Some Words on Cryptanalysis of Stream Ciphers

Maximov, Alexander

2006

Some Words on Cryptanalysis of Stream Ciphers

Alexander Maximov

LUND UNIVERSITY

Ph.D. Thesis, June 16, 2006

Alexander Maximov
Department of Information Technology
Lund University
Box 118
S-221 00 Lund, Sweden
e-mail: movax@it.lth.se
http://www.it.lth.se/

ISBN: 91-7167-039-4
ISRN: LUTEDX/TEIT-06/1035-SE

Abstract

In the world of cryptography, stream ciphers are known as primitives used to ensure privacy over a communication channel. One common way to build a stream cipher is to use a keystream generator to produce a pseudo-random sequence of symbols. In such algorithms, the ciphertext is the sum of the keystream and the plaintext, resembling the one-time pad principal. Although the idea behind stream ciphers is simple, serious investigation of these primitives has started only in the late 20^{th} century. Therefore, cryptanalysis and design of stream ciphers are important.

In recent years, many designs of stream ciphers have been proposed in an effort to find a proper candidate to be chosen as a world standard for data encryption. That potential candidate should be proven good by time and by the results of cryptanalysis. Different methods of analysis, in fact, explain how a stream cipher should be constructed. Thus, techniques for cryptanalysis are also important.

This thesis starts with an overview of cryptography in general, and introduces the reader to modern cryptography. Later, we focus on basic principles of design and analysis of stream ciphers. Since statistical methods are the most important cryptanalysis techniques, they will be described in detail.

The practice of statistical methods reveals several bottlenecks when implementing various analysis algorithms. For example, a common property of a cipher to produce n-bit words instead of just bits makes it more natural to perform a multidimensional analysis of such a design. However, in practice, one often has to truncate the words simply because the tools needed for analysis are missing. We propose a set of algorithms and data structures for multidimensional cryptanalysis when distributions over a large probability space have to be constructed.

This thesis also includes results of cryptanalysis for various cryptographic primitives, such as A5/1, Grain, SNOW 2.0, Scream, Dragon, VMPC, RC4, and RC4A. Most of these results were achieved with the help of intensive use of the proposed tools for cryptanalysis.

To my Family

Contents

Preface

This doctoral thesis is a monograph presenting the results of my four-year research work during my PhD studies at the Department of Information Technology at Lund University in Sweden. It partly includes the results presented in the following papers:

(i) A. MAXIMOV. On Linear Approximation of Modulo Sum. In B. Roy and W. Meier, editors, *Fast Software Encryption – 2004, Delhi, India (acc. rate 39%)*, volume 3017 of *Lecture Notes in Computer Science*, pages 483–484. Springer-Verlag, 2004 [Max04].

(ii) A. MAXIMOV AND T. JOHANSSON. Fast Computation of Large Distributions and Its Cryptographic Applications. In B. Roy, editor, *Advances in Cryptology – ASIACRYPT'05, Chennai, India (acc. rate 16%)*, volume 3788 of *Lecture Notes in Computer Science*, pages 313–332. Springer-Verlag, 2005 [MJ05].

(iii) A. MAXIMOV. Two Linear Distinguishing Attacks on VMPC and RC4A and Weakness of RC4 Family of Stream Ciphers. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption – 2005, Paris, France (acc. rate 30%)*, volume 3557 of *Lecture Notes in Computer Science*, pages 342–358. Springer-Verlag, 2005 [Max05].

(iv) A. MAXIMOV. Cryptanalysis of "Grain" Family of Stream Ciphers. In S. Shieh and S. Jajodia, editors, *Proceedings of the ACM Symposium on Information, Computer and Communications Security – ASIACCS'06, Taipei, Taiwan (acc. rate 17%)*, pages 283–288. ACM Press, 2006 [Max06].

(v) C. BERBAIN, H. GILBERT, AND A. MAXIMOV. Cryptanalysis of Grain. In *Preproceedings of the Fast Software Encryption – 2006, Graz, Austria (acc. rate 25%)*, pages 15–42. To appear in Springer Verlag's *Lecture Notes of Computer Science* [BGM06].

(vi) A. MAXIMOV, T. JOHANSSON, AND S. BABBAGE. An Improved Correlation Attack on A5/1. In H. Handschuh and A. Hasan, editors, *Selected Areas in Cryptography – 2004, Waterloo, Canada (acc. rate 30%)*, volume 3357 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, 2004 [MJB04].

(vii) A. MAXIMOV AND T. JOHANSSON. A Linear Distinguishing Attack on Scream. In I. Oka, M. Ohashi, and A. Miyaji, editors, *Proceedings of the IEEE International Symposium on Information Theory – 2003, Yokohama, Japan*, page 164. Submitted to *IEEE Transaction on Information Theory* [JM03].

(viii) H. ENