x. The number of preimages is denoted $U_z := |S^{-1}(z)|$, where $U_z = 2^{n-m}$ for balanced S and $m \leq n$. Notice that conditional equations for different outputs z need not be connected in a common implicit equation, and one can find an optimum equation (*i.e.* an equation of minimum degree) for each output z. Degenerated equations are not existing in this situation, and the corresponding matrix M_z has a reduced size of $U_z \times D$. Similar to the case of implicit equations, one obtains:

Proposition 4. *Consider an S-box $S : \mathbb{F}^n \to \mathbb{F}^m$ and let $S(x) = z$. Then, the number of (independent) conditional equations of degree at most d for some z is $R_z = D - \mathrm{rank}(M_z)$. A sufficient criterion for the existence of a non-zero conditional equation is $0 < U_z < D$.*

The condition $R_z > 0$ requires some minimum value of d, which can depend on z. As already proposed in [4], this motivates the following definition of algebraic immunity for S-boxes:

Definition 6. *Consider an S-box $S : \mathbb{F}^n \to \mathbb{F}^m$. Given some fixed output z, let d be the minimum degree of a non-zero conditional equation $F_z(x) = 0$ which holds for all $x \in S^{-1}(z)$. Then the algebraic immunity $\mathcal{AI}$ of S is defined by the minimum of d over all $z \in \mathbb{F}^m$.*

The $\mathcal{AI}$ can be bounded, using the sufficient condition of Prop. 4. Let d_0 be the minimum degree such that $D > 2^{n-m}$. If the S-box is surjective, then there exists at least one z with a non-zero conditional equation of degree at most d_0, hence $\mathcal{AI} \leq d_0$. In addition, the block size m of the output could be considered as a parameter (by investigating truncated S-boxes S_m, corresponding to partial conditioned equations for S). Let $m_0 := \lfloor n - \log_2 D + 1 \rfloor$ for some degree d. Then, the minimum block size m to find non-zero conditional equations of degree at most d is bounded by m_0. See Tab. 4.1 for some numerical values of m_0. A single output z is called *weak*, if non-zero conditional equations of degree d exist for $U_z \gg D$ (or if the output is strongly imbalanced). This roughly corresponds to the condition $d \ll d_0$, or $m \ll m_0$.

4.2.3 Algorithmic Methods

As already mentioned in [33], memory requirements to determine the rank of M are impractical for about $n > 20$. In the case of conditional equations, the matrix M_z can be much smaller, but the bottleneck is to compute an exhaustive list of preimages, which requires a time complexity of 2^n. However, one could use a probabilistic variant of this basic method: Instead of determining the rank of M_z which contains all U_z inputs x, one may solve for a smaller matrix M'_z with $V < U_z$ random inputs. Then, one can determine the *non-existence* of a solution: If no solution exists for M'_z, then no solution exists for M_z either. On the other hand, if one or more solutions exist for M'_z, then they hold true for the subsystem of V inputs, but possibly not for all U_z inputs. Let the probability p be the fraction of preimages that satisfy the equation corresponding to such a solution. With the heuristical argument $(1-p)V < 1$, we expect that $p > 1 - 1/V$. However, this argument holds only for $V > D$, because otherwise, there are always at least $D-V$ solutions (which could be balanced). Consequently, if V is a small multiple of D, the probability can be quite close to one. For this reason, all solutions of the smaller system can be useful in later attacks. As M_z corresponds to a homogeneous system, any linear combination of these solutions is a solution of the subsystem. However, a linear combination may have a different probability with respect to all U_z inputs. Determining only a few random preimages can be very efficient: In a naive approach, time complexity to find a random preimage of an output z is about $2^n/U_z$ (which is 2^m for balanced S), and complexity to find D preimages is about $2^n D/U_z$. This is an improvement compared to the exact method if $U_z \gg D$, *i.e.* equations can be found efficiently for weak outputs. Memory requirements of the probabilistic algorithm are about $C_M = D^2$, and time complexity is about $C_T = D2^m + D^3$.

4.3 Algebraic Attacks based on the Augmented Function

In this section, we focus on algebraic cryptanalysis of S-boxes in the context of stream ciphers. Given a stream cipher, one may construct an S-box as follows:

Definition 7. *Consider a stream cipher with internal state x of n bits, an update function L, and an output function f which outputs one bit of keystream in a single iteration. Then, the augmented function S_m is defined by*

$$\begin{aligned} S_m : \mathbb{F}^n &\to \mathbb{F}^m \\ x &\mapsto (f(x), f(L(x)), \ldots, f(L^{m-1}(x)) \ . \end{aligned} \tag{4.1}$$

The update L can be linear (*e.g.* for filter generators), or nonlinear (*e.g.* for Trivium). The input x correspond to the internal state at some time t, and the output z corresponds to an m-bit block of the known keystream. Notice that m is a very natural parameter here. The goal is to recover the initial state x by algebraic attacks, using (potentially probabilistic) conditional equations $F_z(x) = 0$ of degree d for outputs z of the augmented function S_m. This way, one can set up equations for state variables of different time steps

t. In the case of a linear update function L, each equation can be transformed into an equation of degree d in the initial state variables x. In the case of a nonlinear update function L, the degree of the equations is increasing with time. However, the nonlinear part of the update is sometimes very simple, such that equations for different time steps can be efficiently combined. Finally, the system of equations in the initial state variables x is solved.

If the augmented function has some weak outputs, then conditional equations can be found with the probabilistic algorithm of Sect. 4.2.3, which requires about D preimages of a single m-bit output. One may ask if there is a dedicated way to compute random preimages of m-bit outputs in the context of augmented functions. Any stream cipher as in Def. 7 can be described by a system of equations. Nonlinear systems of equations with roughly the same number of equations as unknowns are in general NP-hard to solve. However, due to the special (simple) structure of some stream ciphers, it may be easy to partially invert the nonlinear system. For example, given a single bit of output of a filter generator, it is easy to find a state which gives out this bit. Efficient computation of random preimages for m-bit outputs is called *sampling*. The sampling resistance is defined as 2^{-m} where m is the maximum value for which we can efficiently produce all preimages of m-bit outputs (without trial and error). Some constructions have very low sampling resistance, see [25, 10].

The parameters of our framework are the degree d of equations, and the block-size m of the output. An optimal tradeoff between these parameters depends on the algebraic properties of the augmented function. The attack is expected to be efficient, if:

1. There are many low-degree conditional equations for S_m.
2. Efficient sampling is possible for this block size m.

This measure is well adapted to the situation of augmented functions, and can be applied to sometimes quite large augmented functions, see Sect. 4.5 and 4.6. This way, we intend to prove some immunity of a stream cipher, or present attacks with reduced complexity.

4.4 Generic Scenarios for Filter Generators

Our framework is investigated in-depth in the context of LFSR-based stream ciphers (and notably for filter generators), which are the main target of conventional and fast algebraic attacks. We describe some elementary conditional equations induced by annihilators. Then, we investigate different methods for sampling, which are necessary to efficiently set up conditional equations. We suggest a basic scenario and estimate data complexity of an attack, the scenario is refined and improved.

4.4.1 Equations Induced by Annihilators

Let us first discuss the existence of conditional equations of degree $d = \mathcal{AI}$, where $\mathcal{AI}$ is the ordinary algebraic immunity of f here. With $m = 1$, the number of conditional

equations for $z = 0$ (resp. $z = 1$) corresponds to the number of annihilators of $f \oplus 1$ (resp. f) of degree d. If one increases m, then all equations originating from annihilators are involved: For example, if there is 1 annihilator of degree d for both f and $f \oplus 1$, then the number of equations is expected to be at least m for any m-bit output z. Notice that equations of fast algebraic attacks are not involved if m is small compared to n.

4.4.2 Sampling

Given an augmented function S_m of a filter generator, the goal of sampling is to efficiently determine preimages x for fixed output $z = S_m(x)$ of m bits. Due to the special structure of the augmented function, there are some efficient methods for sampling:

Filter Inversion. One could choose a fixed value for the k input bits of the filter, such that the observed output bit is correct (using a table of the filter function). This can be done for about n/k successive output bits, until the state is unique. This way, preimages of an output z of n/k bits can be found in polynomial time, and by partial search, preimages of larger outputs can be computed. Time complexity to find a preimage of $m > n/k$ bits is about $2^{m-n/k}$, *i.e.* the method is efficient if there are only few inputs k.

Linear Sampling. In each time step, a number of l linear conditions are imposed on the input variables of f, such that the filter becomes linear. The linearized filter gives one additional linear equation for each keystream bit. Notice that all variables can be expressed by a linear function of the n variables of the initial state. Consequently, for an output z of m bits, one obtains $(l+1)m$ (inhomogeneous) linear equations for n unknowns, *i.e.* we expect that preimages can be found in polynomial time if $m \leq n/(l+1)$. To find many different preimages, one should have several independent conditions (which can be combined in a different way for each clock cycle).

In practice, sampling should be implemented carefully in order to avoid contradictions (*e.g.* with appropriate conditions depending on the keystream), see [25].

4.4.3 Basic Scenario

We describe a basic scenario for algebraic attacks on filter generators based on the augmented function: With C_D bits of keystream, one has $C'_D = C_D - m + 1$ (overlapping) windows of m bits. Assume that there are $R := \sum_z R_z$ equations of degree d for m-bit outputs z. For each window, we have about $r := R/2^m$ equations, which gives a total of $N = rC'_D$ equations.[1] Each equation has at most D monomials in the initial state variables, so we need about the same number of equations to solve the system by linearization. Consequently, data complexity is $C_D = D/r + m - 1$ bits. The initial state can then be recovered in $C_T = D^3$. This should be compared with the complexity of conventional

[1] From a heuristical point of view, the parameter r is only meaningful if the conditional equations are approximately uniformly distributed over all outputs z.

algebraic attacks $C_D = 2E/R_A$ and $C_T = E^3$, where $e := \mathcal{AI}$, $E := \sum_{i=0}^{e} \binom{n}{i}$, and R_A the number of annihilators of degree e. Notice that the augmented function may give low-degree equations, which are not visible for single-bit outputs; this increases information density and may reduce data complexity. Our approach has reduced time complexity if $d < e$, provided that sampling (and solving the matrix) is efficient.

4.4.4 Refined Basic Scenario

The basic scenario for filter generators should be refined in two aspects, concerning the existence of dependent and probabilistic equations: First, with overlapping windows of m bits, it may well happen that the same equation is counted several times, namely if the equation already exists for a substring of $m' < m$ bits (*e.g.* in the case of equations produced by annihilators). In addition, equations may be linearly dependent by chance. If this is not considered in the computation of R, one may have to enlarge data complexity a little bit. Second, one can expect to obtain probabilistic solutions. However, depending on the number of computed preimages, the probability p may be large and the corresponding equations can still be used in our framework, as they increase R and reduce data complexity, but potentially with some more cost in time. As we need about D (correct and linearly independent) equations to recover the initial state, the probability p should be at least $1 - 1/D$ (together with our estimation for p, this justifies that the number of preimages should be at least D). In the case of a contradiction, one could complement a few equations in a partial search and solve again, until the keystream can be verified. Depending on the actual situation, one may find an optimal tradeoff in the number of computed preimages. Notice that our probabilistic attack deduced from an algebraic attack with equations of degree 1 is a powerful variant of a conditional correlation attack, see [91]. A probabilistic attack with nonlinear equations is a kind of higher order correlation attack, see [39].

4.4.5 Substitution of Equations

It is possible to further reduce data complexity in some cases. Consider the scenario where one has $N = rC'_D$ linear equations. On the other hand, given an annihilator of degree $e := \mathcal{AI}$, one can set up a system of degree e as in conventional algebraic attacks. The N linear equations can be substituted into this system in order to eliminate N variables. This results in a system of $D' := \sum_{i=0}^{e} \binom{n-N}{i}$ monomials, requiring a data complexity of $C_D = D'$ and time complexity $C_T = D'^3$. Notice that data can be reused in this case, which gives the implicit equation in C_D. Obviously, a necessary condition for the success of this method is $rE > 1$. A similar improvement of data complexity is possible for nonlinear equations of degree d. One can multiply the equations by all monomials of degree $e - d$ in order to obtain additional equations of degree e, along the lines of XL [44] and Gröbner bases algorithms.

Table 4.2: Different setups for our experiments with filter generators.

Setup	n	k	feedback taps	filter taps
1	18	5	$[2, 3, 5, 15, 17, 18]$	$[1, 2, 7, 11, 18]$
2	18	5	$[1, 2, 5, 7, 9, 14, 15, 16, 17, 18]$	$[1, 3, 7, 17, 18]$
3	18	5	$[3, 5, 7, 15, 17, 18]$	$[1, 5, 8, 16, 18]$
4	18	5	$[4, 5, 6, 10, 13, 15, 16, 18]$	$[1, 6, 7, 15, 18]$
5	18	5	$[2, 3, 5, 7, 11, 15, 17, 18]$	$[1, 3, 6, 10, 18]$
6	20	5	$[7, 10, 13, 17, 18, 20]$	$[1, 3, 9, 16, 20]$
7	20	5	$[1, 2, 4, 7, 8, 10, 11, 12, 13, 15, 19, 20]$	$[1, 5, 15, 18, 20]$
8	20	5	$[2, 3, 4, 5, 6, 7, 8, 11, 13, 14, 19, 20]$	$[1, 4, 9, 16, 20]$
9	20	5	$[1, 2, 3, 4, 5, 8, 10, 11, 12, 13, 15, 17, 19, 20]$	$[1, 2, 15, 17, 20]$
10	20	5	$[1, 2, 6, 7, 9, 11, 15, 20]$	$[1, 5, 13, 18, 20]$
11	40	5	$[3, 8, 9, 10, 11, 13, 14, 15, 18, 19,$ $23, 24, 25, 26, 27, 30, 33, 34, 36, 40]$	$[1, 3, 10, 27, 40]$

4.5 First Application: Some Specific Filter Generators

Many conventional algebraic attacks on filter generators require about $\binom{n}{e}$ output bits where e equals the algebraic immunity of the filter function. On the other hand, in [57], algebraic attacks based on Gröbner bases are presented, which in a few cases require only $n + \varepsilon$ data. It is an open issue to understand such a behavior from the Boolean function and the tapping sequence. We present attacks on the corresponding augmented functions, requiring very low data complexity. This means, we can identify the source of the above behavior, and in addition, we can use our method also for other functions.

4.5.1 Experimental Setup for Filter Generators

In Tab. 4.2, we collect the setups of our experiments with filter generators, where n is the size of the LFSR, and k the number of inputs to the filter function. The feedback taps are chosen such that the LFSR has maximum period (*i.e.*, the corresponding polynomial is primitive), and filter taps are chosen according to a full positive difference set (*i.e.*, all the positive pairwise differences are distinct). Tap positions are counted from the left (starting by 1), and the LFSR is shifted to the right.

4.5.2 Existence of Equations

In this subsection, we give extensive experimental results for different filter generators. Our setup is chosen as follows: The filter functions are instances of the CanFil family (see [57]) or the Majority functions. These instances all have five inputs and algebraic immunity 2 or 3. Feedback taps correspond to a random primitive feedback polynomial, and filter taps are chosen randomly in the class of full positive difference sets, see Tab. 4.2 in Appendix 4.5.1 for an enumerated specification of our setups. Given a specified filter

Table 4.3: Counting the number of linear equations R for the augmented function of different filter generators, with $n = 20$ bit input and m bit output.

Filter	m	R for setups $6-10$				
CanFil1	14	0	0	0	0	0
	15	3139	4211	3071	4601	3844
CanFil2	14	0	0	0	0	0
	15	2136	2901	2717	2702	2456
CanFil5	6	0	0	0	2	0
	7	0	0	0	8	0
	8	0	0	0	24	0
	9	0	0	0	64	0
	10	6	0	0	163	0
	11	113	0	2	476	0
	12	960	16	215	1678	29
Majority5	9	0	0	0	2	0
	10	1	10	1	18	1
	11	22	437	40	148	56

generator and parameters d and m, we compute the number R_z of independent conditional equations $F_z(x) = 0$ of degree d for each output $z \in \mathbb{F}^m$. The overall number of equations $R := \sum_z R_z$ for $n = 20$ is recorded in Tab. 4.3. Thereby, preimages are computed by exhaustive search in order to exclude probabilistic solutions.

In the case of CanFil1 and CanFil2, linear equations exist only for $m \geq m_0 - 1$, independent of the setup. On the other hand, for CanFil5 and Majority5, there exist many setups where a large number of linear equations already exists for $m \approx n/2$, see Ex. 9. We conclude that the number of equations weakly depends on the setup, but is mainly a property of the filter function. The situation is very similar for other values of n, see Tab. 4.4 for $n = 18$. This suggests that our results can be scaled to larger values of n. Let us also investigate existence of equations of higher degree: CanFil1 and CanFil2 have $\mathcal{AI} = 2$ and there is 1 annihilator for both f and $f \oplus 1$, which means that at least m quadratic equations can be expected for an m-bit output. For each setup and $m < m_0 - 1$, we observed only few additional equations, whereas the number of additional equations is exploding for larger values of m. This was observed for many different setups and different values of n.

Example 9. Consider CanFil5 with $n = 20$ and setup 9. For the output $z = 000000$ of $m = 6$ bits, there are exactly 2^{14} preimages, hence the matrix M_z has 2^{14} rows and $D = 21$ columns for $d = 1$. Evaluation of M_z yields a rank of 20, *i.e.* a nontrivial solution exists. The explicit solution is $F_z(x) = x_1 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_9 \oplus x_{10} \oplus x_{11} \oplus x_{12} \oplus x_{13} \oplus x_{14} \oplus x_{16} = 0$.□

Table 4.4: Counting the number of linear equations R for the augmented function of different filter generators, with $n = 18$ bit input and m bit output.

Filter	m	R for setups 1-5				
CanFil1	12	0	0	0	0	0
	13	625	288	908	335	493
CanFil2	12	0	0	0	0	0
	13	144	346	514	207	418
CanFil3	12	0	0	4	0	0
	13	1272	1759	2173	2097	983
CanFil4	7	0	0	0	0	0
	8	19	4	0	0	0
	9	102	17	1	0	12
	10	533	69	9	20	167
CanFil5	6	1	0	0	0	0
	7	4	0	0	0	0
	8	15	0	0	0	1
	9	55	1	0	0	39
	10	411	61	3	0	360
	11	2142	1017	166	10	1958
CanFil6	8	0	0	0	0	0
	9	0	10	64	0	0
	10	0	97	256	0	0
	11	0	517	1024	0	0
	12	0	2841	3533	1068	0
	13	152	19531	17626	12627	9828
CanFil7	11	0	2	0	0	6
	12	68	191	36	26	178
Majority5	8	1	0	0	0	0
	9	8	3	42	27	14
	10	97	94	401	282	158

4.5.3 Probabilistic Equations

In the previous subsection, the size n of the state was small enough to compute a complete set of preimages for some m-bit output z. However, in any practical situation where n is larger, the number of available preimages is only a small multiple of D, which may introduce probabilistic solutions. Here is an example with $n = 20$, where the probability can be computed exactly:

Example 10. Consider again CanFil5 with $n = 20$ and setup 9. For the output $z = 000000$ of $m = 6$ bits, there are 2^{14} preimages and one exact conditional equation of degree $d = 1$. We picked 80 random preimages and determined all (correct or probabilistic) linear conditional equations. This experiment was repeated 20 times with different preimages. In each run, we obtained between 2 and 4 independent equations with probabilities $p = 0.98, \ldots, 1$. For example, the (probabilistic) conditional equation $F_z(x) = x_1 \oplus x_2 \oplus x_3 \oplus x_6 \oplus x_9 \oplus x_{15} \oplus x_{16} \oplus x_{17} = 0$ holds with probability $p = 1 - 2^{-9}$. □

In the above example, there are only few probabilistic solutions and they have impressively large probability, which makes the equations very useful in an attack. Notice that experimental probability is in good agreement with our estimation $p > 1 - 1/80 = 0.9875$. The situation is very similar for other parameters. With the above setup and $m = 10$, not only $z = 000\ldots0$ but a majority of outputs z give rise to linear probabilistic equations. In the case of CanFil1 and CanFil2, we did not observe linear equations of large probability for $m < m_0 - 1$. It is interesting to investigate the situation for larger values of n:

Example 11. Consider CanFil5 with $n = 40$ and setup 11. For the output $z = 000\ldots0$ of $m = 20$ bits, we determine 200 random preimages. With $d = 1$, evaluation of M_z yields a rank of 30, *i.e.* 11 (independent) solutions exist. With 2000 random preimages, we observed a rank of 33, *i.e.* only 3 solutions of the first system were detected to be merely probabilistic. An example of an equation is $F_z(x) = x_0 \oplus x_7 \oplus x_9 \oplus x_{13} \oplus x_{14} \oplus x_{17} \oplus x_{18} \oplus x_{25} \oplus x_{30} \oplus x_{33} = 0$. □

The remaining 8 solutions of the above example may be exact, or probabilistic with very high probability. By sampling, one could find (probabilistic) conditional equations for much larger values of n. For example, with CanFil5, $n = 80$, $m = 40$ and filter inversion, time complexity to find a linear equation for a weak output is around 2^{32}.

4.5.4 Discussion of Attacks

Our experimental results reveal that some filter functions are very vulnerable to algebraic attacks based on the corresponding augmented function. For CanFil5 with $n = 20$ and setup 9, we observed $R = 163$ exact equations using the parameters $m = 10$ and $d = 1$, which gives a ratio of $r = 0.16$. Including probabilistic equations, this ratio may be even larger. Here, preimages of any z can be found efficiently by sampling: using filter inversion, a single preimage can be found in $2^{m-n/k} = 2^6$ steps, and a single equation in around 2^{13} steps. Provided that equations are independent and the probability is large,

data complexity is about $C_D = (n+1)/r + m - 1 = 140$. The linear equations could also be substituted into the system of degree $\mathcal{AI} = 2$, which results in a data complexity of about $C_D = 66$. Notice that conventional algebraic attacks would require $C_D = E = 211$ bits (and time complexity E^3). As we expect that our observation can be scaled, (*i.e.* that r remains constant for larger values of n and $m = n/2$), data complexity is a linear function in n. Considering time complexity for variable n, the matrix M and the final system of equations can be solved in polynomial time, whereas sampling is subexponential (and polynomial in some cases, where linear sampling is possible).

In [57], CanFil5 has been attacked experimentally with $n+\varepsilon$ data, where $n = 40, \ldots, 70$ and $\varepsilon < 10$. Our analysis gives a conclusive justification for their observation. Other functions such as Majority5 could be attacked in a similar way, whereas CanFil1 and CanFil2 are shown to be much more resistant against this general attack: No linear equations have been found for $m < m_0 - 1$, and only few quadratic equations.

4.6 Second Application: Trivium

Trivium [32] is a stream cipher with a state of 288 bits, a nonlinear update and a linear output. It has a simple algebraic structure, which makes it an interesting candidate for our framework. We consider the S-box $S_m(x) = z$, where S is the augmented function of Trivium, x the state of $n = 288$ bits, and z the output of m bits. We will first analyze the sampling of S_m, which is very similar to linear sampling of filter generators.

4.6.1 Sampling

The state consists of the 3 registers $R_1 = (x_0, \ldots, x_{92})$, $R_2 = (x_{93}, \ldots, x_{176})$ and $R_3 = (x_{177}, \ldots, x_{287})$. In each clock cycle, a linear combination of 6 bits of the state (2 bits of each register) is output. Then, the registers are shifted to the right by one position, with a nonlinear feedback to the first position of each register. In the first 66 clocks, each keystream bit is a linear function of the input, whereas the subsequent keystream bit involves a nonlinear expression. Consequently, given any output of $m = 66$ bits, one can efficiently determine some preimages by solving a linear system. It is possible to find preimages of even larger output size. Observe that the nonlinear function is quadratic, where the two factors of the product have subsequent indices. Consequently, one could fix some alternating bits of the state, which results in additional linear equations for the remaining variables. Let `c`, `l`, `q` denote constant, linear, and quadratic dependence on the initial state. Let all the even bits of the initial state be `c`, see Tab. 4.5. After update 83, bits 82 and 83 (counting from 1) of R_2 are both `l`. Variable t_2 takes bits 82 and 83 of R_2 to compute the nonlinear term. So after update 84, $t_2 = x_{177}$ is `q` (where nonlinear terms in t_1 and t_3 appear somewhat later). After 65 more updates, x_{242} is quadratic, where x_{242} is filtered out from R_3 in the next update (after 84 updates, other bits are also `q` and are filtered out from registers R_1 and R_2, but on a later point in time). Consequently, keystream bit number $66 + 84 = 150$ (counting from 1) is `q`, and the first 149 keystream bits are linear in the remaining variables. The number of remaining variables in the state

Table 4.5: Evolution of states with partially fixed input.

Initial state	After 1 update	After 84 updates
$R_1 = \texttt{lclcl}\ldots$	$R_1 = \texttt{llclcl}\ldots$	$R_1 = \texttt{lllll}\ldots$
$R_2 = \texttt{clclc}\ldots$	$R_2 = \texttt{lclclc}\ldots$	$R_2 = \texttt{lllll}\ldots$
$R_3 = \texttt{clclc}\ldots$	$R_3 = \texttt{lclclc}\ldots$	$R_3 = \texttt{qllll}\ldots$

(the degree of freedom) is 144. Consequently, for an output of size $m = 144$ bits, we can expect to find one solution for the remaining variables; this was verified experimentally. The solution (combined with the fixed bits) yields a preimage of z. Notice that we do not exclude any preimages this way. In addition, m can be somewhat larger with partial search for the additional bits.

Example 12. Consider the special output $z = \texttt{000}\ldots\texttt{0}$ of $m = 160$ bits. By sampling and partial exhaustive search, we find the following nontrivial preimage:

$$x = \begin{array}{l} \texttt{10001011110001011100110001010100110100001001001}\texttt{0} \\ \texttt{00010010010011001111101101110110000100110010100}\texttt{0} \\ \texttt{11000000010101100111000011111101100110000110101}\texttt{0} \\ \texttt{01110000010101001100110111101010101111111010000}\texttt{1} \\ \texttt{00000100000110100010000111100110101010000101011}\texttt{1} \\ \texttt{10100000111010010101001100010011100101001010110}\texttt{1} \end{array}$$

□

4.6.2 Potential Attacks

The nonlinear update of Trivium results in equations $S_m(x) = z$ of increasing degree for increasing values of m. However, for any output z, there are at least 66 linear equations in the input variables. It is an important and security related question, if there are additional linear equations for some fixed output z. A linear equation is determined by $D = 289$ coefficients, thus we have to compute somewhat more than 289 preimages for this output. By sampling, this can be done in polynomial time. Here is an experiment:

Example 13. Consider a prescribed output z of 144 bits, and compute 400 preimages x such that $S_m(x) = z$ (where the preimages are computed by a uniform random choice of 144 fixed bits of x). Given these preimages, set up and solve the matrix M of linear monomials in x. For 30 uniform random choices of z, we always observed 66 linearly independent solutions. □

Consequently, Trivium seems to be immune against additional linear equations, that might help in an attack. Because of the lack of probabilistic solutions, Trivium is also supposed to be immune against equations of large probability (compare with CanFil1 and CanFil2). As pointed out in [75], there are some states resulting in a weak output: If R_1, R_2 and R_3 are initialized by some period-3 states, then the whole state (and hence the output)

repeats itself every 3 iterations. Each of these states results in $z = \texttt{000}\ldots\texttt{0}$. Here is an extended experiment (with partial exhaustive search) for this special output:

Example 14. Consider the output $z = \texttt{000}\ldots\texttt{0}$ of 150 bits, and compute 400 random preimages x such that $S_m(x) = z$. By solving the matrix M of linear monomials in x, we still observed 66 linearly independent solutions. □

4.7 Conditional Correlations

Exact equations (or equations with very large probability) are used in algebraic attacks on stream ciphers, and probabilistic linear equations are used in correlation attacks and linear attacks (*e.g.* for the Boolean function in combination generators). In spite of this close relationship, these two approaches have never been analyzed in a unified framework. We briefly sketch how to use the previous methods to find conditional correlations.

4.7.1 Nonlinearity

Consider a Boolean function $f : \mathbb{F}^n \to \mathbb{F}$, where the output $f(x)$ is assumed to be known. Let $u \in \mathbb{F}^n$ and define the dot product by $u{\cdot}x := u_0x_0 \oplus \ldots \oplus u_{n-1}x_{n-1}$. The problem here is to find the vector u such that the probabilistic linear equation $f(x) = u \cdot x$ has maximum probability for uniformly random x. This problem can be restated with the notation of (conventional) *nonlinearity*, which is the Hamming distance of f to the set of affine functions. More precisely, the *Walsh-Transform* of f is defined by $\hat{f}(u) := \sum_x (-1)^{f(x)\oplus u\cdot x}$. The linear correlation coefficient is $C := \max_u \hat{f}(u)$ and conventional nonlinearity becomes $\mathcal{NL} := \frac{1}{2}(2^n - C)$. This means $p := \Pr_x(f(x) = u{\cdot}x) = 1 - \mathcal{NL}/2^n$ for some optimal u. For an S-box $S : \mathbb{F}^n \to \mathbb{F}^m$, one can investigate the maximum probability of $\Pr_x(u{\cdot}S(x) = v{\cdot}x)$ with $u \in \mathbb{F}^m$ and $v \in \mathbb{F}^n$. Zhang and Chan [125] observed that if $S(x)$ is known, then we can compose $S(x)$ with any (and not necessarily linear) Boolean function $g : \mathbb{F}^m \to \mathbb{F}$ and consider the maximum probability of $\Pr_x(g(S(x)) = v \cdot x)$. Since we are choosing from a larger set of equations now, we can find linear approximations with larger bias, but finding the relation is more difficult. This concept was improved again in [36] by considering all implicit equations $F(x, z) = 0$ with $z = S(x)$ which are non-degenerate for all z and linear in x. They introduced the notation of generalized nonlinearity, which can be computed in about 2^{2n} steps (assuming $m < n$). We use a similar notation for *conditional equations* with fixed output z, see also [92, 91].

Definition 8. *Consider an S-box $S : \mathbb{F}^n \to \mathbb{F}^m$ and the set $\mathcal{F}$ of linear Boolean functions $F : \mathbb{F}^n \to \mathbb{F}$. Define the conditional probability for a fixed output z and a function $F \in \mathcal{F}$ by $p_z := \Pr_{S^{-1}(z)}(F(x) = 0)$, where $S^{-1}(z)$ is the set of preimages of size U_z. Let $\hat{F} := \sum_{S^{-1}(z)} (-1)^{F(x)}$ and $C_z := \max_{F\in\mathcal{F}} |\hat{F}|$. The conditional nonlinearity of S with respect to $\mathcal{F}$ (and conditioned by z) is defined by $\mathcal{NL}_z := \frac{1}{2}(U_z - C_z)$, and the maximum conditional probability becomes $p_z = 1 - \mathcal{NL}_z/U_z$.*

Note that $\mathcal{NL}_z$ may be computed for each output z separately, involving the computation of preimages. According to the definition of $\mathcal{AI}$ for an S-box, the nonlinearity $\mathcal{NL}$ could

Table 4.6: List of maximal conditional probabilities for CanFil1 with $n = 20$ and different setups.

m	Setup 6	Setup 7	Setup 8	Setup 9	Setup 10
2	0.625	0.667	0.500	0.563	0.531
3	0.625	0.750	0.625	0.625	0.531
4	0.625	0.750	0.625	0.656	0.625
5	0.625	0.800	0.625	0.664	0.625
6	0.628	0.833	0.657	0.697	0.656
7	0.661	0.835	0.659	0.708	0.658
8	0.677	0.861	0.673	0.734	0.683
9	0.756	0.917	0.733	0.749	0.712
10	0.756	0.935	0.770	0.778	0.762

be (re-)defined by the minimum of $\mathcal{NL}_z$ for all z. For linear functions, the number of monomials is $D = n$ and the size of $\mathcal{F}$ is 2^n. Let us discuss some algorithmic methods to determine $\mathcal{NL}_z$. In a direct approach, each of the 2^n linear functions in $\mathcal{F}$ is evaluated for the given set of preimages in order to find the maximum p_z. For a balanced S it is $U_z = 2^{n-m}$, hence the complexity to compute $\mathcal{NL}_z$ is about 2^{2n-m}, and the complexity to compute the (alternative) $\mathcal{NL}$ is 2^{2n}. We can also use the probabilistic algorithm from Sect. 4.2.3 with only $V \ll U_z$ random preimages. For $V < D$, there are at least $D - V$ independent equations. On the other hand, V should be as large as possible in order to increase the probability of an equation (by increasing the number of conditions). In practice, we will choose V slightly below D. If one (or more) solutions F can be found, one has to estimate the conditional correlation p_z with some additional preimages of z. All steps can be repeated T times to identify the solution F with maximum empirical p_z. The bottleneck of this method is to find preimages of z.

4.7.2 Experimental Results

In Sect. 4.5, it turned out that some Boolean functions of filter generators have many (exact) linear conditional equations already for small values of m (*e.g.* CanFil5), whereas other functions are more resistant (*e.g.* CanFil1). Here, we investigate the existence of probabilistic equations for filter generators for even smaller values of m.

Example 15. Consider a filter generator with $n = 20$, filter CanFil1 and setup 6. Choose parameters $V = 15$, $T = 20$ and compute the (empirical) maximum of the conditional probabilities, see Tab. 4.6 (and Tab. 4.7 for CanFil5). For example, for the output $z = 000\ldots0$ of $m = 10$ bits, the equation $F_z(x) = x_0 \oplus x_2 \oplus x_3 \oplus x_7 \oplus x_{10} \oplus x_{12} \oplus x_{13} = 0$ holds with probability $p_z = 0.739$. □

Consequently, one can find probabilistic equations for small m, even for some functions which turned out to be strong with respect to exact equations. The correlations weakly

Table 4.7: List of maximal conditional probabilities for CanFil5 with $n = 20$ and different setups.

m	Setup 6	Setup 7	Setup 8	Setup 9	Setup 10
2	0.929	0.928	0.833	0.942	0.929
3	0.978	0.800	0.929	0.985	0.935
4	0.978	0.800	0.978	0.997	0.935
5	0.995	0.933	0.979	0.996	0.949
6	0.995	0.933	0.984	0.998	0.982
7	0.996	0.970	0.985	1.000	0.985
8	0.997	0.977	0.995	1.000	0.987
9	0.998	0.994	0.999	1.000	0.988
10	1.000	1.000	1.000	1.000	0.994

depend on the setup, and strongly depend on the filter. We observe that the probabilistic algorithm can find the maximum conditional probability p_z (resp. the nonlinearity $\mathcal{NL}$) in most of the cases.

4.8 Summary

Intrinsic properties of augmented functions of stream ciphers have been investigated with regard to algebraic attacks and linear attacks. Certain properties of the augmented function enable efficient algebraic attacks with lower data complexity than established algebraic attacks. In order to assess resistance of augmented functions against such improved algebraic attacks, a prespecified number of preimages of outputs of various size of these functions have to be found. For a random function, the difficulty of finding preimages increases exponentially with the output size. However, due to a special structure of the augmented function of a stream cipher, this can be much simpler than in the random case. For any such stream cipher, our results show the necessity of checking the augmented function for algebraic relations of low degree for output sizes for which finding preimages is feasible. In this chapter, this has been successfully carried out for various filter generators as well as for the eSTREAM candidate Trivium.

Chapter 5

Attacks on the Alternating Step Generator

In the previous chapter it was noticed that sampling may be useful along with other attacks in a unified framework. The results of this chapter represent a positive attempt to exploit such a connection for a concrete stream cipher: we present some reduced complexity attacks on the Alternating Step Generator (ASG). The attacks mostly benefit from the low sampling resistance of the ASG, and of an abnormal behavior related to the distribution of the initial states of the stop/go LFSR's which produce a given segment of the output sequence.

5.1 Introduction

The Alternating Step Generator (ASG), a well-known stream cipher proposed in [71], consists of two stop/go clocked binary LFSR's, LFSR_x and LFSR_y, and a regularly clocked binary LFSR, LFSR_c of which the clock-control sequence is derived. The original description of ASG [71] is as follows. At each time, the clock-control bit determines which of the two stop/go LFSR's is clocked, and the output sequence is obtained as bitwise sum of the two stop/go clocked LFSR sequences. It is known [79, 65, 78] that instead of working with the original definition of ASG we can consider a slightly different description for which the output is taken from the stop/go LFSR which has been clocked. More precisely, at each step first LFSR_c is clocked; then if the output bit of LFSR_c is one, LFSR_x is clocked and its output bit is considered as the output bit of the generator, otherwise LFSR_y is clocked and the output bit of the generator is taken from this LFSR. Since in a cryptanalysis point of view these two generators are equivalent, we use the later one all over this chapter and for simplicity we still call it ASG.

Several attacks have been proposed on ASG in the literature. Most of these attacks are applied in a divide-and-conquer based procedure targeting one or two of the involved LFSR's. We will focus on a divide-and-conquer attack which targets one of the two stop/go LFSR's.

A correlation attack on individual LFSR_x or LFSR_y which is based on a specific edit probability has been introduced in [66]. The amount of required keystream is linear in terms of the length of the targeted LFSR and the correct initial state of the targeted LFSR

is found through an exhaustive search over all possible initial states. In [79] some reduced complexity attacks on ASG and SG (Shrinking Generator, see [38]) were presented and the effectiveness of the attacks was verified numerically for SG while only few general ideas were proposed for ASG without any numerical or theoretical analysis. These methods avoid exhaustive search over all initial states, however, the amount of needed keystream is exponential in terms of the length of the targeted LFSR. One of our contributions of this chapter is to give a closed form for the reduced complexity attacks on ASG.

Our major objective of this chapter is to investigate a general method which does not perform an exhaustive search over all possible initial states of the targeted LFSR. We will take advantage of the low *sampling resistance* of ASG. For ASG, sampling is easy if the output length m is chosen to be about the total length of the two stop/go LFSR's. Another weakness of ASG which enables us to mount our attack is that different initial states of any of the two stop/go LFSR's have far different probabilities to be accepted as a candidate which can produce a given segment of length m of the output sequence. Systematic computer simulations confirm this striking behavior. The highly non-uniform distribution of different initial states of any of the stop/go LFSR's is valid for any segment of length about m, and the effect is more abnormal for some special outputs which we refer to as weak outputs. Thanks to the low sampling resistance of ASG we first try to find a subset of the most probable initial states which contains the correct one, then using the probabilistic edit distance [66] we distinguish the correct initial state. Our general approach can be faster than exhaustive search even if the amount of keystream is linear in terms of the length of the targeted LFSR, improving the results in [66]. With regard to reduced complexity attacks, our approach does assume less restricted output segments than in [79], a fact that has been confirmed by large-scale experiments. This enables attacks with significantly lower data complexity even for large instances of ASG (whereas asymptotical complexity is shown to be comparable over known methods).

5.2 Previous Attacks on ASG

Several attacks have been proposed on the ASG in the literature. This section will provide an overview of the different attacks. We will denote the length of registers $\mathsf{LFSR_c}$, $\mathsf{LFSR_x}$ and $\mathsf{LFSR_y}$ by n_c, n_x and n_y, respectively. If we only use parameter n, we apply the simplification $n := n_c = n_x = n_y$.

5.2.1 Divide-and-Conquer Linear Consistency Attack

It is shown in [71] that the initial state of $\mathsf{LFSR_c}$ can be recovered by targeting its initial state in a divide-and-conquer based attack based on the fact that the output sequence of the ASG can be split into the regularly clocked $\mathsf{LFSR_x}$ and $\mathsf{LFSR_y}$ sequences, which are then easily tested for low linear complexity. Hence the complexity of this attack is $\mathcal{O}(\min^2(n_x, n_y)2^{n_c})$ assuming that only the feedback polynomial of $\mathsf{LFSR_c}$ is available. Under the assumption that the feedback polynomial of all LFSR's are available, which is the basic assumption of all other known attacks (including ours in this chapter), the

complexity of this attack would be $\mathcal{O}(\min(n_x, n_y)2^{n_c})$ instead, since a parity check test can be used in place of linear complexity test. In this case the attack is a linear consistency attack [124]. We will use the idea of this attack to sample ASG in Sect. 5.4.1.

5.2.2 Edit Distance Correlation Attack

A correlation attack on $\mathsf{LFSR_x}$ and $\mathsf{LFSR_y}$ combined, which is based on a specific edit distance, was proposed in [65]. If the initial states of $\mathsf{LFSR_x}$ and $\mathsf{LFSR_y}$ are guessed correctly, the edit distance is equal to zero. If the guess is incorrect, the probability of obtaining the zero edit distance was experimentally shown to exponentially decrease in the length of the output string. Later, a theoretical analysis of this attack was developed in [78, 63]. The minimum length of the output string to be successful for an attack is about four times total lengths of $\mathsf{LFSR_x}$ and $\mathsf{LFSR_y}$. As the complexity of computing the edit distance is quadratic in the length of the output string, the complexity of this attack is $\mathcal{O}((n_x + n_y)^2 2^{n_x+n_y})$. In addition, it was shown that the initial state of $\mathsf{LFSR_c}$ can then be reconstructed with complexity $\mathcal{O}(2^{0.27n_c})$.

5.2.3 Edit Probability Correlation Attack

A correlation attack on individual $\mathsf{LFSR_x}$ or $\mathsf{LFSR_y}$ which is based on a specific edit probability was developed in [66]. For a similar approach, see [79]. The edit probability is defined for two binary strings: an input string, produced by the regularly clocked targeted LFSR from an assumed initial state, and a given segment of the ASG output sequence. The edit probability is defined as the probability that the given output string is produced from an assumed input string by the ASG in a probabilistic model, where the LFSR sequences are assumed to be independent and purely random. It turns out that the edit probability tends to be larger when the guess about the LFSR initial state is correct. More precisely, by experimental analysis of the underlying statistical hypothesis testing problem, it was shown that the minimum length of the output string to be successful for an attack is about forty lengths of the targeted LFSR. As the complexity of computing the edit probability is quadratic in the length of the output string, the complexity of reconstructing both LFSR initial states is $\mathcal{O}(\max^2(n_x, n_y)2^{\max(n_x,n_y)})$. This yields a considerable improvement over the edit distance correlation attack if n_x and n_y are approximately equal and relatively large, as is typically suggested (for example, see [98]).

Remark 3. Note that "edit distance correlation attack" means that the initial states of $\mathsf{LFSR_x}$ and $\mathsf{LFSR_y}$ can be recovered regardless of the unknown initial state of $\mathsf{LFSR_c}$, whereas "edit probability correlation attack" means that the initial state of $\mathsf{LFSR_x}$ ($\mathsf{LFSR_y}$) can be recovered regardless of unknown initial states of $\mathsf{LFSR_y}$ ($\mathsf{LFSR_x}$) and $\mathsf{LFSR_c}$. However, the targeted LFSR initial states should be tested exhaustively. The main motivation for this chapter is to investigate if the initial states of $\mathsf{LFSR_x}$ ($\mathsf{LFSR_y}$) can be reconstructed faster than exhaustive search regardless of unknown initial states of $\mathsf{LFSR_y}$ ($\mathsf{LFSR_x}$) and $\mathsf{LFSR_c}$.

5.2.4 Reduced Complexity Attacks

A first step to faster reconstruction of LFSR's initial states was suggested in [79], in which some reduced complexity attacks on ASG and SG are presented. In the next section, we will give a general expression in the parameter n_x, the length of target register $\mathsf{LFSR_x}$. A second movement to faster reconstruction of LFSR initial states was suggested in [68], using an approach based on computing the posterior probabilities of individual bits of the regularly clocked $\mathsf{LFSR_x}$ and $\mathsf{LFSR_y}$ sequences, when conditioned on a given segment of the output sequence. It is shown that these probabilities can be efficiently computed and the deviation of posterior probabilities from one half are theoretically analyzed. As these probabilities represent soft-valued estimates of the corresponding bits of the considered LFSR sequences when regularly clocked, it is argued that the initial state reconstruction is thus in principle reduced to fast correlation attacks on regularly clocked LFSR's such as the ones based on iterative probabilistic decoding algorithms. Although this valuable work shows some vulnerability of the ASG towards fast correlation attacks, the practical use of these probabilities has not yet been deeply investigated. Nonetheless, these posterior probabilities can certainly be used to mount a distinguisher on ASG. This can be compared with [62], a similar work on SG for which a distinguisher was later developed in [69].

5.3 Johansson's Reduced Complexity Attacks

In [79] some reduced complexity attacks on the ASG and SG were presented, and the effectiveness of the attacks was verified numerically for the SG (while only few general ideas were proposed for the ASG without any numerical or theoretical analysis). We give a closed form for the reduced complexity attack on ASG, using the approximation $\binom{n}{w} \approx 2^{nh(w/n)}$ where $h(p)$ is the binary entropy function defined as

$$h(p) := -p\log_2(p) - (1-p)\log_2(1-p)\ . \tag{5.1}$$

In the first scenario, the attacker waits for a segment of m consecutive zeros (or ones) in the output sequence and assumes that exactly $m/2$ of them are from $\mathsf{LFSR_x}$. This is true with probability $\beta = \binom{m}{m/2}2^{-m}$. The remaining $n - m/2$ bits of $\mathsf{LFSR_x}$ are then found by exhaustive search. Time and data complexities of this attack are $C_T = n^2 2^{n-m/2}\beta^{-1} = n^2 2^{n+m/2}\binom{m}{m/2}^{-1}$ and $C_D = 2^{m-1}\beta^{-1} = 2^{2m-1}\binom{m}{m/2}^{-1}$ (using overlapping blocks of keystream). Ignoring the polynomial and constant terms and equaling the time and data complexities, we have $n - m/2 = m$, which shows $m = \frac{2}{3}n$. Thus the optimal complexities of this attack are $C_T = \mathcal{O}(n^2 2^{\frac{2}{3}n})$ and $C_D = \mathcal{O}(2^{\frac{2}{3}n})$. These arguments apply to both $\mathsf{LFSR_x}$ and $\mathsf{LFSR_y}$.

Remark 4. The total time of the attack is composed of the time to filter the blocks of data with desired properties, and of the time to further process the filtered blocks. Although the unit of examination time of these two phases are not equal, we ignore this difference to simplify the analysis.

In another scenario in [79], it is suggested to wait for a segment of length m containing at most w ones (zeros) and make the assumption that only half of the zeros (ones) come from the $\mathsf{LFSR_x}$. All the ones (zeros) and the remaining zeros (ones) are assumed to come from the $\mathsf{LFSR_y}$. This is true with probability $\beta = 2^{-w}\binom{m-w}{(m-w)/2}2^{-(m-w)}$. The time and data complexities of this attack are then $C_T = n^2 2^{n-(m-w)/2}\beta^{-1}$ and $C_D = 2^{m-1}\binom{m}{w}^{-1}\beta^{-1}$, respectively. With $w := \alpha m$, ignoring the constant and polynomial terms, and equaling the time and data complexities, we have $n-(1-\alpha)m/2+\alpha m = m-h(\alpha)m+\alpha m$, which results in $m = n/(3/2-\alpha/2-h(\alpha))$. The minimum value of the exponents $m(1-h(\alpha)+\alpha)$ is $0.6406n$, which is achieved for $\alpha \approx 0.0727$ (and hence $m = 0.9193n$ and $w = 0.0668n$). Therefore, the optimal complexities are $C_T = \mathcal{O}(n^2 2^{0.64n})$ and $C_D = \mathcal{O}(2^{0.64n})$. Note that this complexity is only for reconstruction of the initial state of $\mathsf{LFSR_x}$. The complexity for recovering the initial state of $\mathsf{LFSR_y}$ highly depends on the position of ones (zeros) in the block. In the best case, the block starts with w ones (zeros) and the complexity becomes $C_T = n^2 2^{n-(m+w)/2}$. In the worst case, the attacker has to search for the positions of ones (zeros), and the complexity becomes $C_T = \binom{(m+w)/2}{w} n^2 2^{n-(m-w)/2}$. It is difficult to give an average complexity, but we expect that it is close to the worst case complexity. With $m = 0.9193n$ and $w = 0.0668n$, this gives $C_T = \mathcal{O}(n^2 2^{0.69n})$ to recover the initial state of $\mathsf{LFSR_y}$. Consequently, as a distinguishing attack, this scenario operates slightly better than the previous one, but as an initial state recovery it is slightly worse.

5.4 New Reduced Complexity Attack

Before we describe our attack in detail, let us introduce some notations. We use $A := (a_i)$ for a general binary sequence, $A_k^m := (a_i)_{i=k}^m$ for a segment of it and $A^m := (a_i)_{i=1}^m$ for a prefix of length m. The number of 1's in A is denoted by $\mathrm{wt}(A)$. We define the first derivative of A as $(a_i \oplus a_{i+1})$ and denote it by $\dot{A}$. Let C, X, Y and Z denote the regular output sequences of $\mathsf{LFSR_c}$, $\mathsf{LFSR_x}$, $\mathsf{LFSR_y}$ and the output sequence of the ASG itself, respectively. The initial state of the LFSR's can be represented by $c = C^n$, $x = X^n$ and $y = Y^n$. The output sequence of the ASG of size m is denoted by $z = Z^m$.

5.4.1 Sampling Resistance

Let us consider the augmented function of ASG with $S : \mathbb{F}^{3n} \to \mathbb{F}^m$. Any initial state (c, x, y) of ASG which can produce z, a given prefix of size m of the output sequence of ASG, is called a preimage of z. As in the previous chapter, the sampling resistance is defined as 2^{-m} where m is the maximum value for which we can efficiently produce all preimages of m-bit outputs. As will be shown in this subsection, the low sampling resistance of ASG is an essential ingredient for our attack. Let $S^{-1}(z)$ of size $U_z = |S^{-1}(z)|$ denote the set of all preimages of z. Based on the divide-and-conquer linear consistency attack, introduced in Sect. 5.2, we can compute $S^{-1}(z)$ according to Alg. 3. Let us discuss the complexity of Alg. 3. If $U_z \leq 2^n$, then the overall complexity is 2^n, because the complexity of Steps 3 to 8 are $\mathcal{O}(1)$. On the other hand, if $U_z > 2^n$, then Steps 3

to 8 introduce additional solutions, and overall complexity is about U_z. The following statement is given under the assumption of balancedness, *i.e.* the average number of preimages of ASG for any output z of m bits is about 2^{3n-m}, where $m \leq 3n$.

Algorithm 3 Sampling of ASG

Input: Output sequence z of m bits.
Output: Find $S^{-1}(z)$ with all preimages of z.
1: Initially, set $S^{-1}(z) = \emptyset$.
2: **for all** non-zero initial states c **do**
3: Set $\mathcal{X} = \mathcal{Y} = \emptyset$.
4: Compute C^m, a prefix of length m of the output sequence of $\mathsf{LFSR_c}$.
5: Based on C^m, split up z into X^w and Y^{m-w}, where $w = \mathrm{wt}(C^m)$.
6: Add all (non-zero) x to $\mathcal{X}$, if $\mathsf{LFSR_x}$ can generate X^w.
7: Add all (non-zero) y to $\mathcal{Y}$, if $\mathsf{LFSR_y}$ can generate Y^{m-w}.
8: For all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, add (c, x, y) to the set $S^{-1}(z)$.
9: **end for**

Statement 1. *Time complexity of Alg. 3 is* $C_T = \mathcal{O}(\max(2^n, 2^{3n-m}))$.

With the previous definition of sampling resistance, this algorithm can be considered as an efficient sampling algorithm iff $U_z \geq \mathcal{O}(2^n)$ or equivalently $m \leq 2n$. That is, the sampling resistance of ASG is about 2^{-k} with $k = 2n$ the total length of the two stop/go LFSR's. A related problem is how to find a multiset $\mathcal{A}$ with T uniformly random independent elements of $S^{-1}(z)$. We suggest to modify Alg. 3 as follows: $S^{-1}(z)$ is replaced by $\mathcal{A}$ and T is added as another input parameter. In Step 2, a uniform random (non-zero) initial state c is chosen, and Steps 3 to 8 are not modified. The new Steps 2 to 8 are then repeated, until T preimages have been found. This modified algorithm will be referred to as Alg. 3B. We assume correctness of the algorithm, *i.e.* the preimages found with Alg. 3B are uniformly random elements of $S^{-1}(z)$ (for which we will give experimental evidence). The following statement is presented under the assumption that the average number of preimages of ASG for any output z, given some fixed initial state of $\mathsf{LFSR_c}$, is about 2^{2n-m}, where $m \leq 2n$.

Statement 2. *Time complexity of Alg. 3B is* $C_T = \mathcal{O}(T)$ *for* $m \leq 2n$, *and* $C_T = \mathcal{O}(\min(2^n, T2^{m-2n}))$ *for* $m > 2n$, *where* $1 \leq T \leq \mathcal{O}(2^{3n-m})$.

5.4.2 Conditional Distribution of the Initial States

With the sampling algorithm described in Sect. 5.4.1, we can find T random preimages of an output sequence z. The natural question which arises is *how large should* T *be so that our subset contains the correct initial state?* The answer is related to the conditional distribution of different initial states of $\mathsf{LFSR_x}$ which can produce a given segment of length m of the output sequence of the ASG. Consider the following two general propositions:

Proposition 5. *Let $X_0, \ldots, X_T$ be a sequence of i.i.d. random variables, defined over the finite set $\{s_1, \ldots, s_N\}$ with probability distribution $p_i := \Pr(X_j = s_i)$. Then, the probability $P := \Pr(X_0 \in \{X_1, \ldots, X_T\})$ that a realization of $X_1, \ldots, X_T$ contains a realization of X_0 is*

$$P = 1 - \sum_{i=1}^{N} (1 - p_i)^T p_i \ . \tag{5.2}$$

Proof. The probability P can be expressed as

$$\begin{aligned}
&\Pr(X_0 \in \{X_1, \ldots, X_T\}) = 1 - \Pr(X_0 \neq X_j,\ 1 \leq j \leq T) \\
&= 1 - \sum_{i=1}^{N} \Pr(X_0 \neq X_j,\ 1 \leq j \leq T \mid X_0 = s_i) \cdot \Pr(X_0 = s_i) \\
&= 1 - \sum_{i=1}^{N} \Pr(s_i \neq X_j,\ 1 \leq j \leq T\ ) \cdot \Pr(X_0 = s_i) \\
&= 1 - \sum_{i=1}^{N} (1 - p_i)^T p_i \ .
\end{aligned}$$

□

Proposition 6. *Let $H := -\sum_{i=1}^{N} p_i \log_2(p_i)$ be the entropy of random variable X_j. With about $T = 2^H$, the probability $\Pr(X_0 \in \{X_1, \ldots, X_T\})$ is significant.*

Proof. From Prop. 5 we have $\Pr(X_0 \in \{X_1, \ldots, X_T\}) = 1 - \sum_{i=1}^{N} (1 - p_i)^T p_i$. With the assumption $Tp_i \ll 1$, we obtain $(1-p_i)^T \approx 1 - Tp_i$, which gives the approximation $\Pr(X_0 \in \{X_1, \ldots, X_T\}) \approx 1 - \sum_{i=1}^{N} (1 - Tp_i) p_i = T \sum_{i=1}^{N} p_i^2$. Assuming $\Pr(X_0 \in \{X_1, \ldots, X_T\}) \approx 1$, we have $T \approx 1 / \sum_{i=1}^{N} p_i^2$, or equivalently $T \approx 2^G$ with $G := -\log_2 \sum_{i=1}^{N} p_i^2$. This can be compared with the entropy function H. Both H and G are approximated with a multivariate Taylor series of order 2 at the point p_0, such that $p_i = p_0 + \varepsilon_i$. If T_2 denotes the second order part, this gives

$$\begin{aligned}
T_2(H) &= \frac{N p_0}{\ln 2} - \frac{1}{\ln 2} - \log_2 p_0 \\
T_2(G) &= \frac{2}{\ln 2} - \frac{2}{N p_0 \ln 2} - \log_2 N - \log_2 p_0^2 \ .
\end{aligned}$$

Now let $p_0 := 1/N$, then we have $T_2(H) = \log_2 N$ and $T_2(G) = -\log_2 N + 2 \log_2 N = \log_2 N$. Consequently, the difference becomes $T_2(H) - T_2(G) = 0$, hence $H = G$ of order 2 on the points $p_i = 1/N$. □

Remark 5. The quantity $G := -\log_2 \sum_{i=1}^{N} p_i^2$ is the Rényi entropy of order 2. It is known that guessing a random value, drawn from a *known* nonuniform probability distribution, on average requires the number of steps related to the Rényi entropy of order 2, *e.g.* see [117] or references therein. Prop. 6 shows that this is still true when the distribution is not directly known but can be simulated. One can directly use this entropy instead

of Shannon entropy which is only an approximation in this regard, however, we prefer to use the better known Shannon entropy. For the case $p_i = 1/N$ we have $G = H = \log_2 N$, hence $T = N$ and $P = 1 - \sum_{i=1}^{N}(1-1/N)^N(1/N)$. For $N \gg 1$ we have $(1-1/N)^N \approx e^{-1}$ which shows $P \approx 1 - e^{-1} \approx 0.63$. We guess that in general we have $P \geq 1 - e^{-1}$. Our extensive simulations for several distributions verify this conjecture.

To apply these propositions to the situation of ASG, let $S^{-1}(z|x)$ of size U_x be a subset of $S^{-1}(z)$, defined by $\{(c, x, y) \in S^{-1}(z)\}$ for fixed x. The conditional probability for a fixed initial state x of $\mathsf{LFSR_x}$ is then defined by $p(z|x) = U_x/U_z$. Consequently, we need to draw about $T = 2^{H_x}$ uniformly random elements of $S^{-1}(z)$ to include the correct initial state of $\mathsf{LFSR_x}$ where H_x is the conditional entropy of the initial state of the $\mathsf{LFSR_x}$ given z, defined by

$$H_x(z) = -\sum_x p(z|x) \cdot \log_2 p(z|x) \ . \tag{5.3}$$

The same argument applies to $\mathsf{LFSR_y}$, and the symmetry of ASG motivates the simplification $H := H_x = H_y$ (if not mentioned otherwise). Another natural question is the expected number of different elements Q drawn in this sample of size T. This is related to the Coupon-Collector Problem with non-uniform distribution. However, we can assume that $Tp \ll 1$, which results in $Q \approx T$.

Remark 6. Any adversary who *would know* the distribution $p(z|x)$ could try to recover the unknown initial state of $\mathsf{LFSR_x}$ by considering the most probable initial state first, then the second most probable one and so on. Here, to cope with *unknown* distribution $p(z|x)$, we simulate it by choosing uniformly random elements of $S^{-1}(z)$ (where element x is chosen with probability $p(z|x)$). This procedure is similar to [97] in which an equivalent description of the underlying cipher was used, for which the initial states were no longer equiprobable.

Remark 7. As mentioned in Sect. 5.2.4, it has been suggested in [68] to take advantage of the posterior probabilities of the individual bits of the regularly clocked $\mathsf{LFSR_x}$ and $\mathsf{LFSR_y}$ sequences, when conditioned on a given segment of the output sequence for faster reconstruction of LFSR initial states. Our attack can be considered as a generalization of this attack in which we take advantage of the posterior probabilities of the initial states rather than individual bits, when conditioned on a given segment of the output sequence. Although unlike [68] we are able to give an estimation for the time and data complexities of our attack, a theoretical analysis of the conditional entropy of the initial states remains an open problem, see Sect. 5.5.1.

5.4.3 Description of the Attack

In the basic edit probability correlation attack on the ASG [67, 66], the edit probability is computed for each of the 2^n possible initial states of $\mathsf{LFSR_x}$ (given a segment of length $N \approx 40n$ of the output sequence of the ASG) to find the correct initial state. This is repeated also for $\mathsf{LFSR_y}$, and finally the initial state of $\mathsf{LFSR_c}$ can be recovered. In our

improved attack, we take the output sequence z of m bits into account to compute a smaller multiset $\mathcal{A}$ of candidates of initial states, which is of size T and contains the correct initial state of $\mathsf{LFSR_x}$ (resp. $\mathsf{LFSR_y}$) with some probability P, see Prop. 5. The multiset $\mathcal{A}$ is constructed with Alg. 3B. In Alg. 4, we give a formalization of this attack.

Remark 8. One would think that it is better to compute the edit probability between the Z^N and only the LFSR output sequence of all *distinct* initial states suggested by multiset $\mathcal{A}$ to avoid processing the same initial state several times. However, this needs memory of $\mathcal{O}(|\mathcal{A}|)$ and extra effort to keep the track of the non-distinct initial states. Since $|\mathcal{A}| \approx T$ the achieved gain is negligible and therefore we alternatively compute the edit probability at the time where a preimage is found.

Algorithm 4 Attack on ASG

Input: Parameters T, m, N, output Z^N.
Output: Recover the initial state of ASG with some error probability p_e.
1: Given the segment $z = Z^m$, find T preimages using Alg. 3B.
2: Compute the edit probability between Z^N and the output sequence for each suggested initial state.
3: Choose the most probable candidates for $\mathsf{LFSR_x}$ resp. $\mathsf{LFSR_y}$.
4: Recover $\mathsf{LFSR_c}$ and verify the validity, see Sect. 5.2.3.

Parameters for the Entropy. The complexity of the attack is related to the conditional entropy H. However, for large instances of ASG, the conditional probabilities and hence H are unknown. To be able to evaluate our attack and give an explicit expression for the data and time complexities, we need to know the relation between conditional entropies H and all parameters which can possibly affect them. The parameters are LFSR's feedback polynomials and the output prefix z, which implicitly include the lengths of LFSR's and output segment length as well. In our simulations we noticed that feedback polynomials have almost no effect and the only important parameters are LFSR lengths n, the size of the output segment m (as larger values of m reduce uncertainty about the correct preimage), and the weight w of the output segment z or the weight w of the first derivative of the output segment z (as will be shown in our simulations). The entropy is significantly reduced if $|\mathrm{wt}(z)/m - 0.5| \gg 0$ (*i.e.* many zeros or ones) or if $\mathrm{wt}(\dot{z})/m \ll 0.5$ (*i.e.* many runs of zeros or ones). This can be explained by the fact that a biased output segment results in a biased LFSR segment, and we will refer to such outputs as *weak* outputs. In Sect. 5.5.1, we will predict the average value of H depending on these parameters using some regression analysis, hence $H = f(n, m, w)$.

Time Complexity. Let us discuss time complexity of Alg. 4. According to Prop. 6, we set $T = 2^H$. Complexity of Step 1 is described in Statement 2. Computation of the edit probability distance of a single preimage takes about $\mathcal{O}(n^2)$, hence complexity of Step 2 is at most $\mathcal{O}(n^2 T)$. Finally, the complexity of Step 4 is $\mathcal{O}(2^{0.27n})$, which can be neglected here.

Statement 3. *Time complexity of Alg. 4 is about* $C_T = \mathcal{O}(n^2 2^H)$ *for* $m \leq 2n$*, and* $C_T = \mathcal{O}(2^{H+m-2n})$ *for* $m \gg 2n$*.*

This should be compared to the attack by Golic *et al.* of complexity $C_T = \mathcal{O}(n^2 2^n)$ using an output sequence of length about $C_D = 40n$ which was described in Sect. 5.2.3, and Johansson's attack of complexity $C_T = \mathcal{O}(n^2 2^{\frac{2}{3}n})$ using an output sequence of length $C_D = \mathcal{O}(2^{\frac{2}{3}n})$ as described in Sect. 5.3.

Data Complexity. The parameter w has some influence on the data complexity of our attack. Once we know that the weight of z is at most w or at least $m - w$, or that the weight of the first derivative of z is at most w, a prefix of length about $N = 40n$ suffices to recover the initial states, see [66]. However, in order to obtain such an output segment Z_{t+1}^{t+m} for some t, the required amount of keystream bits is $C_D = 2^m (3 \sum_{i=0}^{w} \binom{m}{i})^{-1}$. This can be roughly approximated by $C_D = \mathcal{O}(2^{m(1-h(w/m))})$.

Success Probability. The error probability p_e of the attack depends on three events: 1) The probability that our multiset $\mathcal{A}$ of size $T = 2^H$ contains the correct initial state. 2) The probability that our prediction of the entropy gives at least H. 3) The success probability of the edit distance correlation attack. The first probability corresponds to P according to Eq. 5.2. The second probability comes from the fact that we use an estimation of the average value of H instead of the exact value of H.

5.5 Experimental Results

In this section, we give experimental results on ASG. We estimate the conditional entropy, give a detailed discussion of the complexity for different scenarios and present an example of an attack.

5.5.1 Distribution of Initial States

For specific instances of ASG, we investigate the distributions of initial states. Here, ASG is small enough such that an exact computation of initial states with Alg. 3 is feasible. We use registers of the same length, but our results do not significantly change if the lengths are pairwise coprime and about the same, as suggested in [98]. The following example has illustrative character: First, we compute the distributions for one fixed output sequence. Second, the block size m is varied for average-weighted output sequences. Third, an output sequences of low weight is investigated.

Example 16. Consider a setup with $n = 20$ and some randomly chosen primitive feedback polynomials. Fix a random output sequence z of $m = 40$ bits according to

$$z = \texttt{1110110110100101010000100100101011000110} \ .$$

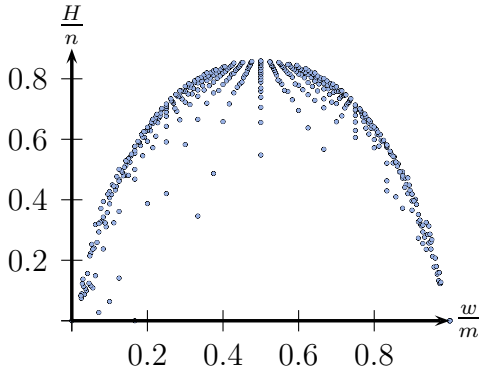

Figure 5.1: H/n versus w/m for all $0 \leq w \leq m$ and $5 \leq n \leq 21$.

The number of preimages is $U_z = 1\,046\,858 = 2^{20.00}$, and the entropies are $H_c = 17.49$, $H_x = 17.32$, and $H_y = 17.34$. If this output is padded by the 2 additional bits `01`, then the number of preimages becomes $U_z = 264\,265 = 2^{18.01}$ and the entropies are $H_c = 16.26$, $H_x = 16.46$, and $H_y = 16.45$. On the other hand, consider the following output sequence for $m = 40$ and with weight $w = 7$,

$$z = \texttt{0001010000100000000110000001000100000000}\ .$$

The number of preimages for this low-weight output sequence is $U_z = 1\,117\,725 = 2^{20.09}$, with entropies $H_c = 17.39$, $H_x = 12.24$, and $H_y = 12.63$. □

Let us discuss this example. The number of preimages is about 2^{60-m}, as expected. In all three registers, the entropy is not maximal for the random output sequence of size $m = 40$. This may be explained by the fact that sequences are not fully random, as they satisfy a linear recurrence. In the stop/go LFSR's, the entropy is strongly reduced for outputs of low weight, without any losses in the number of preimages. Notice that H_c does not depend on the weight of the output, which is optimal for efficient sampling.

In the following we will focus on the case $m = 2n$. The entropy H of the stop/go LFSR's is exactly determined for different values of n and w, where $n = 5, \ldots, 21$ and $w = 0, \ldots, m$. More precisely, given some n (and randomly chosen primitive feedback polynomials), we determine the average entropy H using 500 randomly chosen outputs of weight w. The values of H/n as a function of w/m are shown in Fig. 5.1. The inner dots in this figure relate to smaller values of n, and the outer dots relate to larger values of n. A convergence behavior of H/n for increasing n is perceivable from this figure.

It turns out that H/n can be well approximated by a scaled binary entropy function, namely $H/n \approx \gamma \cdot h(w/m)$ with $0 < \gamma \leq 1$ depending on n. Notice that $\gamma = \max_w(H/n)$, which can be well approximated by $\gamma \approx 1 - 1/(0.19n + 3.1)$. Fig. 5.2 shows some additional figures of the average entropy, together with our approximations using nonlinear regression. Fig. 5.3 compares the average value of the entropy as a function of the weight of the output sequence and as a function of the weight of the derivative of the output sequence. Consequently, with this regression analysis, the average value of the entropy is

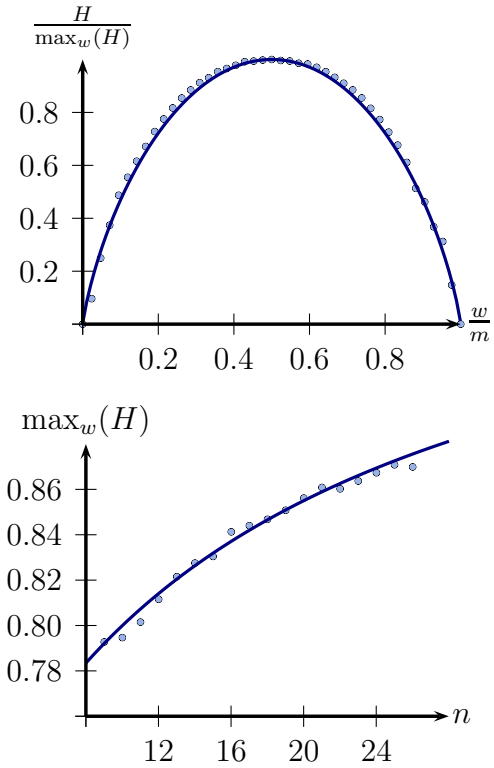

Figure 5.2: Up: $H/(\max_w(H))$ versus w/m for $n = 21$, approximated by the entropy function. Down: $\max_w(H)$ versus n, approximated by $\gamma(n)$.

approximated by:

$$H(n, m, w) \approx \gamma(n) \cdot n \cdot h\left(\frac{w}{m}\right) \tag{5.4}$$

$$\gamma(n) \approx 1 - \frac{1}{0.19n + 3.1} \, . \tag{5.5}$$

In the case $w = \mathrm{wt}(\dot{z})$ the shape is not symmetric, however it seems that for $w/m < 0.5$ for a fixed n the figures of H/n versus w/m are well comparable regardless of what w represents ($w = \mathrm{wt}(\dot{z})$ or $w = \mathrm{wt}(z)$), see Fig. 5.3. For $m > 2n$, the expected entropy does not correspond to this functional form anymore. The maximum of H against w/m decreases linearly with m, but the graph of H/n versus w/m is broader compared to $h(w/m)$, which means that a reduction of the entropy requires an output of very low weight. We do not further investigate this scenario.

5.5.2 Complexity of the Attack

Our attack allows different time/data trade-offs. We describe the complexity of our attack for $m = 2n$ and different values of parameters n and w. According to Statement 3, time complexity of our attack is $C_T = \mathcal{O}(n^2 2^H)$. Including the approximation for H, we obtain $C_T = \mathcal{O}(n^2 2^{\gamma n h(w/m)})$. Given an random output sequence, the complexity of our attack

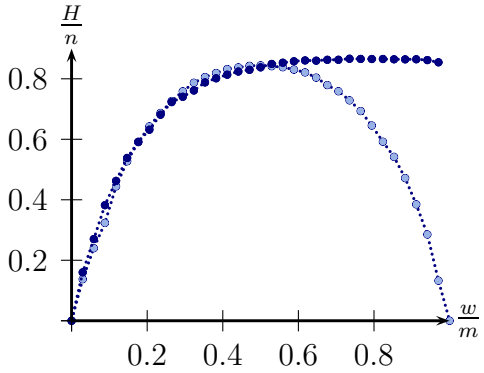

Figure 5.3: H/n versus w/m for $n = 17$ in two cases: $w = \mathrm{wt}(\dot{z})$ and $w = \mathrm{wt}(z)$.

is $C_T = \mathcal{O}(n^2 2^{\gamma n})$. In this case the data complexity is minimal and our attack should be compared to the attack by Golic *et al.* [66] which shows an improvement by a factor $2^{(1-\gamma)n}$. In the limit $\gamma \to 1$ (hence for $n \to \infty$), our attack reduces to the previous attack. However for moderate values there is some gain. For example, we expect $\gamma = 0.945$ for $n = 80$, which gives an improvement of a factor $2^{4.4}$.

Reduced complexity attacks can be mounted by using weak outputs. This can be compared to the attack by Johansson [79]. Asymptotical data complexity of our attack becomes $C_D = \mathcal{O}(2^{m(1-h(w/m))})$. Similar to what we do in Sect. 5.3, the optimized complexity is achieved if time and data complexities are almost equal. Considering only the exponential terms and $\gamma = 1$, this happens when $h(w/m)n = m(1 - h(w/m))$, that is $h(w/m) = 2/3$ and hence $w \in \{0.174m, 0.826m\}$. The asymptotical complexities become $C_T = \mathcal{O}(n^2 2^{\frac{2}{3}n})$ and $C_D = \mathcal{O}(2^{\frac{2}{3}n})$, which is identical to the complexities of the attack by Johansson, see Sect. 5.3. However, compared to the simple attack in [79], it is clear that our attack allows for more flexibility in the structure of the output sequence: the weight can be arbitrary, we can also use outputs of low weight derivative, and we do not need a hypothesis about the origin of the output bits. With a more subtle (non-asymptotical) investigation of the complexities, we show that data (and/or time) complexity can be significantly reduced with our attack. More precisely, we evaluate the exact complexities of our and Johansson's attack for reasonable value of n. Regarding Johansson's attack, consider the special point $m = \frac{2}{3}n$ in the time/data tradeoff curve. For $n = 80$, this gives $C_T = 2^{69.4}$ and $C_D = 2^{55.2}$. If we choose $w = 0.21m$ in our attack, we obtain about the same time complexity and require only $C_D = 2^{42.3}$ data. This is an improvement of a factor $2^{12.9}$ (notice that a significant reduction can be expected even for $\gamma = 1$).

5.5.3 Example of an Attack

In this section, we present a large-scale example of a partial attack. We fix a random initial state in all three registers, such that the corresponding output sequence has weight $w = 0.174m$. Then, H is computed according to Eq. 5.4 and 5.5. Alg. 3B is used to compute the multiset $\mathcal{A}$ of size $T = 2^H$, and we check if the correct initial state of the LFSR_x (resp. LFSR_y) is included in $\mathcal{A}$. This is repeated several times, in order to determine the success

probability P. In addition, time complexity of Alg. 3B is measured experimentally: For each choice of c, the complexity is increased by the number of preimages found (and by one if no preimage can be found), see Remark 8. For $m = 2n$, this should be compared to $C_T = \mathcal{O}(T)$, see Statement 2. Notice that we do not implement the edit probability correlation attack and rely on the results of [66].

Example 17. Let $n = 42$ and fix a random initial state such that the corresponding output sequence z of $m = 84$ bits has weight $w = 14$. The expected entropy becomes $H = 24.16$, we set $T = 2^H = 18\,782\,717$ and apply Alg. 3B. This is repeated 200 times, and the correct initial state of LFSR_x is found in 84 cases which shows a success probability of $P = 0.42$ for our algorithm. The average time complexity of the sampling algorithm is $2^{25.35}$. □

5.6 Summary

A reduced complexity attack on the Alternating Step generator (ASG) has been presented, the success of which has been confirmed experimentally. For comparison, the complexity of the best previous attack has been determined and described in closed form. Estimates of the overall complexity of our new attack are shown to improve the complexity of the previous attack. Our attack allows for greater flexibility in known output data constraints, and hence for lower data complexity, for being successful. The attack method demonstrates the usefulness of a quite general attack principle exemplified in the case of ASG: to exploit low sampling resistance and heavily biased inputs for outputs satisfying certain constraints.

Chapter 6

Analysis of F-FCSR

We have seen that algebraic attacks (and related concepts) can be a real threat for LFSR-based stream ciphers, but are much less efficient for stream ciphers with nonlinear driving devices such as FCSR's. In this chapter, we investigate the security of the eSTREAM phase 3 stream cipher candidate F-FCSR. Our analysis shows a link to an equivalent representation of the FCSR.

6.1 Introduction

As a potential replacement device of LFSR's, feedback shift registers with carry (FCSR's) have been investigated. The eSTREAM phase 3 candidate F-FCSR consists of an FCSR with multiple linear (thus hardware-efficient) filters applied to the main register of the FCSR automaton. The security of F-FCSR was investigated in different directions, and it was recently observed in [116] that an equivalent description exists. It is an open question if this equivalent description results in a simplified structure to be used in a cryptanalytic attack (as it was the case for ASG in the previous chapter). The new description reveals that (1) only one variable of the main state is updated in each iteration, (2) the memory is very small, but (3) with a transformed filter. We focus our analysis on this alternative description of F-FCSR.

6.2 Theoretical Background

An FCSR can be represented in Fibonacci or Galois architecture, see [70]. In this section, we review the definition and some basic theory from [5, 6, 70, 83] on FCSR's in both representations.

6.2.1 2-Adic Numbers and Periods

Following the definition in [83], we call a state of a finite state machine (an FCSR, for instance) periodic if, left to run, the machine will return to that same state after a finite number of steps. Similarly, we call a sequence $U = (u_0, u_1, \ldots)$ periodic (or strictly

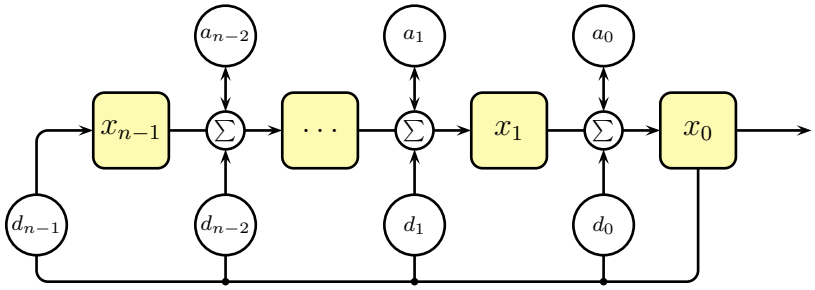

Figure 6.1: FCSR with Galois architecture.

periodic) with period T if $u_{i+T} = u_i$ for all $i \geq 0$. We call a sequence U eventually periodic if there exists a $t \geq 0$ such that $U' = (u_t, u_{t+1}, \ldots)$ is periodic. A 2-adic integer is a formal power series $\alpha = \sum_{i=0}^{\infty} u_i 2^i$ with $u_i \in \{0, 1\}$. The collection of all such formal power series forms the ring of 2-adic numbers. This ring especially contains rational numbers p/q where p and q are integers and q is odd. Such rational numbers and eventually periodic binary sequences are linked by the following well-known theorem [83].

Theorem 3. *There is a one-to-one correspondence between rational numbers $\alpha = p/q$ (where q is odd) and eventually periodic binary sequences u which associates to each such rational number α the bit sequence $u = (u_0, u_1, u_2, \ldots)$ of its 2-adic expansion. The sequence u is strictly periodic if and only if $\alpha \leq 0$ and $|\alpha| < 1$.*

6.2.2 Galois FCSR's

Description. A Galois FCSR (which is similar to a Galois LFSR) consists of n binary register cells $x = (x_0, \ldots, x_{n-1})$ with some fixed binary feedback positions $d = (d_0, \ldots, d_{n-1})$, and $n-1$ binary memory cells $a = (a_0, \ldots, a_{n-2})$. We also use the integer representation $x = \sum_{i=0}^{n-1} 2^i x_i$ (correspondingly for d and a). Starting from an initial configuration (x, a), x_0 is output, and the sums $\sigma_i = x_{i+1} + a_i d_i + x_0 d_i$ are computed for $0 \leq i < n$ (with $x_n = 0$, $a_{n-1} = 0$). Then, the state is updated by $x_i \leftarrow \sigma_i \mod 2$ for $0 \leq i < n$, and $a_i \leftarrow \sigma_i \operatorname{div} 2$ for all $0 \leq i < n-1$, see Fig. 6.1.

Evolution of the States. We consider here the special case where memory bits of a Galois FCSR are only present on those positions with feedback (which means that the effective number of memory bits is $l \leq n$, and a can only have some restricted values). In this case, the Galois FCSR can be described by the connection integer $q = 1 - 2d$. The initial state is denoted (x, a), with an associated value $p = x + 2a$ (assuming that x is not the all-zero or all-one state). Note that different states (x, a) may lead to the same p, *i.e.*, the function to compute p from (x, a) is not injective. The sequence generated by the FCSR is the 2-adic expansion of p/q, *i.e.*, the output sequence depends only on p and q [70]. In other words, let p^t be the state at time t, with initial state $p^0 = p$. Then the Galois FCSR

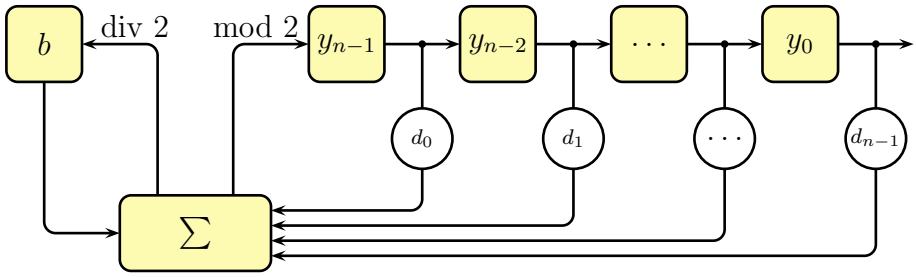

Figure 6.2: FCSR with Fibonacci architecture.

produces $2p^{t+1} = p^t \mod q$, or

$$p^t = 2^{-t}p \mod q \ . \tag{6.1}$$

The output bit at time t is then $z^t = p^t \mod 2$. It is $0 \leq p \leq |q|$, hence the output sequence is periodic (see Th. 3). According to Eq. 6.1, the period of the output sequence is the order of 2 modulo q. The maximum value of the period is $|q| - 1$ and can only be reached if $|q|$ is a prime [5]. In the case of a maximum-length FCSR, the transition graph representing the evolution of the states (x, a) consists of a main cycle of length $|q| - 1$ with small paths converging to it. It is known [8] that any state (x, a) converges to the main cycle (*i.e.* it synchronizes) after at most $n+4$ iterations. Furthermore, a single cycle of the output consists of two half periods (which are binary complements of each other).

6.2.3 Fibonacci FCSR's

Description. A Fibonacci FCSR consists of a main register $y = (y_0, \ldots, y_{n-1})$ of n bits, with some fixed binary feedback taps $d = (d_0, \ldots, d_{n-1})$ and an additional memory register b of l bits. Starting from an initial configuration (y, b), y_0 is output, the sum $\sigma = b + \sum_{i=0}^{n-1} y_i d_{n-i-1}$ is computed, and the registers are updated according to $y \leftarrow (y_1, \ldots, y_{n-1}, \sigma \mod 2)$ and $b \leftarrow \sigma$ div 2, see Fig. 6.2. If the Fibonacci FCSR is in a periodic state, then the value of the memory b is in the range $0 \leq b < \mathrm{wt}(d)$, where $\mathrm{wt}(d)$ denotes the Hamming weight of d, see [70].

Evolution of the States. The connection integer is again defined by $q = 1 - 2d$, and the state is represented by the integer p. Then, the output sequence of the Fibonacci FCSR is again the 2-adic expansion of p/q. However, p does not correspond to $y + 2b$ here, but to

$$p = b2^n - \sum_{k=0}^{n-1} \sum_{j=0}^{k} d_{j-1} y_{k-j} 2^k \tag{6.2}$$

where $d_{-1} = -1$, see [70]. If memory bits of a Galois FCSR are only present on those positions with feedback (*i.e.*, for Galois FCSR's represented by a connection integer q), then the Galois FCSR can be mapped to a Fibonacci FCSR (and vice versa) with the same connection integer $q = 1-2d$ such that both produce the same output. Note that a Galois

FCSR can be implemented more efficiently than a Fibonacci FCSR since the additions may be carried out in parallel.

6.3 Sequences Produced by a Single Galois Register Cell

In a Galois FCSR, the values x_i in the main register are modified in each cycle, and not only shifted. Assume an initial state (x, a) and a connection integer $q = 1 - 2d$. Then, according to Th. 4 in [5], there exists some p_i such that the sequence x_i^t of values produced by a fixed register cell i in a Galois FCSR corresponds to the 2-adic expansion of p_i/q. From [7], we know that $p_i = F_i(x, a) \cdot q + M_i \cdot p$ with $F_i(x, a) = \sum_{j=i}^{n-1}(x_j + 2a_j)2^{j-i}$ and with constants $M_i = 2\sum_{j=i}^{n-1} d_j 2^{j-i}$. The following proposition is a simple consequence of this for periodic states:

Proposition 7. *Consider a maximum-length Galois FCSR with initial state (x, a) and output sequence $p^t \mod 2$, where $p^0 = x + 2a$. If (x, a) is a periodic state, the sequence x_i^t of a fixed register cell i corresponds to $p^{t+s_i} \mod 2$ with a phase shift $s_i = -\log_2(M_i) \mod q$ and $M_i = 2\sum_{j=i}^{n-1} d_j 2^{j-i}$.*

Proof. If (x, a) is periodic, the 2-adic expansions of p_i/q have to be strictly periodic for all i. Th. 3 implies that $0 \leq p_i < |q|$, hence $p_i = p_i \mod q = M_i \cdot p \mod q$. In a maximum-length Galois FCSR, each possible value of $p_i \mod q$ is passed after a number of s_i iterations of p, hence $p_i = 2^{-s_i} p \mod q$, and we can set $M_i = 2^{-s_i} \mod q$. □

Note that the phase shifts s_i are independent of the initial state p and depend on i (and q) only. Here is an example:

Example 18. Consider the toy example of [5] with $q = -347$, hence $n = 8$ and $d = 174$. The output of the FCSR is strictly periodic with period $-q - 1 = 346$. We find $M_0 = 1$, $M_1 = 174$, $M_2 = 86$, $M_3 = 42$, $M_4 = 20$, $M_5 = 10$, $M_6 = 4$, $M_7 = 2$. The phase shifts are $s_0 = 0$, $s_1 = 1$, $s_2 = 23$, $s_3 = 250$, $s_4 = 67$, $s_5 = 68$, $s_6 = 344$, $s_7 = 345$. □

6.4 A Canonical Representative

Note that more than one state (x, a) may be mapped to $p \in \mathbb{Z}_{|q|+1}$. We define an equivalence relation $\sim$ on the set of FCSR-states in Galois representation by

$$(x, a) \sim (x', a') \Leftrightarrow x + 2a \equiv x' + 2a' \mod q .$$

With the following proposition, we can define a canonical representative for the equivalence classes $[p]$.

Proposition 8. *For a state (x, a) with $p = x + 2a$ of a maximum-length Galois FCSR with connection integer q, the only strictly periodic state in the equivalence class $[p]$ is the state (x', a') with $x'_i = M_i \cdot p \mod q \mod 2$ and $a' = (p - x')/2$.*

PROOF. Let $p = x + 2a$. We have $x' + 2a' = x' + 2\frac{p-x'}{2} = x' + p - x' = p$, hence $(x', a') \sim (x, a)$. In the case $p = 0$, we have $(x, a) = (0, 0) = (x', a')$, and $(x^t, a^t) = (0, 0)$ for all t, so (x', a') is periodic. Similarly for $p = |q|$, the only possible state (x, a) is $(2^n - 1, d - 2^{n-1})$, and this state is periodic [5]. If $p \neq 0$, the state transition graph representing the evolution of the states (x^t, a^t) consists of a main cycle of length $|q| - 1$ and paths converging to it. Hence, for each state (x, a) there exists exactly one equivalent state $(\tilde{x}, \tilde{a})$ that lies on the main cycle. For this state $(\tilde{x}, \tilde{a})$, the sequences $\tilde{x}_i^t$ have to be strictly periodic. Due to Prop. 7, the first bit of the 2-adic expansion of $\tilde{p}_i/q$ and hence x'_i is equal to $\tilde{p}_i \mod 2$ with $p_i = M_i \cdot p \mod q$. Moreover, $\tilde{a}$ is uniquely determined by $\tilde{x}$ and p, which implies $(\tilde{x}, \tilde{a}) = (x', a')$. □

This suggests to define the state (x', a') as the canonical representative for the equivalence class $[x' + 2a']$. Here is an example:

Example 19. Let $q = -347$, hence $n = 8$ and $d = 174$. For $p = 100$, we find the canonical representative $(x', a') = (80, 10)$ which is a strictly periodic state. □

6.5 Analysis of F-FCSR in Fibonacci Representation

We recall the specification of two instances of the F-FCSR family of stream ciphers and present our analysis in Fibonacci representation.

6.5.1 Filtered FCSR's

In [6], the stream cipher F-FCSR-H with security level 80 bits was presented. It consists of a Galois FCSR of size $n = 160$ and with a memory of size $l = 82$. There are $k = 8$ fixed linear filter functions (applied on the intermediate state bits of the Galois FCSR) to produce 8 keystream bits in each iteration. A similar stream cipher F-FCSR-16 with security level 128 bits was presented, with $n = 256$, $l = 130$ and $k = 16$. According to [5, 7], we can expect the FCSR to be in a periodic state after the key/IV setup has completed. Our observations imply that both versions of F-FCSR can be equivalently described based on a Fibonacci FCSR instead of a Galois FCSR, but with a transformed filter. This transformation can be done in different ways, which gives different scenarios of potential attacks.

6.5.2 Transformation with Nonlinear Filter

If the initialization p of the Galois FCSR of F-FCSR is known, it can be mapped to an initial state of a Fibonacci FCSR such that both versions produce the same output. The advantage of the Fibonacci representation (from a cryptanalytic point of view) is that only one bit of the main state is modified per iteration, and 8 bits are sent to the keystream. However, this also requires a transformation of the linear filter to obtain the correct keystream: the linear filter of F-FCSR operates on the intermediate states of the

Galois FCSR. In order to compute the input of the filter function, we need to compute the values of certain Galois main register cells in each clock cycle:

Proposition 9. *The value x_i of the i-th cell in the main register of the Galois FCSR can be computed from the (strictly) periodic state (y, b) of the corresponding Fibonacci FCSR by*

$$x_i = M_i\left(b2^n - \sum_{k=0}^{n-1}\sum_{j=0}^{k} d_{j-1}y_{k-j}2^k\right) \mod q \mod 2 \ . \tag{6.3}$$

Proof. We first use Eq. 6.2 to compute the value of p that corresponds to the Fibonacci state (y, b) and then apply Prop. 8 to compute p_i. □

Every keystream bit is a linear combination of several bits given by Eq. 6.3. This results in a nonlinear system of equations in the unknowns (y, b).

6.5.3 Transformation with Linear Filter

Given some periodic initialization (x, a) of a maximum-length Galois FCSR, the sequence of a cell i corresponds to the output with a phase shift s_i, see Prop. 7. Consequently, the F-FCSR keystream can be produced by a linear filter applied on the FCSR output, where the required size of the FCSR output depends on the values of the involved s_i. The FCSR output can be produced with a Fibonacci FCSR (initialized by the state corresponding to p). Alternatively, one could think of an FCSR-combiner with linear filter, where the number of identical FCSR's corresponds to the number of filter taps, and where the initial states are not independent, but related according to Prop. 7.

6.5.4 Potential Attack with Linearization

We describe a trivial attack on a Fibonacci FCSR with $n = 160$, $l = 82$ and with $k = 8$ linear filters. Initially, there are 160 binary variables (ignoring the memory), and each updated bit is represented by a new variable (ignoring the details of the construction and assuming independence). Each iteration gives another 8 linear equations in these (initial and newly introduced) state variables. The main state can be recovered by solving the system of linear equations, if the number of equations is at least as large as the number of variables. This requires r iterations, where $8r \geq 160 + r$. Consequently, $r = 23$ iterations are sufficient, or 184 bits of keystream. Gaussian elimination of this system requires a computational complexity of about 184^3, which is 2^{23}. After recovering the main state, one can recover the memory state. If the FCSR is in a periodic state (which can be expected already after the initialization phase), then the effective size of the memory state reduces to 7 bits. Consequently, the memory can be guessed, or recovered by FCSR-synthesis, and the whole state can be recovered in about 2^{30} steps and with less than 200 bits of keystream. A similar attack is possible for any other construction of this type with $k > 1$.

However, the stream cipher F-FCSR with Fibonacci representation and with linear filters has initially a number of variables which corresponds to the maximum of involved

s_i. In Tab 6.1, we observe that the phase shifts s_i for the first filter of F-FCSR-H are distributed over a significant part of the period of the FCSR output sequence. Depending on the linear filters, the extracted bits to produce one keystream bit may involve the whole cycle of the FCSR output. On the other hand, the FCSR-combination generator requires many new variables. Consequently, we expect that the above scenario does not constitute a practical threat to neither of the two F-FCSR instances.

Table 6.1: List of phase shifts for the first filter of F-FCSR-H.

i	s_i
8	0x084D55C1E9BF6DABABE0BDA75592EE7F4C4DE6BA9
24	0x0F21C59484E27CB4F6D6D72A0F8141F0E2B4734C7
40	0x0C5FA773C15C6E3BE3E651BC3BB22FE735750A436
56	0x0F65F15C09715290BDB70A07C2520E6CB0A081382
80	0x0A778DF1C1F0E55E6B7B6EE796363498223BA75AC
96	0x1171C8C7A5E76A27EAB1E7C54D700A01112A5CE8A
104	0x060AA060A404D38A7307AFA25D3B9ED593CD05F15
112	0x061AD53ACAC8385A916026572D0FE291A53C93D3E
128	0x0FE422A9803989E98E16DB607440C1F40AF8BD82B
136	0x10076D047EA8E3B35E0C9C71B6CDAB88BEE0E6321

6.6 Summary

In this chapter we have given a simplified description of the sequences produced by a single cell of a Galois FCSR given the register's initial state is periodic. Additionally we have shown how to compute for a given state of a maximum length FCSR the unique equivalent periodic state. Based on these observations and the well-known correspondence between Fibonacci and Galois representations of FCSR's, we have proposed several new attack strategies. Currently, our analysis does not lead to an efficient attack, but may be useful as a starting point for further cryptanalytic research.

Chapter 7

Attacks on T-functions

In the previous chapter, we have seen that FCSR's may be suitable building blocks to replace linear driving devices in stream ciphers. Another suggestion is to use nonlinear T-functions. In this chapter, we analyze a class of stream ciphers based on T-functions. We use statistical and linear methods to mount very efficient distinguishing and key recovery attacks, and we observe a non-randomness of the initialization function of one eSTREAM proposal.

7.1 Introduction

In this chapter, we analyze several proposals of stream ciphers based on T-functions and exhibit substantial weaknesses in some of these constructions. The flaws are extended to dedicated attacks. First we analyze the statistical properties of the pure square mapping, which allows us to find an efficient distinguisher (with an expected 2^{32} data complexity) on TF-0 as well as on a previously unbroken multi-word mapping described in [87] and labeled here as TF-0M, both based on the squaring operation. TF-0M operates on a 256-bit state and the output sequence consists of the 32 most significant bits. Then, we cryptanalyze the TSC-family of stream ciphers [76], which operates on a 128-bit state and outputs 32 bits of the state using a filtering function. We find a very efficient distinguisher for TSC-1 with an expected 2^{25} data complexity, which can be used for key recovery; for TSC-2, we describe a different distinguishing attack with an expected 2^{34} data complexity. To confirm our theoretical results, the attacks have been implemented and run many times with success. Our attacks have a negligible error probability and a remarkably small time complexity. For eSTREAM Phase 2 candidate TSC-4, we identify a non-randomness in the initial state over the full eight-round initialization phase.

7.2 Cryptanalysis of Square Mappings

Klimov and Shamir have proposed different types of T-functions based on the squaring operation [85, 87]. After introducing the framework of this section, we focus on the pure square mapping and derive a hypothesis about their probability distribution. This dis-

tribution is used in order to distinguish the proposed mappings TF-0 and TF-0M with significant advantage.

Let us consider a scheme which consists of an update function L and an output function f. Let us further define the random variables X and X' over the set $\mathcal{X} = \{0,1\}^n$, with uniformly distributed X and with $X' = L(X)$. Equivalently, Z and Z' are random variables over $\mathcal{Z} = \{0,1\}^m$ with uniformly distributed Z and with $Z' = f(L(X))$. Given the distributions D_1, D_0 (corresponding to the distributions of Z, Z') and some uniform random or pseudo-random output respectively, we can perform a statistical test in order to assign the output to a distribution. According to Sect. 2.8, let $\Delta(D_0)$ be the imbalance of the distribution D_0. We are interested in the data complexity $N = d/\Delta(D_0)$ of the (optimal) distinguisher, corresponding to some designated overall error probability $p_e = \Phi(-\sqrt{d}/2)$. For small[1] word sizes n, the distribution D_0 can be determined by an exhaustive computation of $f(L(x))$ for all 2^n elements x, resulting in a precomputation time complexity of 2^n and a memory complexity (measured with the number of required memory cells) of 2^m. We assume that the test is performed online, hence we do not need additional memory in order to store the data. The online time complexity is identical to the data complexity. However, a precomputation of D_0 might be infeasible for large values of n (*e.g.* $n = 64$ bit). We perform some detailed analysis of D_0 for small word sizes n and establish an analytical hypothesis for the approximated distribution of Z', considering *only the most biased* elements. This significantly reduces the offline time and memory complexity, but might increase the online time and data complexity of the distinguisher, given some p_e. For small word sizes n, the hypothesis can be verified with the accurate distributions, and for large n, the quality of the hypothesis will be directly examined by the experimental data complexity of the distinguisher.

7.2.1 Distribution of the Pure Square Mapping

Let us define the pure square mapping (PSM) by $L(x) = x^2 \mod 2^n$ and $f(x) = x \gg (n-m)$, which are the m most significant bits of word x. Iteration produces some fixed points such as 0 or 1, hence L can not be considered as an update function for a real application. However, we will be able to reduce more complex single-cycle mappings to some modified square mappings and apply the results obtained in this section; in other words, we will consider the pure square mapping as an ideal case, resulting in distinguishers with minimal data complexity compared to modified square mappings.

We first mention that Klimov and Shamir [85] found an analytical expression for probabilities of single bits of the square mapping. Applying the notation $X' = L(X)$ for an uniformly distributed X, they found that $\Pr([X']_0 = 0) = \frac{1}{2}$, $\Pr([X']_1 = 0) = 1$ and $\Pr([X']_i = 0) = \frac{1}{2}(1 + 2^{-\frac{i}{2}})$ for $i > 1$. However, as we will have to deal with an additional carry bit later on (which would reduce this bias significantly), we are more interested in the distribution of words. We explain how to derive highly biased probability distributions for $X' = L(X)$ and $Z' = f(L(X))$. As shown in the next proposition, L is not a permutation,

[1] The term *small* is used with respect to current computational possibilities, *i.e.* $n \lesssim 40$ bit for personal computers nowadays.

resulting in an unbalanced distribution of X' (there are some predictable elements $L(x)$ with exceptionally large bias).

Proposition 10. *Consider the function $L : \{0,1\}^n \to \{0,1\}^n$ with $L(x) = x^2 \mod 2^n$. For successive elements $x \in \{0,\ldots,2^n-1\}$, the images $L(x)$ have a cyclic structure with cycle length 2^{n-2}. Hence L is neither injective nor surjective.*

Proof. As $x^2 - (2^{n-1}+x)^2 = 0 \mod 2^n$, we have two cycles of length 2^{n-1}, and as $(2^{n-2}+x)^2 - (2^{n-2}-x)^2 = 0 \mod 2^n$, both cycles have two mirrored sequences of length 2^{n-2}. Hence the output of successive numbers x has the shape $abc\ldots cbaabc\ldots cba$. □

Due to the specified output function in PSM, the bias is transferred to the distribution of Z'. For a truly random scheme, any element of the output occurs with probability $p_0 = 2^{-m}$. For the particular scheme PSM with $m = n/2$, we observed (for small word sizes n) that there exist 4 outcomes with biased probability $2 \cdot p_0$, 12 outcomes with biased probability $1.5 \cdot p_0$ and so on. This property appears to be independent of n, and we therefore can establish a hypothesis for the most biased elements (which are explicitly known). Let $\mathcal{Z}_i$ be the aggregate containing elements of constant biased probability p_i. The parameter s_i denotes the cardinality of $\mathcal{Z}_i$, and n_i denotes the minimal word size for a stable occurrence of p_i. The parameters n_i, s_i and p_i are summarized in Tab. 7.1. Then we have for $i = 0,\ldots,k$ (limited by the condition $n \geq n_k$)

$$\begin{array}{lcll} \mathcal{Z}_0 & = & \{2^{(n-n_0)/2} \cdot j^2; & j = 0,\ldots,s_0\} \\ \mathcal{Z}_i & = & \{2^{(n-n_i)/2} \cdot (1+8j); & j = 0,\ldots,s_i\} \\ \mathcal{Z}_\infty & = & \mathcal{Z} - \sum \mathcal{Z}_i\ . & \end{array} \tag{7.1}$$

The values in Tab. 7.1 are determined with empirical methods, however n_i and s_i are exact at least for word sizes within our computational possibilities. In the case of PSM, p_i is exact for $i = 0, 1$, but fluctuating for $i > 1$ so we have to take an average value. A further approximation is done with the remaining elements in $\mathcal{Z}_\infty$, which are assigned to a constant (standardized) probability. The number of aggregates k determines the accuracy of the approximation. However, k is constrained by the condition $n < n_k$, and as the values of p_i are only accurate for n_i up to about 40, we usually choose $k = 8$ for $n > 40$ bit. This corresponds to a memory complexity of 2^{17}. Regarding the complexities of a distinguisher, increasing the number of aggregates k is coupled with more time, more memory and less data.

Table 7.1: Parameters of the approximated distribution for the first 9 aggregates.

i	0	1	2	3	4	5	6	7	8
$p_i 2^m$	2.000	1.500	1.200	1.100	1.050	1.030	1.002	1.005	1.003
$n_i 2^{-2}$	2	3	4	5	6	7	8	9	10
$\log_2(s_i)$	2	3	5	7	9	11	13	15	17

7.2.2 Attacking the Single-Word Mapping TF-0

Let us now consider the running single-word proposal TF-0 with the update function $L(x) = x + (x^2 \vee C) \mod 2^n$ where $C = 5, 7 \mod 8$, and with the output function $f(x) = x \gg (n - m)$ as described in [85,88]. As the low-order bits are known to be weak, the authors of the scheme proposed $m = 1, 8, 16, 32$ for the standard word size $n = 64$ bit. Klimov and Shamir showed that L is an invertible T-function over an n-bit state x with a single cycle of length 2^n. The number of extracted bits m controls a tradeoff between security and efficiency of the scheme. We give some relationship to PSM with the next proposition.

Proposition 11. *Consider the scheme TF-0. If one requires $C < 2^{n-m}$, it is $f(L(x)) - f(x) = f(x^2) + \alpha \mod 2^m$ for $n - m > 2$ and for a carry bit $\alpha \in \{0, 1\}$.*

Proof. As $L(x) = y = x + (x^2 \vee C) \mod 2^n$, we conclude $y - x = x^2 \vee C \mod 2^n$ for $C < 2^{n-m}$. Hence, $f(y - x) = f(x^2 \vee C) \mod 2^m$ and $f(y - x) = f(x^2) \mod 2^m$ for $C < 2^{n-m}$. We finally have $f(y) - f(x) - \alpha = f(x^2) \mod 2^m$ for $C < 2^{n-m}$ and for some carry bit $\alpha \in \{0, 1\}$. □

Prop. 11 states that the difference of two consecutive outputs of TF-0 differs only by an additive carry bit $\alpha \in \{0, 1\}$ from the output of PSM. Therefore, we may accurately approximate the distribution of the random variable $f(L(X)) - f(X)$ by the distribution of PSM (*i.e.* we neglect the influence of the carry bit). We choose the standard parameters $C = 5$, $n = 64$ and $m = n/2$ and use 9 aggregates for the distribution D_0. This gives an imbalance of $\Delta(D_0) = 2^{-28}$, and for $p_e = 0.05$ the estimated data complexity of the distinguisher is 2^{32}. This could be verified with experiments, and is somewhat larger than the lower limit derived by extrapolation for the accurate probability distribution. If the scheme is used as a pseudo-random number generator in large computer simulations, the output may not be considered as random after 2^{32} iterations, although we have a single-cycle of 2^{64} states. This observation is consistent with the practice nowadays, not to use more data than $\sqrt{P}$ of a pseudo-random number generator (PRNG) with period P. However, we also examined modified output functions with a smaller number of extracted bits m. Experiments show that (independently of the word size n), decreasing m by one bit increases the data complexity by a factor of 2. We conclude that, in contradiction to previous assumptions, not only the lower bits of this T-function are weak, but also the higher bits. This is an intrinsic property of the scheme, which will have consequences for other square mappings and may have consequences for more complicated output functions.

We mention that state-recovery attacks on TF-0 have been described in [14,86]. Moreover, Mitra and Sarkar [99] described a time-memory tradeoff for the squaring problem, which may be applied to consecutive output differences of TF-0. The most efficient algorithms have a complexity of about 2^{16}.

7.2.3 Attacking the Multi-Word Mapping TF-0M

Several multi-word update functions proposed in [87] have been attacked with a time-memory tradeoff by Mitra and Sarkar [99]. We now present a distinguishing attack against

a multi-word proposal which has not been broken yet, and which we will refer as TF-0M. The update function L corresponds to Eq. 12 in [87], it is an invertible T-function over a $4n$-bit state $x = (x_0, x_1, x_2, x_3)$ with a single cycle of length 2^{4n}:

$$L : \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} \mapsto \begin{pmatrix} x_0 & + & (s_0^2 \vee C_0) & & \\ x_1 & + & (s_1^2 \vee C_1) & + & \kappa_0 \\ x_2 & + & (s_2^2 \vee C_2) & + & \kappa_1 \\ x_3 & + & (s_3^2 \vee C_3) & + & \kappa_2 \end{pmatrix} . \tag{7.2}$$

It is $s_0 = x_0$, $s_1 = s_0 \oplus x_1$, $s_2 = s_1 + x_2$, $s_3 = s_2 \oplus x_3$. The constants are satisfying $[C_i]_0 = 1$ for $i \in \{0, 1, 2, 3\}$, and $[C_3]_2 = 1$. All operations are carried out on n bit words and κ_i denotes the carry bit of x_i. The output function is $f(x) = x_3 \gg (n - m)$ with $m = n/2$. We choose the standard word size $n = 64$ bit. We observe that the multi-word update function of Eq. 7.2 consists of 4 approximatively independent and identically distributed random variables similar to the single-word update function of TF-0. We may concentrate only on the most significant variable x_3. The argument to be squared s_3 can be approximated as uniformly distributed, and therefore produces the same output as x^2. The carry bit modifies the output with a probability of 2^{-33}; this infrequent event will not have a significant influence to the distinguisher. Therefore, we do not have to modify the approximate distribution used for the distinguisher. Theoretical data complexity remains the same, and simulations result in an experimental data complexity of 2^{32} for a 256 bit state with 224 unknown bits. We have performed 20 experiments, observing no incorrect decision of our distinguisher. The data complexity is very close to the complexity for TF-0, confirming our assumption on the influence of κ and s. We emphasize the practical applicability of this result and the small number of required data, compared to the large number of unknown bits. As before, we also considered to extract less bits $m < n/2$. Again, we found that decreasing m by one bit increases the data complexity by a factor of 2. Hence reduction of m may still not prevent practical attacks.

7.3 Cryptanalysis of TSC-1

We start this section with a description of the recent proposal of stream cipher TSC-1 [76]. We find a very efficient distinguishing attack on TSC-1, which can be transformed in a state-recovery attack.

7.3.1 Description of the Scheme

The stream cipher TSC-1 consists of a state vector of 128 bits $x = (x_0, x_1, x_2, x_3)$, an update T-function L and an output function f. Bit-slice i of the state is defined by $[x]_i = ([x_0]_i, [x_1]_i, [x_2]_i, [x_3]_i)$. The update function consists of an odd 32-bit parameter $\alpha(x)$ and a single-cycle S-box S, mapping a 4 bit input to a 4 bit output. If $[\alpha]_i = 0$, then the mapping S^e is applied on bit-slice i of the state, otherwise the mapping S^o is applied. e (resp. o) is an even (resp. odd) number. This procedure is repeated for all 32

bit-slices in a single update period. With the satisfaction of these properties, L is a single-cycle T-function, hence the period of the cipher is 2^{128}. The odd parameter is defined by $\alpha = (p+C) \oplus p \oplus 2s$ with a constant C, $p = x_0 \wedge x_1 \wedge x_2 \wedge x_3$ and $s = x_0 + x_1 + x_2 + x_3$. Except for the lower few bits, each output bit of α is equal to 1 almost half of the time. Due to the properties of an odd parameter, one has $[\alpha]_0 = 1$, meaning that the least significant bit-slice is always mapped by S^o. Consequently, the bits from the least significant bit-slice of the state will be referred as *irregular bits*. In TSC-1, the powers of the S-box are $e = 2$ and $o = 1$, the constant used in the odd parameter is $C =$ `0x12488421`, the S-box (in standard notation) is defined by $S =$ (3,5,9,13,1,6,11,15,4,0,8,14,10,7,2,12) and the output function is

$$f(x) = (((x_0 \lll 9) + x_1) \lll 15) + ((x_2 \lll 7) + x_3) \ .$$

The output functions have a period of 2^{128}, however, three state variables in the output equation determine the remaining variable, hence the maximum security of the ciphers is 96 bit. Furthermore, there are some time-memory tradeoffs on TSC with large precomputation time complexities.

7.3.2 Description of the Attack

In this section, we present a linearization attack on TSC-1. Probabilistic linear relations in the update function (*i.e.* relations between state bits at different time instants) and in the output function (*i.e.* relations between state bits and output bits) are combined, in order to obtain relations between output bits at different time instants. Provided that the relations are biased, the output of TSC-1 can be distinguished from a random stream.

Let us first discuss a linear approximation of the T-function. We focus on a single bit $[x_j^t]_i$ and analyze the statistical effect of Δ iterations to this bit. Let Y_Δ be the indicator variable of the event $[x_j^t]_i = [x_j^{t+\Delta}]_i$, implying that a fixed bit is repeated after Δ iterations. After Δ iterations, bit-slice i (including the bit under observation) is mapped δ times by S, with $\Delta \leq \delta \leq 2\Delta$ (the mapping S is applied $2\Delta - \delta$ times, and the mapping S^2 is applied $\delta - \Delta$ times). Hence, in order to compute $\Pr(Y_\Delta = 1)$, we have to analyze the distribution of δ and the bit-flip probabilities of the mappings S^δ. Let us denote $b_\Delta(\delta)$ the probability that after Δ iterations, the S-box is applied δ times. For regular bit-slices, we reasonably assume equal probabilities for the application of S and S^2 (which is, however, a simplification for some lower bit-slices), and binomial distribution for b_Δ,

$$b_\Delta(\delta) = \binom{\Delta}{\delta - \Delta} \cdot \left(\frac{1}{2}\right)^\Delta \ . \tag{7.3}$$

For the irregular bit-slice, it is $b_\Delta(\delta) = 1$ for $\delta = \Delta$, and zero otherwise. In order to describe the effect of the mappings S^δ, let us analyze the S-box. We will denote w an uniform random number $0 \leq w \leq 15$, and i an index $0 \leq i \leq 31$. Let also X_δ be the indicator variable of the event $[w]_i = [S^\delta(w)]_i$ for any fixed bit position i. The S-box is designed such that the *bit-flip probability* for an application of S and S^2 is balanced. However, there is a huge bias of the bit-flip probability for some multiple applications of

S, namely for $\Pr(X_\delta = 1)$ with $\delta = 0 \mod 4$ (this observation is of course portable to the mapping S^2). We find $\Pr(X_4 = 1) = \Pr(X_{12} = 1) = 1/8$, $\Pr(X_8 = 1) = 3/4$ and of course $\Pr(X_{16} = 1) = 1$. These results are independent of bit-position i, other values of δ result in balanced probabilities. Finally, the bit-flip probability $P(Y_\Delta)$ of a single bit in the state for Δ iterations simply becomes the weighted sum

$$\Pr(Y_\Delta = 1) = \sum_{\delta=\Delta}^{2\Delta} \Pr(X_\delta = 1) \cdot b_\Delta(\delta) \ . \tag{7.4}$$

We find a maximal bias for $\Delta = 3$ with $\Pr(Y_3 = 1) = 0.3594$, and still a large bias for many other values of Δ. The predicted probabilities are in good agreements with experiments. In the case of irregular bits, Eq. 7.4 simply becomes $\Pr(Y_\Delta = 1) = \Pr(X_\Delta = 1)$ with a large bias for $\Delta = 0 \mod 4$. In the fictive case of a perfect single-cycle S-box (which, however, does not exist) with $\Pr(X_\delta = 1) = 1/2$ for $\delta \neq 16$ and $\Pr(X_{16} = 1) = 1$, Eq. 7.4 becomes $\Pr(Y_\Delta = 1) = (b_\Delta(16) + 1)/2$ for regular bits. A maximal bias is obtained for $\Delta = 11$, resulting in $\Pr(Y_{11} = 1) = 0.6128$.

Let us combine Eq. 7.4 with a simple linear approximation of the output function. The bias of Y_Δ strikes through the output function, such that the loops in the state are also present in the output. We consider a single bit $[z^t]_i$ of the output and analyze the statistical effect of Δ iterations to this bit. Let Z_Δ be the indicator variable of the event $[z^t]_i = [z^{t+\Delta}]_i$, implying that a fixed bit of the output is repeated after Δ iterations. We approximate the output function by $[z]_i = [x_0]_{i+8} \oplus [x_1]_{i+17} \oplus [x_2]_{i+25} \oplus [x_3]_i \oplus c$, for $i = 0, \ldots, 31$ (additions of indices are performed modulo 32) and a carry bit $c \in \{0, 1\}$. For bit-positions $i = 0, 7, 15, 24$, one irregular bit is involved in the linear approximation of $[z]_i$; consequently, these output bits are called irregular. Neglecting the carry bit and availing the fact that the output bits are composed of independent state bits, the probability $\Pr(Z_\Delta = 1)$ is approximated using Matsui's *Piling-up Lemma* [94]. For regular output bits, we obtain

$$\Pr(Z_\Delta = 1) = \frac{1}{2} + 2^3 \cdot \left(\Pr(Y_\Delta = 1) - \frac{1}{2} \right)^4 \ . \tag{7.5}$$

Notice that $\varepsilon = \Pr(Y_\Delta = 1) - \frac{1}{2}$ is the probability bias. In the case of irregular output bits, one of the four factors ε in Eq. 7.5 is substituted by $\varepsilon' = \Pr(X_\Delta = 1) - \frac{1}{2}$. Let us consider the case of $\Delta = 3$; it is $\Pr(Z_3 = 1) = 0.5031$ for regular output bits (and a balanced probability for irregular output bits). However, as we neglected the carry bit in this simple model, the above probability is considered as an upper limit. Notice that the carry is also biased and inclines towards absorbing itself. Experiments show that indeed, most of the regular output bits are biased for $\Delta = 3$. We emphasize that higher bits are affected equivalently to lower bits. Due to the integer addition, the exact bias depends on the bit-position. We find the maximum bias for bit-position $i = 1$ with $\Pr = 0.5003$. A similar result is obtained for $\Delta = 8$ and $i = 0$. This biased probability distribution D_0 is accessible to a cryptanalyst with known plaintext and may be used to distinguish the outcome of the cipher from a uniform random outcome. The imbalance

of the distribution is $\Delta(D_0) = 2^{-21}$, and for $p_e = 0.05$ the estimated data and online time complexity of the distinguisher is 2^{25} (4 MB of keystream); offline time complexity is negligible. We performed a number of experiments (taking all biased bits into account) and verified the predicted complexity. As described above, a variant of this attack even works without taking into account any specific property of the single-cycle S-box.

7.3.3 A State-Recovery Attack

The bias of Z_Δ can be transformed in a state-recovery attack by guess-and-determine. In a first step, we guess the least-significant bit-slice $[x^t]_0$, which may be iterated separately. The four corresponding bits are subtracted independently from appropriate output bits in order to construct a modified index variable. Considering Eq. 7.5, we expect the bias to significantly increase for a right guess, and we expect a balanced output for a false guess. After recovering $[x^t]_0$, we may continue with consecutive bit-slices. Considering all available equations, experiments showed that a single bit-slice may be accepted or rejected (with a reasonable probability of error) using 2^{25} iterations. Repeating this for all 2^4 values of a single bit-slice, and for all 2^5 bit-slices, we obtain an overall complexity of about 2^{34}. A similar result has also been obtained by Peyrin and Muller [101].

7.4 Cryptanalysis of TSC-2

In this section, we describe the proposal TSC-2 and find an efficient distinguishing attack.

7.4.1 Description of the Scheme

The stream cipher TSC-2 [76] is defined as TSC-1, with the following differences: In TSC-2, one has $e = 0$ (hence, the identical mapping is used), $o = 1$ and $C =$ `0x00000001`. The S-box is defined by $S =$ (5,2,11,12,13,4,3,14,15,8,1,6,7,10,9,0) and the output function is

$$f(x) = (((x_0 \lll 11) + x_1) \lll 14) + (((x_0 \lll 13) + x_2) \lll 22) + ((x_0 \lll 12) + x_3) \ .$$

7.4.2 Description of the Attack

The 32 bits of α determine the update of the 128 bits of the state. Hence we may wait for appropriate values of α in order to initiate some attacks. In TSC-2, an interesting case is the minimal-weight parameter $\alpha = 1$, for which only the least significant bit-slice is modified and two similar successive outputs may be detected. The *detector* is an algorithm which takes as input the keystream z and gives out 1 if $\alpha = 1$, and 0 otherwise. The detector can make two types of errors: it can either output 1 when $\alpha \neq 1$ (false alarm) or 0 when $\alpha = 1$ (non-detection). The error probabilities are denoted by p_α and p_β, respectively.

The complete set of states $\mathcal{U}$ resulting in $\alpha(x^t) = 1$ is given with the conditions $\sum_{i=0}^{3} x_i^t \in \{\texttt{0x00000000}, \texttt{0x80000000}\}$ and $[x^t]_0 \in \{\texttt{0x0}, \texttt{0x3}, \texttt{0x5}, \texttt{0x6}, \texttt{0x9}, \texttt{0xA}, \texttt{0xC}\}$, where $[x^t]_i$ denotes bit-slice i of the state matrix. In the following, let us assume that such a state occurs at time $t = 0$. Hence we have $\alpha^0 = 1$, and only the least significant bit-slice of the state is changed by the mapping $L: \ x^0 \to x^1$; consequently, we suppose that the subsequent outputs z^0 and z^1 have low distance. Let us analyze the exemplary integer modular difference $z^0 - z^1$ for $x \in \mathcal{U}$ with $[x^0]_0 = \texttt{0x5}$; we find that $[x^1]_0 = \texttt{0x4}$ and $[x^0]_i = [x^1]_i$ for $i \neq 0$. The output function produces $z^0 = z^1 + (\texttt{1} \lll 25) + (\texttt{1} \lll 3) + (\texttt{1} \lll 12)$ and hence $z^0 - z^1 = \texttt{0x02001008}$. In fact, we find that $z^0 - z^1 = \texttt{const}$ for any $x \in \mathcal{U}$, where the constant `const` depends only on the least-significant bit-slice $[x^0]_0$ in most of the cases, see Tab. 7.2. For less than 1% of the states in $\mathcal{U}$, the integer modular difference is not constant because an addition in the output function may cause a carry bit, which propagates from the msb to the lsb due to the cyclic shift.

Table 7.2: List of output differences for $\alpha = 1$, some of which will be applied in the attack.

$[x^0]_0$	$[x^1]_0$	$z^0 - z^1$
0x0	0x5	0xFDBFEFF8
0x3	0xC	0x01C05007
0x5	0x4	0x02001008
0x6	0x3	0xFE3FEFF8
0x9	0x8	0x02001008
0xA	0x1	0xFE002FF9
0xC	0x7	0xFDFFAFF9

Detection of single constants only would result in a huge amount of false alarms. However, examining Tab. 7.2, we find a path for the iteration of $[x^0]_0$ with $\texttt{0x6} \to \texttt{0x3} \to \texttt{0xC}$ which is closed in $\mathcal{U}$, meaning that $\alpha^0 = \alpha^1 = \alpha^2 = 1$. Therefore, we may restrict the detector to detect only a subset of states $\mathcal{V} \subset \mathcal{U}$, where $\mathcal{V}$ is defined by the conditions $\sum_{i=0}^{3} x_i^t \in \{\texttt{0x00000000}, \texttt{0x80000000}\}$ and $[x^t]_0 \in \{\texttt{0x6}, \texttt{0x3}\}$. The detector takes three successive outputs, computes two differences of consecutive outputs and compares them with the fixed values; if there is a match of both, the detector returns 1, and 0 otherwise. The probability of $x \in \mathcal{V}$ is 2^{-33}, and a false detection due to random outputs[2] occurs with probability 2^{-64}. As the differences are constant almost all the time, the error p_β (which would increase the running time of the detector) is negligible, too. The time and data complexity is around 2^{33} (no precomputation and negligible memory).

The detector can be transformed in a distinguisher by feeding the detector with a fixed amount of data N. If the detector always returns 0, then the distinguisher returns 0 (random stream); if the detector returns 1 at least once, then the distinguisher returns 1 (keystream produced by TSC-2). The probability of false positives can be neglected, and the probability of false negatives is $p_\beta = (1 - 2^{-33})^n$. For $p_\beta = 0.05$, we obtain a data complexity of about $N = 2^{34}$. With a successful detection of $\alpha(x^t) = 1$, we obtain the

[2] In order to increase the set $\mathcal{V}$, we do not make use of the connection of the whole path.

information $\sum_{i=0}^{3} x_i^t \in \{\texttt{0x00000000}, \texttt{0x80000000}\}$, as well as the value of bit-slice $[x^t]_0$ and the output equation $f(x^t) = z^t$. This information may be used for a state-recovery attack with a complexity smaller than 2^{96}. However, TSC-2 appears to be seriously injured with our efficient distinguishing attack, and we did not study the state-recovery attack in more detail.

7.5 Non-randomness of TSC-4

The attack on TSC-1 exploits a bit-flip bias for multiple applications of the state update function L. In [102], a similar attack was applied for eSTREAM Phase 2 candidate TSC-3. For the tweaked version TSC-4, this bias still exists for regular updates, but the strong filter function f prevents from an attack. In this section, we disregard the details of the filter function and investigate the statistical properties of multiple warm-up updates of TSC-4: While the regular updates have some guaranteed properties, the warm-up updates use additional *ad hoc* operations that are designed to accelerate diffusion. Notice that our analysis is embedded in a more general context: we actually consider the initialization function F of TSC-4 and try to detect some non-random behavior in a set of outputs (*i.e.* in the TSC-4 initial states) that are produced by a set of well-chosen inputs (*i.e.* in the IV's).

7.5.1 Description of the Scheme

The stream cipher TSC-4 is specified in [100]. It consists of two states x and y of 4×32 bits each, denoted $x = (x_0, x_1, x_2, x_3)^T$ and $y = (y_0, y_1, y_2, y_3)^T$. We first describe the regular update function L, which updates the two states x and y independently by single-cycle T-functions. In the case of state x, a 32-bit parameter α_x is computed as a function of x. It is defined by $\alpha_x = p \oplus (p + c_x) \oplus s$ with $p = x_0 \wedge x_1 \wedge x_2 \wedge x_3$ and $s = (x_0 + x_1 + x_2 + x_3) \ll 1$ and constant $c_x = \texttt{0x51291089}$. If $[\alpha_x]_i = 1$, then the fixed S-box $S = (9,2,11,15,3,0,14,4,10,13,12,5,6,8,7,1)$ is applied to bit-slice i of x, and if $[\alpha_x]_i = 0$, then the S-box S^6 is applied to bit-slice i of x (for all $i = 0, \ldots, 31$). The state y is similarly updated where parameter α_y has constant $c_y = \texttt{0x12910895}$. Notice that the least significant bit-slice is always mapped by S. The output function f produces a keystream byte z by combining some bytes of both states (using integer addition, XOR, shift and rotation), see [100] for more details.

Let us consider the initialization function of TSC-4. To start, the internal state of 256 bits is loaded with the secret key $K = (k_0, \ldots, k_9)$ and the initialization vector $V = (v_0, \ldots, v_9)$ each of 10×8 bits (a single 32-bit word is denoted as a concatenation of four 8-bit words).

$$x = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} k_3 & k_2 & k_1 & k_0 \\ k_7 & k_6 & k_5 & k_4 \\ v_3 & v_2 & v_1 & v_0 \\ v_7 & v_6 & v_5 & v_4 \end{pmatrix}, \quad y = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} v_1 & v_0 & v_9 & v_8 \\ v_5 & v_4 & v_3 & v_2 \\ k_1 & k_0 & k_9 & k_8 \\ k_5 & k_4 & k_3 & k_2 \end{pmatrix}$$

Table 7.3: Average imbalance $\Delta(D_0)$ in the statistical model for $r = 6, \ldots, 12$ rounds, and for different bit-slices.

r	lsb in x	lsb in y	non-lsb in x	non-lsb in y
6	$2^{-3.1}$	$2^{-3.9}$	$2^{-9.7}$	$2^{-11.8}$
8	$2^{-6.0}$	$2^{-8.2}$	$2^{-13.8}$	$2^{-17.1}$
10	$2^{-8.9}$	$2^{-13.0}$	$2^{-18.2}$	$2^{-22.6}$
12	$2^{-11.4}$	$2^{-15.9}$	$2^{-23.4}$	$2^{-28.1}$

A single round of the initialization function (denoted as a *warm-up* update) consists of a regular update and some additional operations: A byte $z = f(x)$ is produced, x_1 and y_0 are rotated to the left by eight positions, and then byte z is `XOR`ed to the 8 least significant bits of x_1 and y_0. The specifications of TSC-4 propose $r = 8$ rounds.

7.5.2 Statistical Model of Initialization

We investigate the statistical properties of the initialization process. In our statistical model, we assume that the parameter α (with exception of the lsb) and the feedback z are uniformly randomly distributed. For a single bit-slice i (not the least significant one) in the state x, our assumptions imply for each round:

1. Bit-slice i is mapped uniformly randomly by S or by S^6.
2. After application of the S-box, bit 1 of bit-slice i is chosen uniformly randomly.

With a fixed input $w \in \{0, \ldots, 15\}$, these two steps are repeated for r rounds, so we can analyze the distribution of the output $v \in \{0, \ldots, 15\}$. Within this model, the distribution can be computed exactly in 2^{2r} steps. The other cases (*i.e.* the least-significant bit-slice and the state y) are treated similarly. The imbalance is measured by $\Delta(D_0)$, where D_0 is the distribution with probabilities $\Pr(v)$ for an output v (given some fixed parameters). In Tab. 7.3, the imbalance is shown for different parameters. To simplify the presentation we compute imbalance for all inputs w and show the *average* values only. As expected, the average imbalance is decreasing with the number of rounds r. In the case of the least-significant bit-slice in the state x, it is reduced by a factor of about 2.6 with each additional round. Interestingly, the position of the random bit (see Step 2) has a notable influence on the distribution and diffusion is better for state y. And, as expected, diffusion is better for bit-slices which are not on the least-significant position (intuitively a combination of S and S^6 results in larger diffusion than using S only).

7.5.3 Experimental Results

Now we attempt to detect the bias of the previous subsection in the genuine initialization function $F(K, V)$ of TSC-4. We need N different inputs (K, V) where the value of a fixed bit-slice i is the same for all inputs. Each bit-slice consists of two key bits and two IV

Table 7.4: Average χ^2 statistic in the experiment for $r = 8$ rounds and a varying number of samples.

	All Keys	
N	average χ^2 value	% values > 80
2^{10}	40	3
2^{12}	119	67
2^{14}	421	100

bits. Consequently, bit-slice i is the same for all inputs, if the key is fixed (and unknown), and if the IV bits of bit-slice i are fixed (though the other IV bits can be varied). The N outputs can then be used to evaluate the distribution of bit-slice i. Provided that the assumptions on the model of the previous section are valid, bit-slice $i = 0$ of the state x is expected to have maximum bias. Here is an example for $r = 8$ rounds.

Example 20. Take N different inputs (K, V) where $V = (v_0, \ldots, v_9)$. The key is fixed, IV bytes $v_0, v_1 \ldots, v_7$ are zero, and v_8, v_9 increments from 0 to $N - 1$. Compute $N = 2^{10}$ outputs after $r = 8$ rounds of $F(K, V)$ and evaluate the imbalance $\Delta(D_0)$ of the distribution of the least-significant bit-slice in the initial state x. In 100 experiments using random keys, we find an imbalance of $\Delta(D_0) = 2^{-5.2}$. The results for the corresponding χ^2 statistics are listed Tab. 7.4. □

The measured bias is in good agreement with the model of Section 7.5.2, which predicts an average imbalance of $\Delta(D_0) = 2^{-6.0}$ in this setup[3]. Of course, the initial state cannot be accessed by an attacker, so the χ^2 test has perhaps a certificational character. However, the setup of Ex. 20 does not require any key bit to be known, and the number of samples N is very small. Consequently, this non-randomness may be a basis for future attacks that includes analysis of the filter function f. The non-randomness is not limited to the least significant bit-slice. A notable example is $i = 8$ (and with other parameters as in Ex. 20). This is a consequence of the specific setup in Ex. 20 where bit-slices $i = 8, 9 \ldots$ of x after the first round are the same for all N states and so the effective number of rounds is only $r-1$ (in addition, the biased bit 1 of bit-slice 0 is rotated into bit-slice 8). The experiment with $i = 8$ was carried out for a varying number of rounds, see Fig. 7.1. The imbalance $\Delta(D_0)$ in terms of r can be approximated by an exponential decay and in one round it is reduced by a factor of about 2.5. By extrapolation, we expect that about $r = 35$ rounds would be necessary to obtain an imbalance of $\Delta(D_0) = 2^{-40}$. In an extended experiment one could also measure the effectiveness of the combined initialization function F and update function L^t. For example, with $r = 8$, $t = 50$, we observed an average value of $\Delta(D_0) = 2^{-14}$ when using the same setup as previously. However we did not observe a bias in the keystream.

[3]Notice that two input bits of bit-slice $i = 0$ are always zero in the setup of Ex. 20. This has a small influence on the modeled bias in Tab. 7.3.

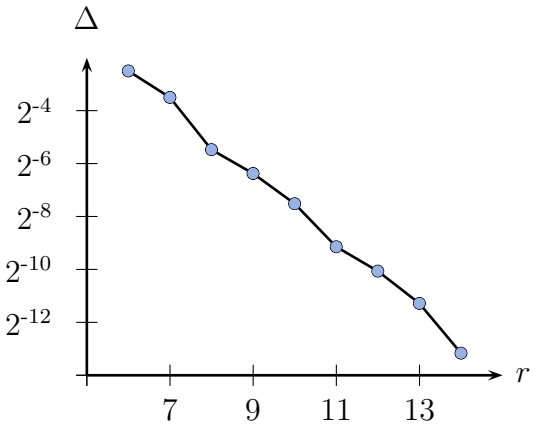

Figure 7.1: The average imbalance $\Delta(D_0)$ for $r = 6, \ldots, 14$ rounds.

7.6 Summary

In this chapter, we examined some specific proposals of stream ciphers based on T-functions. Two proposals by Klimov and Shamir are based on the squaring operation, namely a single word T-function as well as a previously unbroken multi-word T-function with a 256-bit state, both revealing some part of the state. It turned out that the integer differences of consecutive outputs have significant statistical deviation even in the high-order bits. Based on that deviation, we described efficient distinguishing attacks with a 2^{32} data complexity. We conclude that the squaring operation has some undesirable properties when used in the design of T-functions and possibly in other cryptographic primitives. The two proposals by Hong *et al.* have a 128-bit state, which are controlled by a 32-bit parameter and tiny S-boxes. The output function uses some integer additions and rotations. For one of the proposals, we found small loops in the state and in the output produced by the S-box, resulting in a distinguishing attack of complexity 2^{25}. For the other proposal, we wait for an appropriate value of the parameter, which produces some detectable structure in the output. This results in a distinguisher of complexity 2^{34}. We conclude that the small size of the parameter (and potentially also the tiny S-boxes) may be critical, and that the integer additions and rotations in the output functions have a very limited randomizing effect. In the case of TSC-4, we considered the way the key and the initialization information is used. The initial cipher state is derived using eight applications of a warm-up function. Non-randomness over all eight iterations can be detected in the initial state with about 1000 inputs. Each additional round increases the data requirements by a factor of about 2.5 and this non-randomness requires the attacker to choose IV bits only.

Chapter 8

Attacks on Salsa20 and Related Primitives

The stream cipher Salsa20 is currently one of the most promising software oriented eSTREAM candidates. It is a unique design with a trivial update function and a complex, round-based output function, similar to block ciphers or hash functions. The only operations used are integer addition, exclusive-or and rotation, hence it is a suitable aim for differential attacks. In this chapter, we investigate Salsa20 and related primitives such as ChaCha and the compression function Rumba.

8.1 Introduction

Salsa20 [17] is a stream cipher introduced by Bernstein in 2005 as a candidate in the eSTREAM competition, and has been selected for the third phase. Bernstein also submitted to public evaluation the 8 and 12 round variants Salsa20/8 and Salsa20/12 [18], though these are not formal eSTREAM candidates. More recently, he suggested a modification of the core function aiming at bringing faster diffusion without slowing down encryption, calling the variant ChaCha [21]. The compression function Rumba [20] was later presented in the context of a study of generalized birthday attacks [122] applied to incremental hashing [13], as the component of a hypothetical iterated hashing scheme. Rumba maps a 1536-bit value to a 512-bit (intermediate) digest, and Bernstein only conjectures collision resistance for this function, letting a further convenient operating mode provide extra security properties as pseudo-randomness.

First, we present a framework for finding high-probability differential trails for reduced-round Salsa20 (throughout we assume that the IV's can be chosen). This results in a key recovery attack on Salsa20/6, and a related-key attack on Salsa20/7. Inspired from correlation attacks, and from the notion of neutral bit, as introduced by Biham and Chen [22], we present then a novel method for probabilistic detection of the output-difference. More precisely, we first use an empirical measure of the correlation between certain key bits of the state and the bias observed after working a few rounds backward, in order to split key bits into two subsets: the extremely relevant key bits to be subjected to an exhaustive search and filtered by observations of a biased output-difference value, and the less significant key bits ultimately determined by exhaustive search. We present

the first key-recovery attack for Salsa20/8, and improve a previous attack on seven rounds by a factor 2^{38}. The method can also be applied to break 7 rounds of ChaCha. In a second part, we show collision and preimage attacks for derived versions of Rumba, and present a differential analysis of the original version using methods of linearization and neutral bits: our main result is a collision-search algorithm for 3-round Rumba running in about 2^{79} steps (compared to 2^{256} with a birthday attack). We also give examples of near-collision over three and four rounds.

8.2 Description of Salsa20

The stream cipher Salsa20 [17] works with 32-bit words and takes as input a key $K = (k_0, k_1, \ldots, k_7)$ of $n = 256$ bits, a nonce $V = (v_0, v_1)$ of 64 bits and a counter $T = (t_0, t_1)$ of 64 bits to produce a 512-bit block of the keystream. The counter is initialized by zero, and incremented after each application of the encryption function. At each application, Salsa20 acts on the following 4×4 matrix of 32-bit words:

$$x = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} = \begin{pmatrix} c_0 & k_0 & k_1 & k_2 \\ k_3 & c_1 & v_0 & v_1 \\ t_0 & t_1 & c_2 & k_4 \\ k_5 & k_6 & k_7 & c_3 \end{pmatrix} . \tag{8.1}$$

The constants are $c_0 =$ `0x61707865`, $c_1 =$ `0x3320646E`, $c_2 =$ `0x79622D32` and $c_3 =$ `0x6B206574`. There is also a mode for a 128-bit key K', where the 256 key bits in the matrix are filled with $K = K' \| K'$. If not mentioned otherwise, we focus on the 256-bit version. Let x^r denote the state after r rounds of Salsa20 (where the superscript 0 is omitted). Then, the keystream block z is produced using the following rule

$$z = x + x^{20} , \tag{8.2}$$

using wordwise addition modulo 2^{32}. The round function is based on the following nonlinear operation, which transforms a vector (x_0, x_1, x_2, x_3) to (y_0, y_1, y_2, y_3) by sequentially computing

$$\begin{aligned} y_1 &= x_1 \oplus ((x_3 + x_0) \lll 7) \\ y_2 &= x_2 \oplus ((x_0 + y_1) \lll 9) \\ y_3 &= x_3 \oplus ((y_1 + y_2) \lll 13) \\ y_0 &= x_0 \oplus ((y_2 + y_3) \lll 18) . \end{aligned} \tag{8.3}$$

This operation is called the quarterround function, see Fig. 8.1. In odd numbers of rounds (which are called columnrounds in the original specification of Salsa20), the nonlinear operation is applied to the columns (x_0, x_4, x_8, x_{12}), (x_5, x_9, x_{13}, x_1), $(x_{10}, x_{14}, x_2, x_6)$, $(x_{15}, x_3, x_7, x_{11})$. In in even numbers of rounds (which are also called the rowrounds), the nonlinear operation is applied to the rows (x_0, x_1, x_2, x_3), (x_5, x_6, x_7, x_4), $(x_{10}, x_{11}, x_8, x_9)$, $(x_{15}, x_{12}, x_{13}, x_{14})$. At each application 512 bits of keystream are generated by using the entirety of the final state as the keystream. We write Salsa20/R for R-round variants, *i.e.*

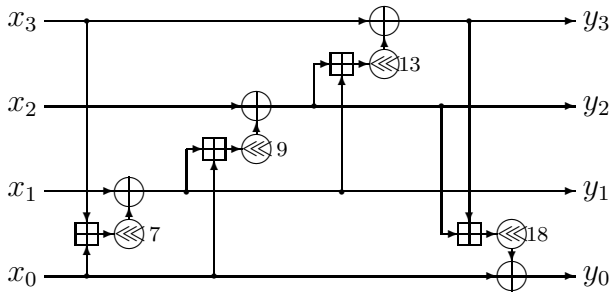

Figure 8.1: The quarterround function of Salsa20.

with $z = x + x^R$. Note that the r-round inverse function is defined differently whether it inverts after an odd or and even number of rounds.

8.3 Key-Recovery Attack on Salsa20/6

In this section we will demonstrate a non-random behavior which is detectable over six rounds of Salsa20. To start, we illustrate our approach by building on the earlier work of Crowley [47] and we describe a framework that allows a more sophisticated analysis to take place. This is achieved in two steps. First, we identify interesting differential effects in a simplified version of Salsa20. Second, we identify key and IV choices (where the IV denotes then nonce V and counter T here) that allow us to ensure that the behavior of the genuine Salsa20 is reasonably well-approximated by the simplified version. This technique allows us to make a systematic research of possible input differences $\mathcal{ID}$'s and consequently to find $\mathcal{ID}$'s with optimal properties. As mentioned, our observations are differential in nature. We will work with two copies of the state where x is filled with the input (K, V, T) and a second state x' is initialized according to $x' = x \oplus \Delta$ where $\Delta = (\Delta_0, \ldots, \Delta_{15})$ is the $\mathcal{ID}$. Note that the specifications of Salsa20 require that any $\mathcal{ID}$ must be zero in the diagonal words Δ_0, Δ_5, Δ_{10}, and Δ_{15}. After r rounds of Salsa20 the output difference $\mathcal{OD}$ is given[1] by $\Delta^r = x^r \oplus (x')^r$.

8.3.1 A Linearized Version of Salsa20

In previous work, Crowley [47] identified a truncated differential over three rounds of Salsa20. Consider setting $\Delta_i = 0$ for $i \neq 9$ and $\Delta_9 =$ 0x80000000. Then the following truncated differential for the first three rounds holds with a theoretical probability 2^{-12}. In practice a variety of effects conspire to give an average probability of 2^{-9}.

[1]Note that due to the feedforward in Salsa20 that uses addition modulo 2^{32} this is not necessarily the same as the difference in the corresponding keystream.

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \texttt{0x80000000} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \xrightarrow[\text{col}]{\text{row}}{\text{col}} \begin{pmatrix} ? & ? & ? & \texttt{0x02002802} \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{pmatrix}$$

Given the behavior exhibited in $x_3^3 \oplus (x_3')^3$ it is tempting to look for some impact in the next round. Yet, it is not clear how to proceed in a methodical manner. To establish an appropriate framework for analysis, we introduce an alternative algorithm LinSalsa20. This is identical to Salsa20 except that all integer additions have been replaced by exclusive-or. The corresponding round functions are denoted LinColumnround and LinRowround. Assume that two initial states x and $x' = x \oplus \Delta$ are iterated by LinSalsa20. Then since LinSalsa20 is completely linear in GF(2), the difference $\Delta^r = x^r \oplus (x')^r$ coincides exactly with computing r iterations of Δ with LinSalsa20. This computation does not require knowledge of the key and we refer to a differential path generated by LinSalsa20 as a *linear differential*. It is straightforward to see that there are many (admissible) input differences for which the output of LinSalsa20 is trivially non-random.

Proposition 12. *Consider an input* $\Delta_i \in \{\texttt{0xFFFFFFFF}, \texttt{0x00000000}\}$ *for all words* $i = 0, \ldots, 15$. *Then, for any number of updates with LinSalsa20, one has* $\Delta_i^r \in \{\texttt{0xFFFFFFFF}, \texttt{0x00000000}\}$.

However we need to be more careful. While LinSalsa20 allows some straightforward analysis, the further the behavior of LinSalsa20 is from the true Salsa20, the less useful it will be. Since a differential of large Hamming weight is likely to induce carries and hence non-linear behavior to the genuine Salsa20, we will need a linear differential of low Hamming weight. Such a differential is intended to offer a reasonably good approximation to the same differential in genuine Salsa20. We will consider a linear differential to be of low weight if any computation involving active words in the difference only uses words of low Hamming weight ($\ll 16$). Let us consider Crowley's differential within this linear model.

Example 21. Consider an input difference with $\Delta_9 = \texttt{0x80000000}$ as the one non-zero word. The weight of differences for the first four rounds is as follows.

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \xrightarrow{\text{col}} \begin{pmatrix} 0 & 2 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \xrightarrow{\text{row}} \begin{pmatrix} 4 & 2 & 2 & 2 \\ 7 & 10 & 3 & 6 \\ 1 & 3 & 4 & 1 \\ 0 & 1 & 1 & 2 \end{pmatrix} \xrightarrow{\text{col}} \begin{pmatrix} 9 & 19 & 6 & 5 \\ 3 & 13 & 5 & 5 \\ 4 & 11 & 11 & 7 \\ 1 & 16 & 2 & 10 \end{pmatrix}$$

$$\xrightarrow{\text{row}} \begin{pmatrix} 13 & 13 & 14 & 10 \\ 13 & 13 & 13 & 19 \\ 16 & 18 & 19 & 11 \\ \boxed{11} & 17 & 20 & 15 \end{pmatrix}$$

The top line of this differential is as far as Crowley goes, but when using LinSalsa20 it appears we can go one round further. Indeed, one can identify a low-weight linear

differential for word x_{12}^4, among others. Note that x_{12} is a right-to-diagonal word (with wrap) and is updated first in round four; the 16 in x_{13}^3 has no effect on x_{12}^4. □

8.3.2 Non-randomness in Four Rounds of Salsa20

Consider the linear differential of Ex. 21 and set $\mathcal{ID}$ to be identical to that of [47]. By using LinSalsa20 we suspect a statistical imbalance in $\Delta_{12}^4 = x_{12}^4 \oplus (x'_{12})^4$. Given a set of N different pairs of (K, V, T) for a fixed key K, where each pair takes the same fixed $\mathcal{ID}$, the distribution of the output difference for the N pairs can be analyzed. However, we might consider a subset of the bits or even a single bit, and by examining each bit in Δ_{12}^4 one finds that bit 26 is heavily unbalanced[2]. This differential is denoted $([\Delta_{12}^4]_{26} \mid [\Delta_9]_{31})$, and the bias ε of the $\mathcal{OD}$ is defined by

$$\Pr_{V,T}([\Delta_{12}^4]_{26} = 1 \mid [\Delta_9]_{31}) = \frac{1}{2} + \varepsilon \ , \tag{8.4}$$

where the probability holds over all nonces and counters (note that our statistical model considers the counter to be random). Furthermore, considering key as a random variable, we denote the median value of of ε by $\varepsilon^\star$. Hence, for half of the keys this differential will have a bias of at least $\varepsilon^\star$. The imbalance can be detected using a optimal distinguisher or a χ^2 test, see Sect. 2.8.

The behavior of the differential heavily depends on the input. The presence or absence of carries, on which Salsa20 relies, depends on the actual values of the operands. Thus some inputs will dampen, and others amplify, the evolution of a differential. The imbalance in bit 26 is greater the closer Salsa20 is to LinSalsa20. Therefore to find optimal inputs we will need to consider which conditions allow the non-linear differential trail to be closely approximated by the linear differential. The only non-linear operation in Salsa20 is integer addition in the quarterround function, denoted $x_a + x_b$. Given a corresponding $\mathcal{ID}$ (Δ_a, Δ_b), the nonlinear $\mathcal{OD}$ corresponds to the XOR of $x_a + x_b$ and $(x_a \oplus \Delta_a) + (x_b \oplus \Delta_b)$. Thus, the nonlinear $\mathcal{OD}$ is identical to the linear $\mathcal{OD}$, if

$$(x_a + x_b) \oplus ((x_a \oplus \Delta_a) + (x_b \oplus \Delta_b)) = \Delta_a \oplus \Delta_b \ . \tag{8.5}$$

Each non-zero bit in Δ_a and Δ_b may cause integer addition to create or annihilate a sequence of carry bits. Hence we focus on low-weight trails to keep more control of such events. Note that a difference in the most significant bit is always linear. We will (indirectly) consider Eq. 8.5 to place conditions on the inputs so that a differential in Salsa20 follows a linear differential in LinSalsa20 for some steps before diverging. We refer to this as partially *linearizing* the differential. Such conditions might be on the nonce, on the counter (assuming that the nonce and the counter are user-controlled inputs), or on the key (thereby establishing classes of weak keys). A close inspection of the first round of the differential of Ex. 21 reveals that the first two additions, differentially speaking, act as XOR while the third does not. However, depending on how t_1 is incremented, we can

[2] In fact there are many unbalanced bits in the state of Salsa20 after four rounds.

Table 8.1: Non-randomness in four rounds of Salsa20.

	All keys and nonces		Weak key class	
N	av. χ^2 value	% values > 40	av. χ^2 value	% values > 40
2^{12}	33	20	51	34
2^{14}	123	41	192	46
2^{16}	315	46	656	68

establish conditions on the key to ensure that it does. Thus there are keys for which the imbalance in bit 26 is boosted. The key conditions for the weak key class are on k_0 and k_6. First set the following bits of k_0 to the values shown:

bit number:	0	1	20	21	22	23
value:	0	1	0	0	1	1

Next set bit 7 of k_6 equal to bit 7 of A where $c_1 =$ `0x3320646E` and $A = (((k_0 + c_1) \lll 7) + c_1) \lll 9$. Note that all these conditions are randomly satisfied with a probability of 2^{-7}. A more sophisticated set of conditions can be derived to linearize the entirety of the first round. However for clarity we restrict ourselves to the simpler case.

Example 22. Take N inputs (K, V, T) with randomly fixed key K and random (V, T). For each input, we use values of t_1 to generate an associate input with $\mathcal{ID}$ $\Delta_9 =$ `0x80000000` (and zero otherwise). Compute the $\mathcal{OD}$ after four rounds of Salsa20 and evaluate the bias of bit 26 of Δ_{12}^4. In 100 experiments using random keys and nonces, we find an average bias of $\varepsilon = 0.04$. In the case of the weak key class, we find $\varepsilon = 0.05$. The results for the corresponding χ^2 statistics are listed in Tab. 8.1. □

8.3.3 Non-randomness in Six Rounds of Salsa20

The results presented in Section 8.3.2 give statistical weaknesses, as measured by the bias of a single bit, over four rounds of Salsa20. The statistical anomaly can be detected two rounds later. We intercept the required keystream z and we guess the necessary key words to partially unwind the last two rounds of state update and recover word x_{12}^4. The five key words to guess are k_3, k_4, k_5, k_6, k_7. Thus, for a single guess of the relevant words of key, the backwards computation is carried out over two rounds for N pairs of output, where each output was generated using the chosen input difference. The χ^2 statistic of the target bit of the target word is evaluated, and a χ^2 test is applied. Our analysis tells us that a correct key guess will yield a significant χ^2 score. We assume that an incorrect key guess results in essentially random candidate values for the bit we test. Thus, a significantly large χ^2 value suggests that the key guess may be correct. The remaining key words can be searched exhaustively and the entire key guess verified against the keystream. If the χ^2 value for a key guess is not significant we move on to a new guess. Clearly, the scale of the imbalance in the target bit is important to the success of this method. The closer

Table 8.2: Demonstration of a key recovery attack on five rounds of Salsa20.

	All keys and nonce	Weak key class
N	% success rate	% success rate
2^{12}	20	28
2^{14}	29	41
2^{16}	44	54

Salsa20 behaves to LinSalsa20 then the greater the imbalance in the target bit, and the greater the χ^2 score we expect to observe. This helps an attacker in two ways:

1. If certain keys and IV's give a high χ^2 score, then a greater proportion of the keys from an identified set should be susceptible to attack.
2. Higher χ^2 scores permit less keystream or greater precision in an attack.

To begin to get a picture of how things might behave in practice, we have implemented a restricted version of this style of attack. In principle we could use the four round differential of Ex. 22 to attack six rounds of Salsa20. To keep the experiments tractable, however, we use the same differential to attack a restricted five-round version as a demonstration (*i.e.* we unwind one round only).

Example 23. We recover nine bits (bits 4 to 12) of k_3 under the assumption that k_5 has been correctly guessed. Over 100 random keys and N pairs, we give the success rate when assuming the correct key lies among the candidate values giving the three highest χ^2 values. We repeat the experiment for the weak key class identified in Ex. 22. For the weak key class we observe that the same proportion of keys can be recovered when using one quarter of the text, see Tab. 8.2. We recall that the weak keys only improve the differential propagation and that our attack is also working for other keys. □

8.3.4 Complexity Estimation

As demonstrated in Ex. 23, at least in principle, our observations can be used in the way we intend. In this section, the complexity of an optimal distinguisher is evaluated in more detail. The subkey of our attack has a size of m bits, so we have a set of 2^m sequences of N random variables with $2^m - 1$ of them verifying the hypothesis H_1 (with uniform distribution D_1), and a single one verifying the hypothesis H_0 (having distribution D_0 characterized by ε). The decision rule to accept H_i can lead to two types of errors, $p_\alpha = 2^{-c}$ for false alarms and p_β for non-detection. A complexity of $2^m N/(1-p_\beta)$ is needed to find a number of $2^m p_\alpha$ subkey candidates. Each subkey candidate is then checked for correctness together with the remaining $l = n - m$ key bits, requiring a complexity of $2^l 2^m p_\alpha = 2^{n-c}$ with $n = 256$. In practice, we choose an overall error probability $p_e = \Phi(-\sqrt{d}/2)$ (note that both p_α and p_β are bounded by $2p_e$) and set $N = d/\Delta(D_0)$ with the imbalance

$\Delta(D_0) = 4\varepsilon^2$ according to Sect. 2.8. The probability p_β can be ignored, and $p_\alpha \approx p_e$ should be chosen such that it minimizes $C_T = 2^m N + 2^{256-c}$. Note that the potential improvement from key ranking techniques is not considered here, see *e.g.* [81]. The data complexity of our attack is $C_D = N$, if the counter can be chosen arbitrarily. In the case of Salsa20/6 we have $m = 160$ and $\varepsilon = 0.04$, and the work effort for a key-recovery attack is estimated to be around 2^{176} operations using $N = 2^{16}$ pairs of keystream blocks sampled appropriately from the same keystream. However, since the entirety of the target word can be recovered for any single key guess, using a single bit to test a key will miss much of the information available. We will exploit this in Sect. 8.5 to attack more rounds of Salsa20.

8.4 Related-Key Attack on Salsa20/7

The linear model can also be used to find longer differentials. A well-chosen multi-bit input may cause smaller diffusion than a single-bit input; non-zero bits can be placed in positions where they are annihilated in the update process. To illustrate, we focus again on a single column where the weight of the input (starting with the diagonal element) is $(0, 2, 1, 1)$. With a fixed relative position of the non-zero bits in this input, one can obtain an output after the first linear quarterround of the form $(0, 1, 0, 0)$. The absolute position of the non-zero bits and the choice of column are free parameters and naturally lead to an identified sub-class of inputs. These all have the same properties in LinSalsa20.

Example 24. Consider an input difference with non-zero words $\Delta_2 =$ `0x00000100`, $\Delta_6 =$ `0x00001000`, and $\Delta_{14} =$ `0x80080000`.

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \end{pmatrix} \xrightarrow{\text{col}} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \xrightarrow{\text{row}} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 3 & 4 \end{pmatrix} \xrightarrow{\text{col}} \begin{pmatrix} 4 & 1 & 3 & 4 \\ 1 & 2 & 4 & 8 \\ 1 & 0 & 7 & 10 \\ 3 & 1 & 3 & 14 \end{pmatrix}$$

$$\xrightarrow{\text{row}} \begin{pmatrix} 13 & 1 & 6 & 7 \\ 11 & 14 & 5 & 7 \\ 7 & 4 & 14 & 5 \\ 14 & 21 & 18 & 17 \end{pmatrix} \xrightarrow{\text{col}} \begin{pmatrix} 13 & 16 & 17 & 17 \\ 6 & 16 & 19 & 23 \\ 14 & \boxed{13} & 18 & 15 \\ 18 & 16 & 15 & 15 \end{pmatrix}$$

One can identify a truncated low-weight linear differential for x_9^5 which is an out-of-diagonal word. Note that some words in the final array may have a lower Hamming weight, but their generation required computations using average-weighted words and so they are unlikely to be relevant to genuine Salsa20. □

The non-zero bits of this differential are located in column two. Word x_{14} is updated first by $y_{14} = x_{14} \oplus (x_{10} + x_6) \lll 7$. A second state $x'_i = x_i \oplus \Delta_i$ is updated in the same way and, according to Eq. 8.5, the difference of this first update will follow the linear

Table 8.3: Non-randomness in five rounds of Salsa20.

	All Keys and Nonces		Weak Nonce Class	
N	av. χ^2 value	% values > 40	av. χ^2 value	% values > 40
2^{20}	5	4	27	26
2^{22}	16	11	105	73
2^{24}	78	17	383	89

differential if the following equation holds:

$$(x_{10} + x_6) \oplus ((x_{10} \oplus \Delta_{10}) + (x_6 \oplus \Delta_6)) = \Delta_{10} \oplus \Delta_6 \ . \tag{8.6}$$

Notice that Δ_{10} is zero and that Δ_6 has a single non-zero bit in position 12. Further, $x_{10} = c_2$ and $x_6 = v_0$. Bits $12 \ldots 9$ of c_2 are defined as $(\ldots \texttt{0110} \ldots)_2$. Consequently, if bits $11 \ldots 9$ of v_0 are chosen as $(\ldots \texttt{000} \ldots)_2$, then no carry is produced from the right, and Eq. 8.5 is satisfied. Subsequently x_2 is updated and so provided the previous update followed the linear differential, the only non-zero bit in the difference will be in bit 31 and the linear trial will be followed. Updating x_6 is similar while updating x_{11} only involves zero differences. Thus we have identified conditions on three bits of v_0, part of the nonce, so that the first round of genuine Salsa20 with the $\mathcal{ID}$ of Ex. 24 follows the linear trail. In fact, the $\mathcal{ID}$ of Ex. 24 turns out to be optimal, *i.e.* it seems to have minimum weight after two rounds of Salsa20; bitwise rotations of $\mathcal{ID}$ reduce the number of msb's while shifting the difference to another column shifts the input-condition to a key word instead of v_0. Without input conditions on v_0, the first round would follow the linear trail with a probability of about $\Pr = 0.175$.

Example 25. Take N inputs (K, V, T) with randomly fixed key K and random (V, T). For each input, we use values of k_1, v_0, k_7 to generate an associate input with $\mathcal{ID}$ $\Delta_2 =$ 0x00000100, $\Delta_6 =$ 0x00001000, $\Delta_{14} =$ 0x80080000 (and zero otherwise). Compute the $\mathcal{OD}$ after five rounds of Salsa20 and evaluate the bias of bit 1 of Δ_9^5. In 100 experiments using random keys and nonces, we find an average bias of $\varepsilon = 0.001$. In the case of the weak nonce class, we find $\varepsilon = 0.002$. In Tab. 8.3 the results are listed for the corresponding χ^2 statistics. □

We can intercept the required keystream of Salsa20/7 and guess the necessary key words to partially unwind the last two rounds of state update to recover Δ_9^5. The six key words to guess are k_0, k_2, k_3, k_4, k_5, k_6. According to Sect. 8.3.4, Salsa20/7 might be broken in around 2^{218} operations using 2^{26} pairs of keystream blocks taken from two sets of keystream. However, for the five round imbalance we used non-zero differences in part of the key k_1 and k_7, so the attack is only valid under a related-key scenario. The practical validity of such an attack is debatable [16], so we merely observe that over seven of the 20 rounds in Salsa20, statistical imbalances can be detected.

8.5 Key-Recovery Attack on Salsa20/8

In the previous sections, we focused on sophisticated differentials for Salsa20 and used deterministic backwards computation to distinguish the right subkey. In this section, we introduce differential attacks based on a new technique called probabilistic neutral bits (PNB's). In order to apply it to Salsa20, we first identify suitable choices of truncated single-bit differentials, then describe a general framework for probabilistic backwards computation, and introduce the notion of PNB's, along with a method to find them. After this, we outline the overall attack, and state our results for Salsa20/7, Salsa20/8. Eventually, we discuss our attack scenarios and possibilities of improvements.

8.5.1 Probabilistic Backwards Computation

In the following, assume that the differential $([\Delta_p^r]_q \mid [\Delta_i^0]_j)$ of bias ε_d is fixed, and the corresponding outputs z and z' are observed for nonce V, counter T and key K. Having K, V and T, one can invert the operations in $z = x + x^R$ and $z' = x' + (x')^R$ in order to access to the r-round forward differential (with $r < R$) from the backward direction thanks to the relations $x^r = (z - x)^{r-R}$ and $(x')^r = (z' - x')^{r-R}$. More specifically, define $f(K, V, T, z, z')$ as the function which returns the q-th lsb of the word number p of the matrix $(z - x)^{r-R} \oplus (z' - x')^{r-R}$, hence $f(K, V, T, z, z') = [\Delta_p^r]_q$. Given enough output block pairs with the presumed difference in the input, one can verify the correctness of a guessed candidate K' for the key K by evaluating the bias of the function f. More precisely, we have $\Pr(f(K', V, T, z, z') = 1) = \frac{1}{2} + \varepsilon_d$ conditioned on $K' = K$, whereas for (almost all) $K' \neq K$ we expect f be unbiased *i.e.* $\Pr(f(K', V, T, z, z') = 1) = \frac{1}{2}$. The classical way of finding the correct key requires exhaustive search over all possible 2^{256} guesses K'. However, we can search only over a subkey of $m = 256 - l$ bits, provided that an approximation g of f which effectively depends on m key bits is available. More formally, let $\bar{K}$ correspond to the subkey of m bits of the key K and let f be correlated to g with bias ε_a *i.e.*:

$$\Pr_{V,T}(f(K, V, T, z, z') = g(\bar{K}, V, T, z, z')) = \frac{1}{2} + \varepsilon_a \ . \tag{8.7}$$

Note that deterministic backwards computation (*i.e.* $\bar{K} = K$ with $f = g$) according to Sect. 8.3 and Sect. 8.4 is a special case with $\varepsilon_a = 1$. Denote the bias of g by ε, *i.e.* $\Pr(g(\bar{K}, V, T, z, z') = 1) = \frac{1}{2} + \varepsilon$. Under some reasonable independency assumptions, the equality $\varepsilon = 2\varepsilon_d\varepsilon_a$ holds. Again, we denote $\varepsilon^\star$ the median bias over all keys (we verified in experiments that $\varepsilon^\star$ can be well estimated by the median of $2\varepsilon_d\varepsilon_a$). Here, one can verify the correctness of a guessed candidate $\bar{K}'$ for the subkey $\bar{K}$ by evaluating the bias of the function g based on the fact that we have $\Pr(g(\bar{K}', V, T, z, z') = 1) = \frac{1}{2} + \varepsilon$ for $\bar{K}' = \bar{K}$, whereas $\Pr(g(\bar{K}', V, T, z, z') = 1) = \frac{1}{2}$ for $\bar{K}' \neq \bar{K}$. This way we are facing an exhaustive search over 2^m subkey candidates opposed to the original 2^{256} key candidates which can potentially lead to a faster attack. We stress that the price which we pay is a higher data complexity.

8.5.2 Probabilistic Neutral Bits

Our new view of the problem, described in Sect. 8.5.1, demands efficient ways for finding suitable approximations $g(\bar{K}, W)$ of a given function $f(K, W)$ where W is a known parameter; in our case, it is $W = (V, T, z, z')$. In a probabilistic model one can consider W as a uniformly distributed random variable. Finding such approximations in general is an interesting open problem. In this section we introduce a generalized concept of neutral bits [22] called *probabilistic neutral bits* (PNB's). This will help us to find suitable approximations in the case that the Boolean function f does not properly mix its input bits. Generally speaking, PNB's allows us to divide the key bits into two groups: *significant key bits* (of size m) and *non-significant key bits* (which are the l PNB's). In order to identify these two sets we focus on the amount of influence which each bit of the key has on the output of f. Here is a formal definition of a suitable measure:

Definition 9. *The neutrality measure of the key bit k_i with respect to the function $f(K, W)$ is defined as γ_i, where $\Pr = \frac{1}{2} + \gamma_i$ is the probability (over all K and W) that complementing the key bit k_i does not change the output of $f(K, W)$.*

We use the following Alg. 5 to compute the neutrality measure of a single key bit of Salsa20.

Algorithm 5 Computation of the neutrality measure

Input: Number of rounds R and r, key bit index i.
Output: Determine the neutrality measure γ_i.

1: Choose the number of samples T and let ctr $= 0$.
2: **for** i from 1 to T **do**
3: Pick a random state x (with fixed constants) and apply the $\mathcal{ID}$ to get x'.
4: Compute $z = x + x^R$ and $z' = x' + (x')^R$.
5: Compute $(z - x)^{r-R}$ and $(z' - x')^{r-R}$ and observe the $\mathcal{OD}$.
6: Flip the i-th key bit in x and x'.
7: Compute $(z - x)^{r-R}$ and $(z' - x')^{r-R}$ and observe the $\mathcal{OD}$.
8: Increment ctr if the $\mathcal{OD}$'s are equal.
9: **end for**
10: Output $\gamma_i = \text{ctr}/T - 1/2$.

Singular cases of the neutrality measure are:

- $\gamma_i = 1/2$: $f(K, W)$ does not depend on i-th key bit (*i.e.* it is a neutral bit).
- $\gamma_i = 0$: $f(K, W)$ is stat. independent of the i-th key bit (*i.e.* it is a significant bit).
- $\gamma_i = -1/2$: $f(K, W)$ linearly depends on the i-th key bit.

In practice, we set a threshold γ and put all key bits with $\gamma_i \leq \gamma$ in the set of significant key bits. The less significant key bits we get, the faster the attack will be, provided that the bias ε_a defined in Eq. 8.7 remains non-negligible. Having found significant and non-significant key bits, we simply let $\bar{K}$ be the significant key bits and define $g(\bar{K}, W)$ as $f(K, W)$ with non-significant key bits being set to a fixed value (*e.g.* all zero). Note

that, contrary to the mutual interaction between neutral bits in [22], here we have directly combined several PNB's without altering their probabilistic quality. This can be justified as the bias ε_a smoothly decreases while we increase the threshold γ.

Remark 9. Tsunoo *et al.* [118] used nonlinear approximations of integer addition to identify the dependency of key bits, whereas the independent key bits—with respect to nonlinear approximation of some order—are fixed. This can be seen as a special case of our method.

Remark 10. It is reasonable to assume that a false subkey, which is close to the correct subkey, may introduce a non-negligible bias. In general, this results in an increased value of p_α. If many significant key bits have neutrality measure close to zero, then the increase is expected to be small, but the precise practical impact of this observation is unknown to us.

8.5.3 Overview of the Attack

We sketch the full attack described in the previous subsections. It is split up in a precomputation step (independent of the key) and in the effective attack, see Alg. 6 and Alg. 7. The cost of precomputation is negligible compared to the effective attack, and complexity of the effective attack is the same as in Sect. 8.3.4.

Algorithm 6 Precomputation of the attack

1: Choose an r-round differential with $\mathcal{ID}$ in the nonce or counter.
2: Choose a threshold γ.
3: Construct the function f defined in Sect. 8.5.1.
4: Empirically estimate the neutrality measure γ_i of each key bit for f.
5: Put all key bits with $\gamma_i \leq \gamma$ in the significant key bits set of size $m = 256 - l$.
6: Construct the function g using f by assigning a fixed value to the PNB's, see Sect. 8.5.1 and Sect. 8.5.2.
7: Estimate the median bias $\varepsilon^\star$ by empirically measuring the bias of g using many randomly chosen keys, see Sect. 8.5.1.
8: Estimate the data and time complexity of the attack, see Sect. 8.3.4.

Algorithm 7 Effective attack

Input: N pairs of keystream blocks produced with relevant $\mathcal{ID}$, a function g.
Output: Recover the secret key.

1: **for** each choice of the subkey of m bits **do**
2: Compute the bias of g using the N keystream block pairs.
3: If the optimal distinguisher legitimates the subkey candidate as a (possibly) correct one, perform an additional exhaustive search over the l PNB's in order to check the correctness of this filtered subkey and to find the PNB's.
4: **end for**
5: Output the recovered key.

8.5.4 Experimental Results

We used automatized search to identify optimal differentials for the reduced-round versions Salsa20/7 and Salsa20/8. This search is based on the following observation: The number l of PNB's for some fixed threshold γ mostly depends on the $\mathcal{OD}$, but not on the $\mathcal{ID}$. Consequently, for each of the 512 single-bit $\mathcal{OD}$'s, we can assign the $\mathcal{ID}$ with maximum bias ε_d, and estimate time complexity of the attack. Below we only present the differentials leading to the best attacks. The threshold γ is also an important parameter: Given a fixed differential, time complexity of the attack is minimal for some optimal value of γ. However, this optimum may be reached for quite small γ, such that l is large and $|\varepsilon_a^\star|$ small. We use at most 2^{24} random nonces and counters for each of the 2^{10} random keys, so we can only measure a bias of about $|\varepsilon_a^\star| > \text{const} \cdot 2^{-12}$ (where const ≈ 5 for a reasonable estimation error). In our experiments, the optimum is not reached with these computational possibilities (see *e.g.* Tab. 8.4), and we note that the described complexities may be improved by choosing a smaller γ.

Attack on 256-bit Salsa20/7. We use the differential $([\Delta_1^4]_{14} \mid [\Delta_7^0]_{31})$ with $|\varepsilon_d^\star| = 0.0655$. The $\mathcal{OD}$ is observed after working three rounds backward from a 7-round keystream block. To illustrate the role of the threshold γ, we present in Tab. 8.4 complexity estimates along with the number l of PNB's, the values of $|\varepsilon_d^\star|$ and $|\varepsilon^\star|$, and the optimal values of c for several threshold values. For $\gamma = 0.25$, the attack runs in time 2^{152} and data 2^{27}. The previous best attack in [118] required about 2^{190} trials and 2^{12} data.

Table 8.4: Different parameters for our attack on 256-bit Salsa20/7.

γ	l	$\lvert\varepsilon_a^\star\rvert$	$\lvert\varepsilon^\star\rvert$	c	C_T	C_D
0.50	39	0.500	0.0655	30	2^{230}	2^{13}
0.45	97	0.328	0.0430	86	2^{175}	2^{16}
0.40	103	0.241	0.0317	91	2^{170}	2^{17}
0.35	113	0.101	0.0133	99	2^{162}	2^{19}
0.30	124	0.025	0.0032	106	2^{156}	2^{24}
0.25	131	0.009	0.0011	109	2^{152}	2^{27}

Attack on 256-bit Salsa20/8. We use the differential $([\Delta_1^4]_{14} \mid [\Delta_7^0]_{31})$ with $|\varepsilon_d^\star| = 0.0655$. The $\mathcal{OD}$ is observed after working four rounds backward from an 8-round keystream block. For the threshold $\gamma = 0.06$ we find $l = 36$, $|\varepsilon_a^\star| = 0.0006$, and $|\varepsilon^\star| = 0.00008$. For $c = 8$, this results in time 2^{251} and data 2^{30}. The list of PNB's is {26, 27, 28, 29, 30, 31, 71, 72, 120, 121, 122, 148, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 210, 211, 212, 224, 225, 242, 243, 244, 245, 246, 247}. Note that our attack reaches the same success probability and supports an identical degree of parallelism as brute force. The previous attack in [118] claims 2^{255} trials with data 2^{10} for success probability 44%, but exhaustive search succeeds with probability 50% within the same number of trials,

with much less data and no additional computations. Therefore their attack does not constitute a break of Salsa20/8.

Attack on 128-bit Salsa20/7. Our attack can be adapted to the 128-bit version of Salsa20/7. With the differential $([\Delta_1^4]_{14} \mid [\Delta_7^0]_{31})$ and $\gamma = 0.2$, we find $l = 38$, $|\varepsilon_a^\star| = 0.023$, and $|\varepsilon^\star| = 0.0030$. For $c = 20$, this breaks Salsa20/7 within 2^{111} time and 2^{21} data. Our attack fails to break 128-bit Salsa20/8 because of the insufficient number of PNB's.

Discussion. In our attacks on reduced-round 256-bit Salsa20, we exploit 4-round differentials, then we attack the cipher by going three of four rounds backwards. We made intensive experiments in order to observe a bias after going five rounds backwards from the guess of a subkey, in order to attack Salsa20/9, but without success. Four is probably the maximal number of rounds one can invert from a partial key guess while still observing a non-negligible bias after inversion, and such that the overall cost improves from exhaustive key search. Can one hope to break further rounds by statistical cryptanalysis? We believe that it would require novel techniques and ideas, rather than the relatively simple XOR difference of 1-bit input and 1-bit output. For instance one might combine several biased $\mathcal{OD}$'s to reduce data requirements, but this requires almost equal subset of guessed bits; according to our experiments, this seems hard to achieve. Exploiting multibit differentials such as the single-bit $\mathcal{ID}$ in $[\Delta_7]_{26}$ and $\mathcal{OD}$ in $[\Delta_1^4]_0 \oplus [\Delta_2^4]_9$ with $\varepsilon_d = -0.30$ does not improve efficiency either. Note that an alternative approach to attack Salsa20/7 is to consider a 3-round biased differential, and observe it after going four rounds backwards. This is however much more expensive than exploiting directly 4-round differentials. For the variant with a 128-bit key, we can break up to seven Salsa20 rounds.

8.6 Key-Recovery Attack on ChaCha7

We present the stream cipher ChaCha and use the same method as for Salsa20/8 to attack up to 7 rounds.

8.6.1 Description of the Scheme

The stream cipher ChaCha [21] is similar to Salsa20 with the following three modifications:

1. The input words are placed differently in the initial matrix:

$$x = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix}. = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ t_0 & t_1 & v_0 & v_1 \end{pmatrix} . \tag{8.8}$$

2. The nonlinear operation of Round transforms a vector (x_0, x_1, x_2, x_3) to (y_0, y_1, y_2, y_3) by sequentially computing

$$\begin{array}{rclcrcl} b_0 & = & x_0 + x_1, & & b_3 & = & (x_3 \oplus b_0) \lll 16 \\ b_2 & = & x_2 + b_3, & & b_1 & = & (x_1 \oplus b_2) \lll 12 \\ y_0 & = & b_0 + b_1, & & y_3 & = & (b_3 \oplus y_0) \lll 8 \\ y_2 & = & b_2 + y_3, & & y_1 & = & (b_1 \oplus y_2) \lll 7\ . \end{array} \tag{8.9}$$

3. The round function is defined differently: in odd numbers of rounds, the nonlinear operation is applied to the columns (x_0, x_4, x_8, x_{12}), (x_1, x_5, x_9, x_{13}), $(x_2, x_6, x_{10}, x_{14})$, $(x_3, x_7, x_{11}, x_{15})$, and in in even numbers of rounds, the nonlinear operation is applied to the diagonals $(x_0, x_5, x_{10}, x_{15})$, $(x_1, x_6, x_{11}, x_{12})$, (x_2, x_7, x_8, x_{13}), (x_3, x_4, x_9, x_{14}).

As for Salsa20, the round function of ChaCha is trivially invertible. R-round variants are denoted ChaChaR. The core function of ChaCha suggests that *"the big advantage of ChaCha over Salsa20 is the diffusion, which at least at first glance looks considerably faster"* [19].

8.6.2 Experimental Results

ChaCha is expected to have faster diffusion than Salsa20. Our experiments argue in favor of this conjecture, since we found many biased differentials over 3 rounds, but none over 4 rounds. Such differentials of weight one in both $\mathcal{ID}$ and $\mathcal{OD}$ can easily be found by automatized search.

Attack on 256-bit ChaCha6. We use the differential $([\Delta^3_{11}]_0 \mid [\Delta^0_{13}]_{13})$ with $|\varepsilon^\star_d| = 0.013$. The $\mathcal{OD}$ is observed after working three rounds backward from an 6-round keystream block. For the threshold $\gamma = 0.3$ we find $l = 147$, $|\varepsilon^\star_a| = 0.009$, and $|\varepsilon^\star| = 0.00024$. For $c = 121$, this results in time 2^{140} and data 2^{31}.

Attack on 256-bit ChaCha7. We use again the differential $([\Delta^3_{11}]_0 \mid [\Delta^0_{13}]_{13})$ with $|\varepsilon^\star_d| = 0.013$. The $\mathcal{OD}$ is observed after working four rounds backward from an 7-round keystream block. For the threshold $\gamma = 0.25$ we find $l = 35$, $|\varepsilon^\star_a| = 0.012$, and $|\varepsilon^\star| = 0.00030$. For $c = 10$, this results in time 2^{248} and data 2^{27}. The list of PNB's is {3, 6, 15, 16, 31, 35, 67, 68, 71, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 103, 104, 127, 136, 191, 223, 224, 225, 248, 249, 250, 251, 252, 253, 254, 255}.

Attack on 128-bit ChaCha6. Our attack can be adapted to the 128-bit version of ChaCha6. With the differential $([\Delta^3_{11}]_0 \mid [\Delta^0_{13}]_{13})$ and $\gamma = 0.25$, we find $l = 51$, $|\varepsilon^\star_a| = 0.007$, and $|\varepsilon^\star| = 0.00018$. For $c = 25$, this breaks ChaCha6 within 2^{107} time and 2^{30} data. Our attack fails to break 128-bit ChaCha7.

Discussion. In our attacks on reduced-round ChaCha, we exploit 3-round differentials, then we attack the cipher by going three of four rounds backwards. For five rounds, we did not observe a bias anymore. Unlike Salsa20, our exhaustive search showed no bias in 4-round ChaCha, be it with one, two, or three target output bits. This argues in favor of the faster diffusion of ChaCha. But surprisingly, when comparing the attacks on Salsa20/8 and ChaCha7, results suggest that after four rounds backwards, key bits are more correlated with the target difference in ChaCha than in Salsa20. Nevertheless, ChaCha looks more trustful on the overall, since we could break up to seven ChaCha rounds against eight for Salsa20. For the variant with a 128-bit key, we can break up to six ChaCha rounds.

8.7 Analysis of Rumba

In this section, we describe our results for the recently proposed compression function Rumba. Our goal is to efficiently find colliding pairs for reduced rounds of Rumba, *i.e.* input pairs (M, M') such that $\mathsf{RumbaR}(M) \oplus \mathsf{RumbaR}(M') = 0$. Note that, compared to our attacks on Salsa20 (where a single biased bit could be exploited in an attack), a collision attack targets all 512 bits (or most of the 512 bits in the scenario of near-collision attacks).

8.7.1 Description of the Scheme

Rumba is a *compression function* built on Salsa20, mapping a 1536-bit message to a 512-bit value. The input M is parsed as four 384-bit chunks $m_0,\ldots,m_3$, and the output value is

$$\begin{aligned}\mathsf{Rumba}(m) &= F_0(m_0) \oplus F_1(m_1) \oplus F_2(m_2) \oplus F_3(m_3) \\ &= (x_0 + x_0^{20}) \oplus (x_1 + x_1^{20}) \oplus (x_2 + x_2^{20}) \oplus (x_3 + x_3^{20}) \ ,\end{aligned}$$

where each F_i is an instance of the function Salsa20 with distinct diagonal constants, and the remaining 384 bits of x_i contain the message. A single word j of x_i is denoted $x_{i,j}$. Note that the F_i functions include the feedforward operation of Salsa20. RumbaR stands for R-round variant. The constants for Rumba are in Tab. 8.5.

Table 8.5: Constants for Rumba.

	F_0	F_1	F_2	F_3
c_0	0x73726966	0x6f636573	0x72696874	0x72756f66
c_1	0x6d755274	0x7552646e	0x6d755264	0x75526874
c_2	0x30326162	0x3261626d	0x30326162	0x3261626d
c_3	0x636f6c62	0x6f6c6230	0x636f6c62	0x6f6c6230

8.7.2 Collisions and Preimages in Modified Versions

We show here the weakness of two modified versions of Rumba, respectively an iterated version with 2048-bit input compression function, and the compression function without the final feedforward.

On the Role of Diagonal Constants. The compression function Rumba is fed with 1536 bits, copied in a 2048-bit state whose remaining 512 bits correspond to the diagonal constants. It is tempting to see these values as the IV of an iterated hash function, and use diagonal values for the chaining variable. However, Bernstein implicitly warned against such a construction, when claiming that *"Rumba20 will take about twice as many cycles per eliminated byte as Salsa20 takes per encrypted byte"* [20]. Indeed, the 1536-bit input should contain both the 512-bit chaining value and the 1024-bit message, and thus for a 1024-bit input the Salsa20 function is called four times (256 bits processed per call), whereas in Salsa20 it's called once for a 512-bit input. We confirm here that diagonal values should *not* be replaced with the chaining variables, by presenting a method for finding collisions within about $2^{128}/6$ trials, against 2^{256} with a birthday attack: pick an arbitrary 1536-bit message block M^0, then compute $\mathsf{Rumba}(M^0) = h_0\|h_1\|h_2\|h_3$, and repeat this until two distinct 128-bit chunks h_i and h_j are equal, say h_0 and h_1, corresponding to the diagonal constants of F_0 and F_1 in the next round; consequently, these functions will be identical in the next round. A collision can then be obtained by choosing two distinct message blocks $M^1 = m_0^1\|m_1^1\|m_2^1\|m_3^1$ and $(M')^1 = m_1^1\|m_0^1\|m_2^1\|m_3^1$, or $M^1 = m_0^1\|m_0^1\|m_2^1\|m_3^1$ and $(M')^1 = (m'_0)^1\|(m'_0)^1\|m_2^1\|m_3^1$. How fast is this method? By the birthday paradox, the amount of trials for finding a suitable M^0 is about $2^{128}/6$ (here 6 is the number of distinct sets $\{i,j\} \subset \{0,\ldots,3\}$), while the construction of M^1 and $(M')^1$ is straightforward. Regarding the price-performance ratio, we do not have to store or sort a table, so the price is $2^{128}/6$ (and this for any potential filter function), while performance is much larger than one, because there are many collisions (one can choose 3 messages and 1 difference of 348 bits arbitrarily). This contrasts with the cost of 2^{256} for a serial attack on a 512-bit digest hash function.

On the Importance of Feedforward. In Davies-Meyer-based hash functions as well as in MD5 or SHA-1, the final feedforward is an obvious requirement for one-wayness. In Rumba, the feedforward is applied in each F_i, before a `XOR` of the four branches, and omitting this operation does not trivially lead to an inversion of the function, because of the incremental construction. However, as we will demonstrate, preimage resistance is not guaranteed with this setting. Let $F_i(m_i) = x_i^{20}$ for $i = 0,\ldots,3$ and assume that we are given a 512-bit value h, and our goal is to find M such that $\mathsf{Rumba}(M) = h$. This can be achieved by choosing *random* blocks m_0, m_1, m_2, and set

$$y = F_0(m_0) \oplus F_1(m_1) \oplus F_2(m_2) \oplus h \ . \tag{8.10}$$

We can find then the 512-bit state x_3^0 such that $y = x_3^{20}$. If x_3^0 has the correct diagonal values (the 128-bit constant of F_3), we can extract m_3 from x_0^3 with respect to Rumba's

definition. This randomized algorithm succeeds with probability 2^{-128}, since there are 128 constant bits in an initial state. Therefore, a preimage of an arbitrary digest can be found within about 2^{128} trials, against $2^{512/3}$ with the generalized birthday method.

8.7.3 Scenarios of Differential Attacks

To obtain a collision for RumbaR, it is sufficient to find two messages m and m' such that

$$F_0(m_0) \oplus F_0(m'_0) = F_2(m_2) \oplus F_2(m'_2) \ , \tag{8.11}$$

with $m_0 \oplus m'_0 = m_2 \oplus m'_2$, $m_1 = m'_1$ and $m_3 = m'_3$. The freedom in choosing m_1 and m_3 trivially allows to derive many other collisions (*multi-collision*). We use the following differential notation: Let the initial states x_i and x'_i have the input difference $\Delta_i = x_i \oplus x'_i$. After r rounds, the *observed* difference is denoted $\Delta_i^r = x_i^r \oplus (x'_i)^r$, and the output-difference $\mathcal{OD}$ (without feedforward) becomes $\Delta_i^R = x_i^R \oplus (x'_i)^R$. If feedforward is included in the $\mathcal{OD}$, we use the notation $\nabla_i^R = (x_i + x_i^R) \oplus (x'_i + (x'_i)^R)$. With this notation, Eq. 8.11 becomes $\nabla_0^R = \nabla_2^R$, and if the feedforward operation is ignored, then Eq. 8.11 simplifies to $\Delta_0^R = \Delta_2^R$. To find messages that satisfy these equations, we use an R-round differential path of high-probability, with intermediate target difference δ^r after r rounds, $0 \leq r \leq R$. Notice that the differential is applicable for both F_0 and F_2 (thus we do not have to subscript the target difference). The probability that a random message pair (with input-difference δ) conforms to δ^r is denoted p_r. To satisfy the equation $\Delta_0^R = \Delta_2^R$ it suffices to find message pairs such that the observed differentials equal the target one, that is, $\Delta_0^R = \delta^R$ and $\Delta_2^R = \delta^R$. The naive approach is to try about $1/p_r$ random messages each. This complexity can however be lowered down by

- Finding constraints on the message pair so that it conforms to the difference δ^1 after one round with certainty (this will be achieved by *linearization*).
- From a pair conforming to δ^r, deriving other pairs also conforming to δ^r (this will be achieved by the *neutral bits* technique).

Finally, to satisfy the equation $\nabla_0^R = \nabla_2^R$, it suffices to find message pairs such that $\nabla_0^R = \delta^R \oplus \delta$ and $\nabla_2^R = \delta^R \oplus \delta$ (*i.e.* the additions are not producing carry bits). Given a random message pair that conforms to δ^R, this holds with probability about 2^{-v-w} where v and w are the respective weights of the $\mathcal{ID}$ δ and of the target $\mathcal{OD}$ δ^R (excluding the linear msb's). The three next subsections are respectively dedicated to finding an optimal differential, describing the linearization procedure, and describing the neutral bits technique.

Remark 11. One can observe that the constants of F_0 and F_2 are almost similar, as well as the constants of F_1 and F_3. To improve the generalized birthday attack suggested in [20], a strategy is to find a pair (m_0, m_2) such that $F_0(m_0) \oplus F_2(m_2)$ is biased in any c bits after R rounds (where $c \approx 114$, *cf.* [20]), along with a second pair (m_1, m_3) with $F_1(m_1) \oplus F_3(m_3)$ biased in the same c bits. The sum $F_0(m_0) \oplus F_2(m_2)$ can be seen as the feedforward $\mathcal{OD}$ of two states having an $\mathcal{ID}$ which is nonzero in some diagonal words.

However, differences in the diagonal words result in a large diffusion, and this approach seems to be much less efficient than differential attacks for only one function F_i.

8.7.4 Finding High-Probability Differentials

We search for a linear differential, *i.e.* a differential holding with certainty in the linearized function F_i of Rumba (where addition is replaced by XOR, see Sect. 8.3.1). The differential is independent of the diagonal constants, and it is expected to have high probability in the genuine Rumba if the linear differential has low weight. An exhaustive search for suitable $\mathcal{ID}$'s is not traceable for linear Rumba, so we choose another method. As in Sect. 8.4, we focus on a single column in X_i and consider the weight of the input (starting with the diagonal element, which must be zero). With a fixed relative position of the non-zero bits in this input, one can obtain an output of low weight after the first linear quarterround. Here is a list of the mappings (showing the weight only) which have at most weight 2 in each word of the input and output:

$$
\begin{array}{lll}
g_1: & (0,0,0,0) & \rightarrow (0,0,0,0) \\
g_2: & (0,0,1,0) & \rightarrow (2,0,1,1) \\
g_3: & (0,0,1,1) & \rightarrow (2,1,0,2) \\
g_4: & (0,1,0,1) & \rightarrow (1,0,0,1) \\
g_5: & (0,1,1,0) & \rightarrow (1,1,0,1) \\
g_6: & (0,1,1,1) & \rightarrow (1,0,1,0) \\
g_7: & (0,0,2,1) & \rightarrow (2,1,1,1)
\end{array}
\qquad
\begin{array}{lll}
g_8: & (0,1,2,0) & \rightarrow (1,1,1,0) \\
g_9: & (0,1,2,2) & \rightarrow (1,1,1,2) \\
g_{10}: & (0,2,1,1) & \rightarrow (0,1,0,0) \\
g_{11}: & (0,2,1,2) & \rightarrow (0,0,1,1) \\
g_{12}: & (0,2,2,1) & \rightarrow (0,1,1,1) \\
g_{13}: & (0,2,2,1) & \rightarrow (2,1,1,1) \\
g_{14}: & (0,2,2,2) & \rightarrow (2,0,2,0)
\end{array}
$$

These relations can be used algorithmically to construct a suitable $\mathcal{ID}$ with all 4 columns. Consider the following example, where the state after the first round is again a combination of useful rows: $(g_1, g_{10}, g_1, g_{11}) \rightarrow (g_1, g_2, g_4, g_1)$. After 2 rounds, the difference has weight 6 (with weight 3 in the diagonal words). There is a class of $\mathcal{ID}$'s with the same structure: $(g_1, g_{10}, g_1, g_{11})$, $(g_1, g_{11}, g_1, g_{10})$, $(g_{10}, g_1, g_{11}, g_1)$, $(g_{11}, g_1, g_{10}, g_1)$. The degree of freedom is large enough to construct these 2-round linear differentials: the positions of the nonzero bits in a single mapping g_i are symmetric with respect to rotation of words (and the required g_i have an additional degree of freedom). Any other linear differential constructed with g_i has larger weight after 2 rounds. Eventually, here is the input difference we will consider for our attacks on Rumba (with optimal rotation, such that many msb's are involved):

$$
\begin{array}{ll}
\Delta_{i,2} = \texttt{0x00000002} & \Delta_{i,8} = \texttt{0x80000000} \\
\Delta_{i,4} = \texttt{0x00080040} & \Delta_{i,12} = \texttt{0x80001000} \\
\Delta_{i,6} = \texttt{0x00000020} & \Delta_{i,14} = \texttt{0x01001000}
\end{array}
$$

and $\Delta_{i,j} = 0$ for the other indices j. The weight of differences for the first four linearized rounds is as follows (the subscript of the arrows denotes the probability p_r that a random message pair conforms to this differential):

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 2 & 0 & 2 & 0 \end{pmatrix} \xrightarrow[2^{-4}]{\text{col}} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \xrightarrow[2^{-7}]{\text{row}} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 2 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \xrightarrow[2^{-41}]{\text{col}} \begin{pmatrix} 2 & 2 & 3 & 1 \\ 0 & 3 & 4 & 2 \\ 1 & 1 & 7 & 3 \\ 1 & 1 & 1 & 6 \end{pmatrix}$$

$$\xrightarrow[2^{-194}]{\text{row}} \begin{pmatrix} 8 & 3 & 2 & 4 \\ 5 & 10 & 3 & 4 \\ 9 & 11 & 13 & 7 \\ 6 & 9 & 10 & 9 \end{pmatrix}$$

With this fixed $\mathcal{ID}$, we can determine the probability that the $\mathcal{OD}$ obtained by genuine Rumba corresponds to the $\mathcal{OD}$ of linear Rumba. Note that integer addition is the only nonlinear operation. Each nonzero bit in the $\mathcal{ID}$ of an integer addition behaves linearly (*i.e.* it does not create or annihilate a sequence of carry bits) with probability 1/2, while a difference in the msb is always linear. In the first round, there are only four bits with associated probability 1/2, hence $p_1 = 2^{-4}$ (see also the subsection on linearization). The other cumulative probabilities are $p_2 = 2^{-7}$, $p_3 = 2^{-41}$, $p_4 = 2^{-194}$. For 3 rounds, we have weights $v = 7$ and $w = 37$, thus the overall complexity to find a collision after 3 rounds is about $2^{41+37+7} = 2^{85}$. For 4 rounds, $v = 7$ and $w = 112$, leading to a complexity 2^{313}. The probability that feedforward behaves linearly can be increased by choosing low-weight inputs.

8.7.5 Linearization

The first round of our differential has a theoretical probability of $p_1 = 2^{-4}$ for a random message. This is roughly confirmed by our experiments, where exact probabilities depend on the diagonal constants (for example, we experimentally observed $p_1 = 2^{-6.6}$ for F_0, and $p_1 = 2^{-6.3}$ for F_2, the other two probabilities are even closer to 2^{-4}). We show here how to set constraints on the message so that the first round differential holds with certainty.

Let us begin with the first column of F_0, where $c_{0,0} = x_{0,0} = \texttt{0x73726966}$. In the first addition $x_{0,0} + x_{0,12}$, we have to address $\Delta_{0,12}$, which has a nonzero (and non-msb) bit on position 12 (counting from 0). The bits of the constant are $[x_{0,0}]_{12-10} = (\texttt{010})_2$, hence the choice $[x_{0,12}]_{11,10} = (\texttt{00})_2$ is sufficient for linearization. This corresponds to $x_{0,12} \leftarrow x_{0,12} \wedge \texttt{0xFFFF3FFF}$. The subsequent 3 additions of the first column are always linear as only msb's are involved. Then, we linearize the third column of F_0, where $c_{0,2} = x_{0,10} = \texttt{0x30326162}$. In the first addition $x_{0,10}+x_{0,6}$, we have to address $\Delta_{0,6}$, which has a nonzero bit on position 5. The relevant bits of the constant are $[x_{0,10}]_{5-1} = (\texttt{10001})_2$, hence the choice $[x_{0,6}]_{4-1} = (\texttt{1111})_2$ is sufficient for linearization. This corresponds to $x_{0,6} \leftarrow x_{0,6} \vee \texttt{0x0000001E}$. In the second addition $y_{0,14}+x_{0,10}$, the updated difference $\Delta^1_{0,14}$ has a single bit on position 24. The relevant bits of the constant are $[x_{0,10}]_{24,23} = (\texttt{00})_2$, hence the choice $[y_{0,14}]_{23} = (\texttt{0})_2$ is sufficient. Notice that conditions on the updated words must be transformed to the initial state words. As $y_{0,14} = x_{0,14} \oplus (x_{0,10}+x_{0,6}) \lll 8$, we find

the condition $[x_{0,14}]_{23} = [x_{0,10}+x_{0,6}]_{16}$. If we let both sides be zero, we have $[x_{0,14}]_{23} = (0)_2$ or $x_{0,14} \leftarrow x_{0,14} \wedge \texttt{0xFF7FFFFF}$, and $[x_{0,10} + x_{0,6}]_{16} = (0)_2$. As $[x_{0,10}]_{16,15} = (00)_2$, we can choose $[x_{0,6}]_{16,15} = (00)_2$ or $x_{0,6} \leftarrow x_{0,6} \wedge \texttt{0xFFFE7FFF}$. Finally, the third addition $y_{0,2}+y_{0,14}$ must be linearized with respect to the single bit in $\Delta^1_{0,14}$ on position 24. A sufficient condition for linearization is $[y_{0,2}]_{24,23} = (00)_2$ and $[y_{0,14}]_{23} = (0)_2$. The second condition is already satisfied, so we can focus on the first condition. The update is defined by $y_{0,2} = x_{0,2} \oplus (y_{0,14}+x_{0,10}) \lll 9$, so we set $[x_{0,2}]_{24,23} = (00)_2$ or $x_{0,2} \leftarrow x_{0,2} \wedge \texttt{0xFE7FFFFF}$, and require $[y_{0,14} + x_{0,0}]_{15,14} = (00)_2$. As $[x_{0,10}]_{15-13} = (011)_2$, we can set $[y_{0,14}]_{15-13} = (101)_2$. This is satisfied by choosing $[x_{0,14}]_{15-13} = (000)_2$ or $x_{0,14} \leftarrow x_{0,14} \wedge \texttt{0xFFFF1FFF}$, and by choosing $[x_{0,10} + x_{0,6}]_{8-6} = (101)_2$. As $[x_{0,10}]_{8-5} = (1011)_2$, we set $[x_{0,6}]_{8-5} = (1111)_2$ or $x_{0,6} \leftarrow x_{0,6} \vee \texttt{0x000001E0}$. Altogether, we fixed 18 (distinct) bits of the input, other linearizations are possible.

The first round of F_2 can be linearized with exactly the same conditions. This way, we save an average factor of 2^4 (additive complexities are ignored). This linearization with sufficient conditions does not work well for more than one round because of an avalanche effect of fixed bits. We lose many degrees of freedom, and contradictions are likely to occur.

8.7.6 Neutral Bits

Thanks to linearization, we can find a message pair conforming to δ^2 within about $1/(2^{-7+4}) = 2^3$ trials. Our goal now is to to efficiently derive from such a pair many other pairs that are conforming to δ^2, so that a search for three rounds can start after the second round, by using the notion of neutral bits again (*cf.* Sect. 8.5.2).

Neutral bits can be identified easily for a fixed pair of messages, but if several neutral bits are complemented in parallel, then the resulting message pair may not conform anymore. A heuristic approach was introduced in [22], using a maximal 2-neutral set. A 2-neutral set of bits is a subset of neutral bits, such that the message pair obtained by complementing any two bits of the subset in parallel also conform to the differential. The size of this set is denoted n. In general, finding a 2-neutral set is an NP-complete problem (the problem is equivalent to the Maximum Clique Problem from graph theory), but good heuristic algorithms for dense graphs exist, *e.g.* see [31]. In the case of Rumba, we compute the value n for different message pairs that conform to δ^2 and choose the pair with maximum n. We observe that about $1/2$ of the 2^n message pairs (derived by flipping some of the n bits of the 2-neutral set) conform to the differential[3]. This probability p is significantly increased, if we complement at most $\ell \ll n$ bits of the 2-neutral set, which results in a message space (not contradicting with the linearization) of size about $p \cdot \binom{n}{\ell}$. At this point, a full collision for 3 rounds has a reduced theoretical complexity of $2^{85-7}/p = 2^{78}/p$ (of course, p should not be smaller than 2^{-3}). Since we will have $p > \frac{1}{2}$ for a suitable choice of ℓ, the complexity gets reduced from 2^{85} to less than 2^{79}.

[3]In the case of SHA-0, about $1/8$ of the 2^n message pairs (derived from the original message pair by complementing bits from the 2-neutral set) conform to the differential for the next round.

8.7.7 Experimental Results

We choose a random message of low weight, apply the linearization for the first round and repeat this about 2^3 times until the message pairs conforms to δ^2. We compute then the 2-neutral set of this message pair. This protocol is repeated a few times to identify a message pair with large 2-neutral set.

- For F_0, we find the pair of states (x_0, x_0') of low weight, which has 251 neutral bits and a 2-neutral set of size 147. If we flip a random subset of the 2-neutral bits, then the resulting message pair conforms to δ^2 with probability $\Pr = 0.52$.

$$x_0 = \begin{pmatrix} \texttt{0x73726966} & \texttt{0x00000400} & \texttt{0x00000080} & \texttt{0x00200001} \\ \texttt{0x00002000} & \texttt{0x6d755274} & \texttt{0x000001fe} & \texttt{0x02000008} \\ \texttt{0x00000040} & \texttt{0x00000042} & \texttt{0x30326162} & \texttt{0x10002800} \\ \texttt{0x00000080} & \texttt{0x00000000} & \texttt{0x01200000} & \texttt{0x636f6c62} \end{pmatrix}$$

- For F_2, we find the pair of states (x_2, x_2') of low weight, which has 252 neutral bits and a 2-neutral set of size 146. If we flip a random subset of the 2-neutral bits, then the resulting message pair conforms to δ^2 with probability $\Pr = 0.41$.

$$x_2 = \begin{pmatrix} \texttt{0x72696874} & \texttt{0x00000000} & \texttt{0x00040040} & \texttt{0x00000400} \\ \texttt{0x00008004} & \texttt{0x6d755264} & \texttt{0x000001fe} & \texttt{0x06021184} \\ \texttt{0x00000000} & \texttt{0x00800040} & \texttt{0x30326162} & \texttt{0x00000000} \\ \texttt{0x00000300} & \texttt{0x00000400} & \texttt{0x04000000} & \texttt{0x636f6c62} \end{pmatrix}$$

Given these pairs for 2 rounds, we perform a search in the 2-neutral set by flipping at most 10 bits (which gives a message space of about 2^{50}), to find pairs that conform to δ^3. This step has a theoretical complexity of about 2^{34} for each pair, which could be verified in practice. For example, in the case of (x_0, x_0'), we can flip the bits {59, 141, 150, 154, 269, 280, 294, 425} in order to get a pair of states $(\bar{x}_0, \bar{x}_0')$ that conforms to δ^3. In the case of (x_2, x_2'), the bits {58, 63, 141, 271, 304, 317, 435, 417, 458, 460} are flipped in order to get a pair of states $(\bar{x}_2, \bar{x}_2')$ that conforms to δ^3.

$$\bar{x}_0 = \begin{pmatrix} \texttt{0x73726966} & \texttt{0x08000400} & \texttt{0x00000080} & \texttt{0x00200001} \\ \texttt{0x04400000} & \texttt{0x6d755274} & \texttt{0x000001fe} & \texttt{0x02000008} \\ \texttt{0x01002040} & \texttt{0x00000002} & \texttt{0x30326162} & \texttt{0x10002800} \\ \texttt{0x00000080} & \texttt{0x00000200} & \texttt{0x01200000} & \texttt{0x636f6c62} \end{pmatrix}$$

$$\bar{x}_2 = \begin{pmatrix} \texttt{0x72696874} & \texttt{0x84000000} & \texttt{0x00040040} & \texttt{0x00000400} \\ \texttt{0x0000a004} & \texttt{0x6d755264} & \texttt{0x000001fe} & \texttt{0x06021184} \\ \texttt{0x00008000} & \texttt{0x20810040} & \texttt{0x30326162} & \texttt{0x00000000} \\ \texttt{0x00000300} & \texttt{0x00080402} & \texttt{0x04001400} & \texttt{0x636f6c62} \end{pmatrix}$$

At this point, we have collisions for 3 rounds of Rumba without feedforward, hence $\Delta_0^3 \oplus \Delta_2^3 = 0$. If we include feedforward for the above pairs of states, then $\nabla_0^3 \oplus \nabla_2^3$ has

weight 16, which corresponds to a near-collision. Notice that a near-collision of low weight indicates non-randomness of the reduced-round compression function (we assume a Gaussian distribution centered at 256). This near-collision of low weight was found by using a birthday method: we produce a list of pairs for F_0 that conform to δ^3 (using neutral bits as above), together with the corresponding value of ∇_0^3. The same is done for F_2. If each list has size N, then we can produce N^2 pairs of $\nabla_0^3 \oplus \nabla_2^3$ in order to identify near-collisions of low weight.

However, there are no neutral bits for the pairs $(\bar{x}_0, \bar{x}'_0)$ and $(\bar{x}_2, \bar{x}'_2)$ with respect to δ^3. This means that we can not completely separate the task of finding full collisions with feedforward, from finding collisions without feedforward (and we can not use neutral bits to iteratively find pairs that conform to δ^4). To find a full collision after three rounds, we could perform a search in the 2-neutral set of (x_0, x'_0) and (x_2, x'_2), by flipping at most 20 bits. In this case, the resulting pairs conform to δ^2 with probability at least $\Pr = 0.68$, and the message space has a size of about 2^{80}. The overall complexity becomes $2^{78}/0.68 \approx 2^{79}$ (compared to 2^{85} without linearization and neutral bits). Then, we try to find near-collisions of low weight for 4 rounds, using the birthday method described above. Within less than one minute of computation, we found pairs $(\bar{\bar{x}}_0, \bar{\bar{x}}'_0)$ and $(\bar{\bar{x}}_2, \bar{\bar{x}}'_2)$ such that $\nabla_0^4 \oplus \nabla_2^4$ has weight 129. Consequently, the non-randomness of the differential is propagating up to 4 rounds.

$$\bar{\bar{x}}_0 = \begin{pmatrix} \texttt{0x73726966} & \texttt{0x00020400} & \texttt{0x00000080} & \texttt{0x00200001} \\ \texttt{0x00002400} & \texttt{0x6d755274} & \texttt{0x000001fe} & \texttt{0x02000008} \\ \texttt{0x00000040} & \texttt{0x00220042} & \texttt{0x30326162} & \texttt{0x10002800} \\ \texttt{0x00000080} & \texttt{0x00001004} & \texttt{0x01200000} & \texttt{0x636f6c62} \end{pmatrix}$$

$$\bar{\bar{x}}_2 = \begin{pmatrix} \texttt{0x72696874} & \texttt{0x00001000} & \texttt{0x80040040} & \texttt{0x00000400} \\ \texttt{0x00008804} & \texttt{0x6d755264} & \texttt{0x000001fe} & \texttt{0x06021184} \\ \texttt{0x00000000} & \texttt{0x80800040} & \texttt{0x30326162} & \texttt{0x00000000} \\ \texttt{0x00000300} & \texttt{0x00000450} & \texttt{0x04000000} & \texttt{0x636f6c62} \end{pmatrix}$$

8.8 Summary

Salsa20 is widely viewed as a very promising proposal. Nothing in this chapter affects the security of the full version of the cipher. A new method for attacking Salsa20 and ChaCha (and maybe other ciphers) inspired by correlation attacks and by the notion of neutral bits has been presented. This allows to give the first attack faster than exhaustive search on the stream cipher Salsa20/8 with a 256-bit key. Thus Salsa20 still appears to be a conservative design, and Salsa20/12 could turn out to be a well-balanced proposal. For the compression function Rumba, which is built on Salsa20, the methods of linearization and neutral bits are applied to a high probability differential to find collisions on Rumba reduced to 3 rounds with expected 2^{79} trials, and to efficiently find low weight near collisions on 3-round and 4-round Rumba.

Chapter 9

Chosen IV Statistical Analysis

In the previous chapter, we have introduced the concept of probabilistic neutral bits (PNB's) in the key for the chosen IV scenario. Based on a recent framework for chosen IV statistical distinguishing analysis of stream ciphers, PNB's can be exploited to provide new and general methods for key recovery attacks. As an application, a key recovery attack on simplified versions of two eSTREAM candidates is given.

9.1 Introduction

The initialization function of a stream cipher should have good mixing properties, and it should be efficient (especially in hardware-oriented stream ciphers). If the initialization function uses a round-based approach, one can find a tradeoff between security and efficiency with a well-chosen number of rounds. In [55, 105, 112, 58], a framework for chosen IV statistical analysis of stream ciphers is suggested to investigate the structure of the initialization function. If mixing is not perfect, then the initialization function has an algebraic normal form which can be distinguished from a random one (*e.g.* if some high degree monomials are not produced). In this chapter, we optimize these methods with heuristic approaches (such as probabilistic neutral bits), and we present a framework to mount key recovery attacks. In [55], they say "*It is an open question how to utilize these weaknesses of state bits to attack the cipher.*". The aim of this chapter is to contribute to this problem. As in [55, 105] one selects a subset of IV bits as variables. Assuming all other IV values as well as the key fixed, one can write a keystream symbol as a Boolean function. By running through all possible values of these bits and generating a keystream output each time, one can compute the truth table of this Boolean function. Each coefficient in the algebraic normal form of this Boolean function is parametrized by the bits of the secret key. We now ask whether in the parametrized expression of a coefficient, every key bit does occur, or more generally, how much influence each key bit does have on the value of the coefficient. If a coefficient depends on less than all key bits, this fact can be exploited to filter those keys which do not satisfy the imposed value for the coefficient. In [120], it is shown that in the eSTREAM Phase 3 candidate Trivium with IV initialization reduced to 576 iterations, linear relations on the key bits can be derived

for well chosen sets of variable IV bits. Our framework is more general, as it works with the concept of (probabilistic) neutral key bits, *i.e.* key bits which have no influence on the value of a coefficient with some (high) probability. This way, we can get information on the key for many more iterations in the IV initialization of Trivium, and similarly for the eSTREAM Phase 3 candidate Grain-128. On the other hand, extensive experimental evidence indicates clear limits to our approach: With our methods, it is unlikely to get information on the key faster than exhaustive key search for Trivium or Grain-128 with full IV initialization.

9.2 Problem Formalization

Suppose that we are given a fixed Boolean function $F(K, V) : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}$. An oracle chooses a random and unknown $K = (k_0, \ldots, k_{n-1})$ and returns us the value of $z = F(K, V)$ for every query $V = (v_0, \ldots, v_{t-1})$ of our choice. The function F could stand *e.g.* for the Boolean function which maps the key K and IV V of a stream cipher to the (let say) first output bit. Our goal as an adversary is to determine the unknown key K (or to distinguish F from a random function) in the chosen IV attack model only by dealing with the function F. If F mixes its inputs in a proper way, then one needs to try all possible 2^n keys by sending $\mathcal{O}(n)$ queries to the oracle in order to find the correct key (since each query gives one bit information about the key for a balanced F). Here, we are going to investigate methods which can potentially lead to faster reconstruction of the key in the case where the function F does not properly mix its inputs. This could occur for example when the initialization phase of a stream cipher is performed through an iterated procedure for which the number of iterations has not been suitably chosen. On the other hand these methods may help to give the designers more insight to choose the required number of iterations. The existence of faster methods for finding the unknown key K highly depends on the structure of F. It may be even impossible to uniquely determine the key K. Let $F(K, V) = \bigoplus_\kappa c_\kappa(V) K^\kappa$ where $K^\kappa = k_0^{\kappa_0} \cdots k_{n-1}^{\kappa_{n-1}}$ for the multi-index $\kappa = (\kappa_0, \ldots, \kappa_{n-1})$. Then the following lemma makes this statement more clear.

Lemma 3. *No adversary can distinguish between the two keys K_1 and K_2 for which $K_1^\kappa = K_2^\kappa$ for all $\kappa \in \{0,1\}^n$ such that $c_\kappa(V) \neq 0$.*

Indeed, it is only possible to determine the values of $\{K^\kappa | \forall \kappa, c_\kappa(V) \neq 0\}$ which is not necessarily equivalent to determination of K. As a consequence of Lemma 3, the function F divides $\{0,1\}^n$ into *equivalence classes* $\mathcal{K}_1$, $\mathcal{K}_2, \ldots, \mathcal{K}_J$ (with $J \leq 2^n$). See Ex. 28 as an application on a reduced version of Trivium.

9.3 Scenarios of Attacks

The algebraic description of the function $F(K, V)$ is too complex in general to be amenable to direct analysis. Therefore, from the function $F(K, V)$ and with the partition $V = (U, W)$ we derive simpler Boolean functions $C(K, W)$ with the help of the oracle. In

our main example, $C(K, W)$ is a coefficient of the algebraic normal form of the function deduced from F by varying over the bits in U only (see Sect. 9.4 for more details). If this function $C(K, W)$ does not have a well-distributed algebraic structure, it can be exploited in cryptanalytic attacks. Let us investigate different scenarios:

1. If $C(K, W)$ is imbalanced for (not necessarily uniformly) random W and many fixed K, then the function F (or equivalently the underlying stream cipher) with unknown K can be distinguished from a random one, see [55, 105, 112, 58].
2. If $C(K, W)$ is evaluated for some fixed W, then $C(K, W)$ is an expression in the key bits only. In [120], it was shown that in Trivium case for reduced iterations, linear relations on the key bits can be derived for a well chosen IV part.
3. If $C(K, W)$ has many key bits, which have (almost) no influence on the values of $C(K, W)$, a suitable approximation may be identified and exploited for key recovery attacks. This is the target scenario of this chapter and will be discussed in detail.

In scenario 2, the underlying idea is to find a relation $C(K, W)$, evaluated for some W, which depends only on a subset of m ($< n$) key bits. The functional form of this relation can be determined with 2^m evaluations of $C(K, W)$. By trying all 2^m possibilities for the involved m key bits, one can filter those keys which do not satisfy the imposed relation. The complexity of this precomputation is 2^m times needed to compute $C(K, W)$, see Sect. 9.4. More precisely, if $p = \Pr(C(K, W) = 0)$ for the fixed W, the key space is filtered by a factor of $H(p) = p^2 + (1-p)^2$. For example, in the case of a linear function it is $p = H(p) = 1/2$. In addition, if several imposed relations on the key bits are available, it is easier to combine them to filter wrong keys if they have a simple structure, see *e.g.* [120]. In scenario 3, our main idea is to find a function $A(M, W)$ which depends on a key part M of m bits, and which is correlated to $C(K, W)$ with correlation coefficient ε, that is $\Pr(C(K, W) = A(M, W)) = 1/2 + \varepsilon$. Then, by asking the oracle N queries we get some information (depending on the new equivalence classes produced by A) about m bits of the secret K in time $N2^m$ by carefully analyzing the underlying hypothesis testing problem. We will proceed by explaining how to derive such functions C from the coefficients of the ANF of F in Sect. 9.4, and how to find such functions A using the concept of probabilistic neutral bits in Sect. 9.5.

9.4 Derived Functions from Polynomial Description

The function F can be written in the form $F(K, V) = \bigoplus_{\nu,\kappa} c_{\nu,\kappa} V^\nu K^\kappa$ with binary coefficients $c_{\nu,\kappa}$. We can make a partition of the IV according to $V = (U, W)$ and $\nu = (\alpha, \beta)$ with u bit segments U and α, and $w = t - u$ bit segments W and β. This gives the expression $F(K, V) = \bigoplus_{\alpha,\beta,\kappa} c_{(\alpha,\beta),\kappa} U^\alpha W^\beta K^\kappa = \bigoplus_\alpha c_\alpha(K, W) U^\alpha$ where $c_\alpha(K, W) = \bigoplus_{\beta,\kappa} c_{(\alpha,\beta),\kappa} W^\beta K^\kappa$. For every $\alpha \in \{0, 1\}^u$, the function $c_\alpha(K, W)$ can serve as a function C derived from F. Here is a toy example to illustrate the notation:

Example 26. Let $n = t = 3$ and $F(K, V) = k_1v_1 \oplus k_2v_0v_2 \oplus v_2$. Let $U := (v_0, v_2)$ of $u = 2$ bits and $W := (v_1)$ of $w = 1$ bit. Then $C_0(K, W) = k_1v_1$, $C_1(K, W) = 0$, $C_2(K, W) = 1$, $C_3(K, W) = k_2$.

Note that an adversary with the help of the oracle can evaluate $c_\alpha(K, W)$ for the unknown key K at any input $W \in \{0, 1\}^w$ for every $\alpha \in \{0, 1\}^u$ by sending at most 2^u queries to the oracle, *i.e.* the partitioning of V helps us to define a computable function $c_\alpha(K, W)$ if u is small enough (even though the explicit form of $c_\alpha(K, W)$ remains unknown). To obtain the values $c_\alpha(K, W)$ for all $\alpha \in \{0, 1\}^u$, an adversary asks for the output values of all 2^u inputs $V = (U, W)$ with the fixed part W. This gives the truth table of a Boolean function in u variables for which the coefficients of its algebraic normal form (*i.e.* the values of $c_\alpha(K, W)$) can be found in time $u2^u$ and memory 2^u using the Walsh-Hadamard transform. Alternatively, a *single* coefficient $c_\alpha(K, W)$ for a specific $\alpha \in \{0, 1\}^u$ can be computed by `XOR`ing the output of F for all $2^{|\alpha|}$ inputs $V = (U, W)$ for which each bit of U is at most as large as the corresponding bit of α. This bypasses the need of 2^u memory.

One can expect that a subset of IV bits receives less mixing during the initialization process than other bits. These IV bits are called *weak*, and they would be an appropriate choice of U in order to amplify the non-randomness of C. However, it is an open question how to identify weak IV bits by systematic methods.

9.5 Functions Approximation

We are interested in the approximations of a function $C(K, W) : \{0, 1\}^n \times \{0, 1\}^w \to \{0, 1\}$ which depend only on a subset of key bits. To this end we make an appropriate partition of the key K according to $K = (L, M)$ with M (called *subkey*) of m bits and L of $l = n-m$ bits, and construct the function $A(M, W)$. Such a partitioning can be identified using the concept of probabilistic neutral bits (PNB's), see Sect. 8.5.2 (where the notion of PNB's was used to derive a suitable function A in the case of $W = V$ and $C = F$). We recall that the *neutrality measure* of the key bit k_i is defined as γ_i where $1/2 + \gamma_i$ is the probability (over all K and W) that complementing the key bit k_i does not change the output of C. A key bit with large $|\gamma_i|$ is called *weak* key bit (and it is called a *significant* key bit otherwise). In practice, we will set a threshold γ, such that all key bits with $|\gamma_i| < \gamma$ are included in the subkey M. The approximation $A(M, W)$ could then be defined by $C(K, W)$ either with a fixed or a randomly chosen value for non-significant key bits L. Here is another toy example to illustrate the method:

Example 27. Let $n = t = 3$, $u = 2$ and $C(K, W) = k_0k_1k_2v_0v_1 \oplus k_0v_1 \oplus k_1v_0$. For uniformly random K and W, we find $\gamma_0 = 1/8$, $\gamma_1 = 1/8$, $\gamma_2 = 7/8$. Consequently, it is reasonable to use $M := (k_0, k_1)$ as the subkey. With fixed $k_2 = 0$, we obtain the approximation $A(M, W) = k_0v_1 \oplus k_1v_0$ which depends on $m = 2$ key bits only.

Note that, if L consists only of neutral key bits, then the approximation A is exact, because $C(K, W)$ does not depend on these key bits.

9.6 Description of the Attack

In the precomputation phase of the attack, we need a suitable partitioning of the IV and the key (*i.e.* a function C and an approximation A). The weak IV bits are often found by a random search, while the weak key bits can be easily found with the neutrality measure for some threshold γ. Given C and A, we can find a small subset of candidates for the subkey M with a probabilistic guess-and-determine attack. In order to filter the set of all 2^m possible subkeys into a smaller set, we need to distinguish a correct guess of the subkey M' from an incorrect one. Our ability in distinguishing subkeys is related to the correlation coefficient between $A(M', W)$ and $C(K, W)$ with $K = (L, M')$ under the following two hypotheses. H_0 : the guessed part M' is correct, and H_1 : the guessed part M' is incorrect. More precisely, the values of ε_0 and ε_1 defined in the following play a crucial role:

$$\Pr_W(A(M', W) = C(K, W)|K = (L, M')) = \frac{1}{2} + \varepsilon_0 \tag{9.1}$$

$$\Pr_{M',W}(A(M', W) = C(K, W)|K = (L, M)) = \frac{1}{2} + \varepsilon_1 \ . \tag{9.2}$$

In general, both ε_0 and ε_1 are random variables, depending on the key. If the distributions of ε_0 and ε_1 are separated, we can achieve a small non-detection probability p_β and false alarm probability $p_\alpha = 2^{-c}$ by using enough samples. In the special case where ε_0 and ε_1 are constants (corresponding to the binary distributions D_0 and D_1), the optimum distinguisher is Neyman-Pearson, see Sect. 2.8. The attack will be successful with probability $1 - p_\beta$ and the complexity is as follows: For each guess M' of the subkey, the bias ε of $A(M', W) \oplus C(K, W)$ must be computed, which requires computation of the coefficients $A(M', W)$ by the adversary, and computation of the coefficient $C(K, W)$ through the oracle, for the same N values of W, having a cost of $N2^u$ at most. This must be repeated for all 2^m possible guesses M'. The set of candidates for the subkey M has a size of about $p_\alpha 2^m = 2^{m-c}$. The whole key can then be verified by an exhaustive search over the key part L with a cost of $2^{m-c}2^l = 2^{n-c}$ evaluations of F. The total complexity becomes $C_T = N2^u 2^m + 2^{m-c}2^{n-m} = N2^{u+m} + 2^{n-c}$. The required number of samples is $N = d/\Delta(D_0, D_1)$ (assuming that $C(K, W)$ are independent) to obtain an overall error $p_e = \Phi(-\sqrt{d}/2) \approx p_\alpha$. Using more than one function C or considering several chosen IV bits U may be useful to reduce complexity; however, we do not deal with this case here.

Remark 12. In practice the distributions of ε_0 and ε_1 for different subkeys may not be fully separated, and hence a very small p_β and p_α may not be possible to achieve. However, we propose the following non-optimal distinguisher. We choose a threshold ε_0' such that $\Pr(\varepsilon_0 > \varepsilon_0')$ has a significant value, *e.g.* 1/2. We also identify a threshold ε_1', if possible, such that $\Pr(\varepsilon_1 < \varepsilon_1') = 1$. Then, we estimate the sample size by replacing ε_0 and ε_1 by ε_0' and ε_1', respectively, to obtain $p_\alpha \leq 2^{-c}$ and desired $p_\beta \approx 1/2$. If ε_0' and ε_1' are close, then the estimated number of samples becomes very large. In this case, it is better to chose the number of samples intuitively, and then estimate the related p_α.

Remark 13. It is reasonable to assume that a false subkey M', which is close to the correct subkey, may lead to a larger value of ε. Here, the measure for being "close" could be the neutrality measure γ_i and the Hamming weight: if only a few key bits on positions with large γ_i are false, one would expect that ε is large. However, we only observed an irregular (*i.e.* not continuous) deviation for very close subkeys. The effect on p_α is negligible because subkeys with difference of low weight are rare.

9.7 Application to Trivium

The eSTREAM Phase 3 candidate Trivium [32] has an internal state of 288 bits. To initialize the cipher, the $n = 80$ key bits and $t = 80$ IV bits are written into the registers. The cipher state is then updated $R = 18 \times 64 = 1152$ times without producing output in order to provide a good mixture of the key and IV bits in the initial state. We consider the Boolean function $F(K, V)$ which computes the first keystream bit after r rounds of initialization. In [55], Trivium was analyzed with chosen IV statistical tests and non-randomness was detected for $r = 10 \times 64, 10.5 \times 64, 11 \times 64, 11.5 \times 64$ rounds with $u = 13, 18, 24, 33$ IV bits, respectively. In [120], the key recovery attack on Trivium was investigated with respect to scenario 2 (see Sect. 9.3) for $r = 9 \times 64$. In this section we provide more examples for key recovery attack with respect to scenario 3 for $r = 10 \times 64$ and $r = 10 \times 10.5$. In the following two examples, weak IV bits have been found by a random search. We first concentrate on equivalence classes of the key:

Example 28. For $r = 10 \times 64$ rounds, a variable IV part U with the $u = 10$ bit positions {34, 36, 39, 45, 63, 65, 69, 73, 76, 78}, and the coefficient with index $\alpha = 1023$, we could experimentally verify that the derived function $c_\alpha(K, W)$ only depends on $m = 10$ key bits M with bit positions {15, 16, 17, 18, 19, 22, 35, 64, 65, 66}. By assigning all 2^{10} different possible values to these 10 key bits and putting those M which gives the same function $c_\alpha(K, W)$ (by trying enough samples of W), we could determine the equivalence classes for M with respect to c_α. Our experiment shows the existence of 65 equivalence classes: one with 512 members for which $k_{15}k_{16} \oplus k_{17} \oplus k_{19} = 0$ and 64 other classes with 8 members for which $k_{15}k_{16} \oplus k_{17} \oplus k_{19} = 1$ and the vector $(k_{18}, k_{22}, k_{35}, k_{64}, k_{65}, k_{66})$ has a fixed value. This shows that c_α provides $\frac{1}{2} \times 1 + \frac{1}{2} \times 7 = 4$ bits of information about the key in average. □

Example 29. For $r = 10 \times 64$ rounds, a variable IV part U with the $u = 11$ bit positions {1, 5, 7, 9, 12, 14, 16, 22, 24, 27, 29}, and the coefficient with index $\alpha = 2047$, the derived function $c_\alpha(K, W)$ depends on all 80 key bits. A more careful look at the neutrality measure of the key bits reveals that $\max(\gamma_i) \approx 0.18$ and only 7 key bits have a neutrality measure larger than $\gamma = 0.09$, which is not enough to get a useful approximation $A(M, W)$ for an attack. However, we observed that $c_\alpha(K, W)$ is independent of the key for $W = 0$, and more generally the number of significant bits depends on $|W|$. □

It is difficult to find a good choice of variable IV's for larger values of r, using a random search. The next example shows how we can go a bit further with some insight.

Example 30. Now we consider $r = 10.5 \times 64 = 10 \times 64 + 32 = 672$ rounds. The construction of the initialization function of Trivium suggests that shifting the bit positions of U in Ex. 29 may be a good choice. Hence we choose U with the $u = 11$ bit positions {33, 37, 39, 41, 44, 46, 48, 54, 56, 59, 61}, and $\alpha = 2047$. In this case, $c_\alpha(K, W)$ for $W = 0$ is independent of 32 key bits, and $p = \Pr(c_\alpha(K, 0) = 1) \approx 0.42$. This is already a reduced attack which is $1/H(p) \approx 1.95$ times faster than exhaustive search. □

The following example shows how we can connect a bridge between scenarios 2 and 3 and come up with an improved attack.

Example 31. Consider the same setup as in Ex. 30. If we restrict ourself to W's with $|W| = 5$ and compute the value of γ_i conditioned over these W, then $\max_i(\gamma_i) \approx 0.34$. Assigning all key bits with $|\gamma_i| < \gamma = 0.13$ as significant, we obtain a key part M with the $m = 29$ bit positions {1, 3, 10, 14, 20, 22, 23, 24, 25, 26, 27, 28, 31, 32, 34, 37, 39, 41, 46, 49, 50, 51, 52, 57, 59, 61, 63, 68, 74}. Our analysis of the function $A(M, W)$ shows that for about 44% of the keys we have $\varepsilon_0 > \varepsilon_0' = 0.1$ when the subkey is correctly guessed. If the subkey is not correctly guessed, we observe $\varepsilon_1 < \varepsilon_1' = 0.08$. Then, the correct subkey of 29 bits can be detected using at most $N \approx 2^{15}$ samples, with time complexity $N2^{u+m} \approx 2^{55}$. Note that the condition $N < \binom{69}{5}$ is satisfied here. □

9.8 Application to Grain

The eSTREAM Phase 3 candidate Grain-128 [73] consists of an LFSR, an NFSR and an output function $h(x)$. It has $n = 128$ key bits, $t = 96$ IV bits and the full initialization function has $R = 256$ rounds. We consider again the Boolean function $F(K, V)$ which computes the first keystream bit of Grain-128 after r rounds of initialization. In [55], Grain-128 was analyzed with chosen IV statistical tests. With $N = 2^5$ samples and $u = 22$ variable IV bits, they observed a non-randomness of the first keystream bit after $r = 192$ rounds. They also observed a non-randomness in the initial state bits after the full number of rounds. In [105], a non-randomness up to 313 rounds was reported (without justification). In this section we provide key recovery attack for up to $r = 180$ rounds with slightly reduced complexity compared with exhaustive search. In the following example, weak IV bits for scenario 2 have been found again by a random search.

Example 32. Consider $u = 7$ variable IV bits U with bit positions {2, 6, 8, 55, 58, 78, 90}. For the coefficient with index $\alpha = 127$ (corresponding to the monomial of maximum degree), a significant imbalance for up to $r = 180$ rounds can be detected: the monomial of degree 7 appears only with a probability of $p < 0.2$ for 80% of the keys. Note that in [55], the attack with $u = 7$ could only be applied to $r = 160$ rounds, while our improvement comes from the inclusion of weak IV bits. □

In the following examples, our goal is to show that there exists some reduced key recovery attack for up to $r = 180$ rounds on Grain-128. We use the same weak IV's as in the previous example.

Example 33. Consider again the $u = 7$ IV bits U with bit positions {2, 6, 8, 55, 58, 78, 90}. For $r = 150$ rounds we choose the coefficient with index $\alpha = 117$ and include key bits with neutrality measure less than $\gamma = 0.49$ in the list of significant key bits. This gives a subkey M of $m = 99$ bits. Our simulations show that $\varepsilon_0 > \varepsilon_0' = 0.48$ for about 95% of the keys, hence $p_\beta = 0.05$. On the other hand, for 128 wrong guesses of the subkey with $N = 200$ samples, we never observed that $\varepsilon_1 > 0.48$, hence $p_\alpha < 2^{-7}$. This gives an attack with time complexity $N2^{m+u} + 2^n p_\alpha \approx 2^{121}$ which is an improvement of a factor of (at least) $1/p_\alpha = 2^7$ compared to exhaustive search. □

Example 34. With the same choice for U as in Ex. 32 and 33, we take $\alpha = 127$ for $r = 180$ rounds. We identified $m = 110$ significant key bits for M. Our simulations show that $\varepsilon_0 > \varepsilon_0' = 0.4$ in about 30% of the runs when the subkey is correctly guessed. For 128 wrong guesses of the subkey with $N = 128$ samples, we never observed that $\varepsilon_1 > 0.4$. Here we have an attack with time complexity $N2^{m+u} + 2^n p_\alpha \approx 2^{124}$, *i.e.* an improvement of a factor of 2^4. □

9.9 Summary

A recent framework for chosen IV statistical distinguishers for stream ciphers has been exploited to provide new methods for key recovery attacks. This is based on a polynomial description of output bits as a function of the key and the IV. A deviation of the algebraic normal form (ANF) from random indicates that not every bit of the key or the IV has full influence on the value of certain coefficients in the ANF. It has been demonstrated how this can be exploited to derive information on the key faster than exhaustive key search through approximation of the polynomial description and using the concept of probabilistic neutral key bits. Two applications of our methods through extensive experiments have been given: A reduced complexity key recovery for Trivium with IV initialization reduced to 672 of its 1152 iterations, and a reduced complexity key recovery for Grain-128 with IV initialization reduced to 180 of its 256 iterations. This answers positively the question whether statistical distinguishers based on polynomial descriptions of the IV initialization of a stream cipher can be successfully exploited for key recovery. On the other hand, our methods are not capable to provide reduced complexity key recovery of the eSTREAM Phase 3 candidates Trivium and Grain-128 with full initialization.

Chapter 10

Conclusions

The design and analysis of stream ciphers is a fascinating field of activity. It is scientifically interesting to investigate how much the number of mathematical operations can be reduced to construct a cipher that is still secure. For example, are a few integer additions and bitwise operations sufficient to provide a high level of security? Any insight of this type can also be useful in the design of efficient cryptographic hash functions. On the other hand, there is a practical need for stream ciphers, which will likely be increasing in the coming future due to lightweight applications such as RFID. In this thesis, we addressed cryptanalysis of lightweight stream ciphers. We presented the whole range from full attacks, identification of some partial weaknesses, or verification of security, and we derived or improved cryptanalytic methods for different building blocks (with a focus on algebraic attacks). In particular, we conclude that T-functions have not delivered a good performance in practice, while shift registers with carry are still promising. The security of the reputable hardware-oriented designs Trivium and Grain-128 could be verified with respect to our methods, but the security margin seems larger for software-oriented stream ciphers: only 8 out of 20 rounds of Salsa20 can be broken yet. It will be interesting to see how the winners of eSTREAM will prove in practice.

Appendix A

Attack on MAG

MAG is a synchronous stream cipher submitted to the eSTREAM project. We present a very simple distinguishing attack (with some predicting feature) on MAG, requiring only 129 successive bytes of known keystream, computation and memory are negligible. The attack has been verified.

A.1 Brief Description

In the standard version of the stream cipher MAG [121], the internal state consists of 127 registers x_i of 32 bit size, as well as a carry register C of 32 bit size. The secret key is used to initialize all registers $x_0, \ldots, x_{126}$ and C (where the details of the key setup are not important for the attack). In order to produce the keystream, MAG is applied iteratively; a single iteration consists of an update and an output period. The description of the update does not seem to be consistent in the paper and in the provided code; we will refer to the code (however, the attack is of very general nature and may also work for other versions). In update period i, the carry C and register x_i are modified. In a first step of the algorithm, two neighboring registers are compared in order to determine the operation for the carry update, and in a second step, the carry is used to update register x_i; more precisely,

$$C' = \begin{cases} C \oplus x_{i+1} & \text{if } x_{i+2} > x_{i+3} \\ C \oplus \bar{x}_{i+1} & \text{otherwise} \end{cases} \tag{A.1}$$

$$x_i' = x_i \oplus C'. \tag{A.2}$$

Here, $\bar{x}$ denotes the complement of x; updated variables are primed. Notice that a register x_i is updated only once in 127 iterations, whereas the carry C is updated in each step of iteration. We point out that comparison of registers is the only operation on words, whereas `XOR` and complementation are operations on bits. It remains to describe the (cryptographic) output of MAG: in output period i, the string $x_i \mod 256$ is sent to the keystream z_i (notice that addition of indices in x_i is performed modulo 128, whereas indices in z_i are continuous).

A.2 Distinguishing Attack

The first 127 bytes of keystream z_i reveal the 8 least significant bits (lsb's) of all registers x_i, and the additional keystream byte z_{127} reveals the 8 lsb's of the updated register x'_0. Given these 128 successive bytes of keystream byte $z_0, \ldots, z_{127}$, it is possible to compute two strings, one of them corresponding to the next keystream byte z_{128}: first, Eq. A.2 defines how to reveal the corresponding carry, namely $C' = x_0 \oplus x'_0$. According to Eq. A.1, the carry is updated by $C'' = C' \oplus x_1$ or by $C'' = C' \oplus \bar{x}_1$ (with equal probability). Finally, the register x_1 is updated by $x'_1 = C'' \oplus x_1$. These relations can be reduced modulo 256 (in order to make use of the known keystream bytes) and combined; using the fact that they also hold for other indices, we conclude

$$z_{i+128} = \begin{cases} z_i \oplus z_{i+1} \oplus z_{i+2} \oplus z_{i+127} & \text{with Pr} = 1/2 \\ z_i \oplus z_{i+1} \oplus \bar{z}_{i+2} \oplus z_{i+127} & \text{with Pr} = 1/2 \end{cases} \tag{A.3}$$

Prediction of z_{i+128} may be used to distinguish the keystream of the cipher from a truly random sequence: given the actual keystream byte z_{i+128}, the attacker may verify if it corresponds to one of the two results of Eq. A.3. If not, the keystream is not produced by MAG. If yes, the keystream is produced by MAG with a probability of error corresponding to $p_\alpha = 1/128$. In order to reduce the error p_α (false alarms), more keystream may be used to verify Eq. A.3. Furthermore, the distinguisher may be used to recover some part of the state; each byte of keystream reveals one bit of information, namely the path of the branching. However, we did not study the state-recovery attack in more detail.

We conclude that the design of MAG has substantial weaknesses; revealing some part of the internal state, and sparse use of operations on words may be delicate choices of design for a secure stream cipher.

A.3 Example of an Attack

The attack was verified, using the code provided in [121]. In Tab. A.1, we give an example of keystream produced by the standard implementation of MAG, initialized with the zero seed. According to the previous section, we verify the non-randomness of the last red-colored byte z_{128} (where the index counts from 0): Eq. A.3 yields that either $z_{128} =$ `0x05` $\oplus$ `0xF0` $\oplus$ `0x53` $\oplus$ `0x16` $=$ `0xB0` or $z_{128} =$ `0x05` $\oplus$ `0xF0` $\oplus$ `0x53` $\oplus$ `0xE9` $=$ `0x4F`; obviously, the first result is the appropriate one.

Table A.1: Some example keystream produced by standard implementation of MAG for the zero seed.

0x05 53 16 29 77 23 33 5C 05 FC F8 57 26 1A 98 6B
0xAD 33 E2 2F 02 1B 3D 2E 82 44 82 E9 BF 8E C3 88
0x0F FE 88 21 2E 5D 6E EA 6B 62 1C 62 4D 7B 51 27
0x75 CE 34 53 CA 2A 32 B9 56 23 43 2C 19 5C 14 AE
0xC5 42 BA A8 59 11 8F 41 F0 48 2B 81 4D 52 C7 EA
0xB0 F5 BA 76 62 9B 93 7D 93 24 9C C2 7B 70 EE 3D
0x44 02 B8 E3 CF DF 36 7D EE F3 00 79 20 23 7A 60
0xB3 8B AD 3E 1B F4 BB 57 AF 99 53 AF 5C C7 88 F0
0xB0 23 6B 16 8E 3D 57 0D 0C A0 29 BD 19 F0 51 5B

Bibliography

[1] Ross J. Anderson. Searching for the Optimum Correlation Attack. In Bart Preneel, editor, *FSE*, volume 1008 of *Lecture Notes in Computer Science*, pages 137–143. Springer, 1994.

[2] Frederik Armknecht. Improving Fast Algebraic Attacks. In Roy and Meier [110], pages 65–82.

[3] Frederik Armknecht, Claude Carlet, Philippe Gaborit, Simon Künzli, Willi Meier, and Olivier Ruatta. Efficient Computation of Algebraic Immunity for Algebraic and Fast Algebraic Attacks. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 147–164. Springer, 2006.

[4] Frederik Armknecht and Matthias Krause. Constructing Single- and Multi-output Boolean Functions with Maximal Algebraic Immunity. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP (2)*, volume 4052 of *Lecture Notes in Computer Science*, pages 180–191. Springer, 2006.

[5] François Arnault and Thierry P. Berger. Design and Properties of a New Pseudorandom Generator Based on a Filtered FCSR Automaton. *IEEE Transactions on Computers*, 54(11):1374–1383, 2005.

[6] François Arnault, Thierry P. Berger, and Cédric Lauradoux. Update on F-FCSR Stream Cipher. Technical Report 2006/025, eSTREAM, ECRYPT Stream Cipher Project, 2006.

[7] François Arnault, Thierry P. Berger, and Marine Minier. On the Security of FCSR-based Pseudorandom Generators. In *SASC* [54], pages 179–190.

[8] François Arnault, Thierry P. Berger, and Marine Minier. Some Results on FCSR Automata with Applications to the Security of FCSR-based Pseudorandom Generators. To appear in *IEEE Transactions on Information Theory*, 2008.

[9] Gwénolé Ars. *Application des Bases de Gröbner à la Cryptographie.* PhD thesis, Université de Rennes (France), 2005.

[10] Steve Babbage. A Space/Time Tradeoff in Exhaustive Search Attacks on Stream Ciphers. *IEE Conference Publication, European Convention on Security and Detection*, 408, 1995.

[11] Thomas Baignères, Pascal Junod, and Serge Vaudenay. How Far can we go Beyond Linear Cryptanalysis? In Lee [90], pages 432–450.

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

[13] Mihir Bellare and Daniele Micciancio. A New Paradigm for Collision-Free Hashing: Incrementality at Reduced Cost. In Walter Fumy, editor, *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 163–192. Springer, 1997.

[14] Vincent Bénony, François Recher, Eric Wegrzynowski, and Caroline Fontaine. Cryptanalysis of a Particular Case of Klimov-Shamir Pseudo-Random Generator. In Tor Helleseth, Dilip V. Sarwate, Hong-Yeop Song, and Kyeongcheol Yang, editors, *SETA*, volume 3486 of *Lecture Notes in Computer Science*, pages 313–322. Springer, 2004.

[15] Côme Berbain, Henri Gilbert, and Alexander Maximov. Cryptanalysis of Grain. In Matthew J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2006.

[16] Daniel J. Bernstein. Related-key Attacks: Who Cares? eSTREAM Discussion Forum, ECRYPT Stream Cipher Project, 2005.

[17] Daniel J. Bernstein. Salsa20. Technical Report 2005/025, eSTREAM, ECRYPT Stream Cipher Project, 2005.

[18] Daniel J. Bernstein. Salsa20/8 and Salsa20/12. Technical Report 2006/007, eSTREAM, ECRYPT Stream Cipher Project, 2006.

[19] Daniel J. Bernstein. Salsa20 and ChaCha. eSTREAM Discussion Forum, ECRYPT Stream Cipher Project, 2007.

[20] Daniel J. Bernstein. What Output Size Resists Collisions in a XOR of Independent Expansions? In *ECRYPT Hash Workshop*. ECRYPT Network of Excellence in Cryptology, 2007.

[21] Daniel J. Bernstein. Chacha, a Variant of Salsa20. In *SASC*, pages 273–278. ECRYPT Network of Excellence in Cryptology, 2008.

[22] Eli Biham and Rafi Chen. Near-Collisions of SHA-0. In Franklin [59], pages 290–305.

[23] Eli Biham and Orr Dunkelman. Differential Cryptanalysis in Stream Ciphers. Technical Report 2007/218, Cryptology ePrint Archive, 2007.

[24] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *J. Cryptology*, 4(1):3–72, 1991.

[25] Alex Biryukov and Adi Shamir. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In Tatsuaki Okamoto, editor, *ASIACRYPT*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2000.

[26] Alex Biryukov, Adi Shamir, and David Wagner. Real Time Cryptanalysis of A5/1 on a PC. In Bruce Schneier, editor, *FSE*, volume 1978 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2000.

[27] An Braeken and Joseph Lano. On the (Im)Possibility of Practical and Secure Nonlinear Filters and Combiners. In Bart Preneel and Stafford E. Tavares, editors, *Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 159–174. Springer, 2005.

[28] An Braeken, Joseph Lano, Nele Mentens, Bart Preneel, and Ingrid Verbauwhede. SFINKS: A Synchronous Stream Cipher for Restricted Hardware Environments. Technical Report 2005/026, eSTREAM, ECRYPT Stream Cipher Project, 2005.

[29] An Braeken and Bart Preneel. On the Algebraic Immunity of Symmetric Boolean Functions. In Subhamoy Maitra, C. E. Veni Madhavan, and Ramarathnam Venkatesan, editors, *INDOCRYPT*, volume 3797 of *Lecture Notes in Computer Science*, pages 35–48. Springer, 2005.

[30] Gilles Brassard, editor. *Advances in Cryptology - CRYPTO 1989, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*. Springer, 1990.

[31] Samuel Burer, Renato D.C. Monteiro, and Yin Zhang. Maximum Stable Set Formulations and Heuristics Based on Continuous Optimization. *Mathematical Programming*, 64:137–166, 2002.

[32] Christophe De Cannière. Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *ISC*, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186. Springer, 2006.

[33] Anne Canteaut. Open Problems Related to Algebraic Attacks on Stream Ciphers. In Øyvind Ytrehus, editor, *Coding and Cryptography*, volume 3969 of *Lecture Notes in Computer Science*, pages 120–134. Springer, 2005.

[34] Anne Canteaut and Michaël Trabbia. Improved Fast Correlation Attacks Using Parity-Check Equations of Weight 4 and 5. In Preneel [106], pages 573–588.

[35] Claude Carlet. A Method of Construction of Balanced Functions with Optimum Algebraic Immunity. Technical Report 2006/149, Cryptology ePrint Archive, 2006.

[36] Claude Carlet, Khoongming Khoo, Chu-Wee Lim, and Chuan-Wen Loe. Generalized Correlation Analysis of Vectorial Boolean Functions. In Alex Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 382–398. Springer, 2007.

[37] Philippe Chose, Antoine Joux, and Michel Mitton. Fast Correlation Attacks: An Algorithmic Point of View. In Knudsen [89], pages 209–221.

[38] Don Coppersmith, Hugo Krawczyk, and Yishay Mansour. The Shrinking Generator. In Douglas R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 22–39. Springer, 1993.

[39] Nicolas Courtois. Higher Order Correlation Attacks, XL algorithm and Cryptanalysis of Toyocrypt. Technical Report 2002/087, Cryptology ePrint Archive, 2002.

[40] Nicolas Courtois. Algebraic Attacks on Combiners with Memory and Several Outputs. Technical Report 2003/125, Cryptology ePrint Archive, 2003.

[41] Nicolas Courtois. Fast Algebraic Attacks on Stream Ciphers with Linear Feedback. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 176–194. Springer, 2003.

[42] Nicolas Courtois. Cryptanalysis of Sfinks. In Dongho Won and Seungjoo Kim, editors, *ICISC*, volume 3935 of *Lecture Notes in Computer Science*, pages 261–269. Springer, 2005.

[43] Nicolas Courtois, Blandine Debraize, and Eric Garrido. On Exact Algebraic (Non) Immunity of S-boxes based on Power Functions. Technical Report 2005/203, Cryptology ePrint Archive, 2005.

[44] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In Preneel [106], pages 392–407.

[45] Nicolas Courtois and Willi Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In Eli Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2003.

[46] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In Yuliang Zheng, editor, *ASIACRYPT*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer, 2002.

[47] Paul Crowley. Truncated Differential Cryptanalysis of Five Rounds of Salsa20. Technical Report 2005/073, eSTREAM, ECRYPT Stream Cipher Project, 2005.

[48] Deepak Kumar Dalai, Kishan Chand Gupta, and Subhamoy Maitra. Cryptographically Significant Boolean Functions: Construction and Analysis in Terms of Algebraic Immunity. In Gilbert and Handschuh [60], pages 98–111.

[49] Deepak Kumar Dalai, Kishan Chand Gupta, and Subhamoy Maitra. Notion of Algebraic Immunity and its Evaluation related to Fast Algebraic Attacks. In *BFCA*, pages 107–124, 2006.

[50] Deepak Kumar Dalai, Subhamoy Maitra, and Sumanta Sarkar. Basic Theory in Construction of Boolean Functions with Maximum Possible Annihilator Immunity. *Des. Codes Cryptography*, 40(1):41–58, 2006.

[51] Ivan Damgård. A Design Principle for Hash Functions. In Brassard [30], pages 416–427.

[52] Christophe de Canniére, Joseph Lano, and Bart Preneel. Comments on the Rediscovery of Time Memory Data Tradeoffs. Technical Report 2005/040, eSTREAM, ECRYPT Stream Cipher Project, 2005.

[53] Frédéric Didier. Using Wiedemann's Algorithm to Compute the Immunity Against Algebraic and Fast Algebraic Attacks. In Rana Barua and Tanja Lange, editors, *INDOCRYPT*, volume 4329 of *Lecture Notes in Computer Science*, pages 236–250. Springer, 2006.

[54] ECRYPT Network of Excellence in Cryptology. *SASC - The State of the Art of Stream Ciphers, ECRYPT Workshop, Bochum, Germany, January 31 - February 1, 2007, Workshop Record*, 2007.

[55] Håkan Englund, Thomas Johansson, and Meltem Sönmez Turan. A Framework for Chosen IV Statistical Analysis of Stream Ciphers. In K. Srinathan, C. Pandu Rangan, and Moti Yung, editors, *INDOCRYPT*, volume 4859 of *Lecture Notes in Computer Science*, pages 268–281. Springer, 2007.

[56] Jean-Charles Faugère. A New Efficient Algorithm for Computing Gröbner Bases without Reduction to Zero (F5). In *ISSAC: Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation*, pages 75–83, 2002.

[57] Jean-Charles Faugère and Gwénolé Ars. An Algebraic Cryptanalysis of Nonlinear Filter Generators using Gröbner Bases. Technical Report 4739, INRIA, 2003.

[58] Eric Filiol. A New Statistical Testing for Symmetric Ciphers and Hash Functions. In Robert H. Deng, Sihan Qing, Feng Bao, and Jianying Zhou, editors, *ICICS*, volume 2513 of *Lecture Notes in Computer Science*, pages 342–353. Springer, 2002.

[59] Matthew K. Franklin, editor. *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*. Springer, 2004.

[60] Henri Gilbert and Helena Handschuh, editors. *Fast Software Encryption, 12th International Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers*, volume 3557 of *Lecture Notes in Computer Science*. Springer, 2005.

[61] Jovan Dj. Golic. On the Security of Nonlinear Filter Generators. In Dieter Gollmann, editor, *FSE*, volume 1039 of *Lecture Notes in Computer Science*, pages 173–188. Springer, 1996.

[62] Jovan Dj. Golic. Correlation Analysis of the Shrinking Generator. In Joe Kilian, editor, *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 440–457. Springer, 2001.

[63] Jovan Dj. Golic. Embedding Probabilities for the Alternating Step Generator. *IEEE Transactions on Information Theory*, 51(7):2543–2553, 2005.

[64] Jovan Dj. Golic, Vittorio Bagini, and Guglielmo Morgari. Linear Cryptanalysis of Bluetooth Stream Cipher. In Knudsen [89], pages 238–255.

[65] Jovan Dj. Golic and Renato Menicocci. Edit Distance Correlation Attack on the Alternating Step Generator. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 499–512. Springer, 1997.

[66] Jovan Dj. Golic and Renato Menicocci. Edit Probability Correlation Attack on the Alternating Step Generator. *Sequences and Their Applications - SETA 1998*, pages 213–227, 1998.

[67] Jovan Dj. Golic and Renato Menicocci. Edit Probability Correlation Attacks on Stop/Go Clocked Keystream Generators. *J. Cryptology*, 16(1):41–68, 2003.

[68] Jovan Dj. Golic and Renato Menicocci. Correlation Analysis of the Alternating Step Generator. *Des. Codes Cryptography*, 31(1):51–74, 2004.

[69] Jovan Dj. Golic and Renato Menicocci. Statistical Distinguishers for Irregularly Decimated Linear Recurring Sequences. *IEEE Transactions on Information Theory*, 52(3):1153–1159, 2006.

[70] Mark Goresky and Andrew Klapper. Fibonacci and Galois Representations of Feedback-with-Carry Shift Registers. *IEEE Transactions on Information Theory*, 48(11):2826–2836, 2002.

[71] C. G. Günther. Alternating Step Generators Controlled by De Bruijn Sequences. In David Chaum and Wyn L. Price, editors, *EUROCRYPT*, volume 304 of *Lecture Notes in Computer Science*, pages 5–14. Springer, 1987.

[72] Philip Hawkes and Gregory G. Rose. Rewriting Variables: The Complexity of Fast Algebraic Attacks on Stream Ciphers. In Franklin [59], pages 390–406.

[73] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. A Stream Cipher Proposal: Grain-128. In *IEEE International Symposium on Information Theory*, pages 1614–1618, 2006.

[74] Tor Helleseth, editor. *Advances in Cryptology - EUROCRYPT 1993, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*. Springer, 1994.

[75] Jin Hong. Some Trivial States of Trivium. eSTREAM Discussion Forum, ECRYPT Stream Cipher Project, 2005.

[76] Jin Hong, Dong Hoon Lee, Yongjin Yeom, and Daewan Han. A New Class of Single Cycle T-Functions. In Gilbert and Handschuh [60], pages 68–82.

[77] Jin Hong and Palash Sarkar. New Applications of Time Memory Data Tradeoffs. In Roy [109], pages 353–372.

[78] Shaoquan Jiang and Guang Gong. On Edit Distance Attack to Alternating Step Generator. *Mathematical Properties of Sequences and Other Combinatorial Structures*, pages 85–92, 2003.

[79] Thomas Johansson. Reduced Complexity Correlation Attacks on Two Clock-Controlled Generators. In Kazuo Ohta and Dingyi Pei, editors, *ASIACRYPT*, volume 1514 of *Lecture Notes in Computer Science*, pages 342–356. Springer, 1998.

[80] Fredrik Jönsson. *Some Results on Fast Correlation Attacks.* PhD thesis, Lund University (Sweden), 2002.

[81] Pascal Junod and Serge Vaudenay. Optimal Key Ranking Procedures in a Statistical Cryptanalysis. In Thomas Johansson, editor, *FSE*, volume 2887 of *Lecture Notes in Computer Science*, pages 235–246. Springer, 2003.

[82] John Kelsey, Bruce Schneier, and David Wagner. Mod n Cryptanalysis, with Applications Against RC5P and M6. In Lars R. Knudsen, editor, *FSE*, volume 1636 of *Lecture Notes in Computer Science*, pages 139–155. Springer, 1999.

[83] Andrew Klapper and Mark Goresky. Feedback Shift Registers, 2-Adic Span, and Combiners with Memory. *J. Cryptology*, 10(2):111–147, 1997.

[84] Alexander Klimov. *Applications of T-Functions in Cryptography.* PhD thesis, Weizmann Institute of Science (Israel), 2004.

[85] Alexander Klimov and Adi Shamir. A New Class of Invertible Mappings. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 470–483. Springer, 2002.

[86] Alexander Klimov and Adi Shamir. Cryptographic Applications of T-Functions. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 248–261. Springer, 2003.

[87] Alexander Klimov and Adi Shamir. New Cryptographic Primitives Based on Multiword T-Functions. In Roy and Meier [110], pages 1–15.

[88] Alexander Klimov and Adi Shamir. The TFi Family of Stream Ciphers. Technical report, Weizmann Institute of Science, 2004.

[89] Lars R. Knudsen, editor. *Advances in Cryptology - EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 - May 2, 2002, Proceedings*, volume 2332 of *Lecture Notes in Computer Science*. Springer, 2002.

[90] Pil Joong Lee, editor. *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, volume 3329 of *Lecture Notes in Computer Science*. Springer, 2004.

[91] Bernhard Löhlein. Attacks based on Conditional Correlations against the Nonlinear Filter Generator. Technical Report 2003/020, Cryptology ePrint Archive, 2003.

[92] Yi Lu, Willi Meier, and Serge Vaudenay. The Conditional Correlation Attack: A Practical Attack on Bluetooth Encryption. In Shoup [114], pages 97–117.

[93] James Massey. Shift-Register Synthesis and BCH Decoding. *IEEE Transactions on Information Theory*, 15(1):122–127, 1969.

[94] Mitsuru Matsui. Linear Cryptoanalysis Method for DES Cipher. In Helleseth [74], pages 386–397.

[95] Willi Meier, Enes Pasalic, and Claude Carlet. Algebraic Attacks and Decomposition of Boolean Functions. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 474–491. Springer, 2004.

[96] Willi Meier and Othmar Staffelbach. Fast Correlation Attacks on Certain Stream Ciphers. *J. Cryptology*, 1(3):159–176, 1989.

[97] Willi Meier and Othmar Staffelbach. Analysis of Pseudo Random Sequence Generated by Cellular Automata. In Donald W. Davies, editor, *EUROCRYPT*, volume 547 of *Lecture Notes in Computer Science*, pages 186–199. Springer, 1991.

[98] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[99] Joydip Mitra and Palash Sarkar. Time-Memory Trade-Off Attacks on Multiplications and T-Functions. In Lee [90], pages 468–482.

[100] Dukjae Moon, Daesung Kwon, Daewan Han, Jooyoung Lee, Gwon Ho Ryu, Dong Wook Lee, Yongjin Yeom, and Seongtaek Chee. T-function Based Streamcipher TSC-4. Technical Report 2006/024, eSTREAM, ECRYPT Stream Cipher Project, 2006.

[101] Frédéric Muller and Thomas Peyrin. Personal Communication, 2005.

[102] Frédéric Muller and Thomas Peyrin. Linear Cryptanalysis of the TSC Family of Stream Ciphers. In Roy [109], pages 373–394.

[103] Yassir Nawaz, Kishan Chand Gupta, and Guang Gong. Algebraic Immunity of S-boxes based on Power Mappings: Analysis and Construction. Technical Report 2006/322, Cryptology ePrint Archive, 2006.

[104] ECRYPT Network of Excellence in Cryptology. eSTREAM - The ECRYPT Stream Cipher Project. See `http://www.ecrypt.eu.org/stream`.

[105] Sean O'Neil. Algebraic Structure Defectoscopy. Technical Report 2007/378, Cryptology ePrint Archive, 2006.

[106] Bart Preneel, editor. *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*. Springer, 2000.

[107] Longjiang Qu, Chao Li, and Keqin Feng. A Note on Symmetric Boolean Functions with Maximum Algebraic Immunity in Odd Number of Variables. *IEEE Transactions on Information Theory*, 53(8):2908–2910, 2007.

[108] Sondre Rønjom and Tor Helleseth. A New Attack on the Filter Generator. *IEEE Transactions on Information Theory*, 53(5):1752–1758, 2007.

[109] Bimal K. Roy, editor. *Advances in Cryptology - ASIACRYPT 2005, 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005, Proceedings*, volume 3788 of *Lecture Notes in Computer Science*. Springer, 2005.

[110] Bimal K. Roy and Willi Meier, editors. *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*. Springer, 2004.

[111] Rainer A. Rueppel. *Analysis and Design of Stream Ciphers.* Springer, 1986.

[112] Markku-Juhani Olavi Saarinen. Chosen-IV Statistical Attacks on eSTREAM Ciphers. In Manu Malek, Eduardo Fernández-Medina, and Javier Hernando, editors, *SECRYPT*, pages 260–266. INSTICC Press, 2006.

[113] Claude Shannon. Communication Theory of Secrecy Systems. *Bell System Technical Journal*, 28(4):656715, 1949.

[114] Victor Shoup, editor. *Advances in Cryptology - CRYPTO 2005, 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*. Springer, 2005.

[115] Thomas Siegenthaler. Decrypting a Class of Stream Ciphers Using Ciphertext Only. *IEEE Transactions on Computers*, 34(1):81–85, 1985.

[116] Dirk Stegemann. Extended BDD-Based Cryptanalysis of Keystream Generators. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *Lecture Notes in Computer Science*, pages 17–35. Springer, 2007.

[117] Rajesh Sundaresan. Guessing Under Source Uncertainty. *IEEE Transactions on Information Theory*, 53(1):269–287, 2007.

[118] Yukiyasu Tsunoo, Teruo Saito, Hiroyasu Kubo, Tomoyasu Suzaki, and Hiroki Nakashima. Differential Cryptanalysis of Salsa20/8. In *SASC* [54], pages 39–50.

[119] Serge Vaudenay. *A Classical Introduction to Cryptography: Applications for Communications Security*. Springer, 2006.

[120] Michael Vielhaber. Breaking ONE.FIVIUM by AIDA an Algebraic IV Differential Attack. Technical Report 2007/413, Cryptology ePrint Archive, 2007.

[121] Rade Vuckovac. MAG My Array Generator (A New Strategy for Random Number Generation). Technical Report 2005/014, eSTREAM, ECRYPT Stream Cipher Project, 2005.

[122] David Wagner. A Generalized Birthday Problem. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2002.

[123] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Shoup [114], pages 17–36.

[124] Kencheng Zeng, Chung-Huang Yang, and T. R. N. Rao. On the Linear Consistency Test (LCT) in Cryptanalysis with Applications. In Brassard [30], pages 164–174.

[125] Muxiang Zhang and Agnes Hui Chan. Maximum Correlation Analysis of Nonlinear S-boxes in Stream Ciphers. In Mihir Bellare, editor, *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 501–514. Springer, 2000.

Curriculum Vitae

Personal Details

Family name:	Fischer (birth name Künzli)
First name:	Simon
Nationality:	Swiss
Birthday:	13 November 1978 (Solothurn)

Education

March 2004 - February 2008: PhD candidate in cryptology under supervisions of Prof. Serge Vaudenay, Laboratory of Cryptography and Security (LASEC), Department of Computer & Communication Sciences, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland, and Dr Willi Meier, University of Applied Sciences, Northwestern Switzerland (FHNW).

March 2001 - October 2003: M.Sc. in physics, University of Bern. I graduated with a master thesis entitled "Accretion and Gas Flow" in computational physics (final assessment 6/6).

September 1998 - December 2000: B.Sc. in physics, University of Bern.

August 1993 - January 1998: Matura Typus E (economy), Kantonsschule Solothurn.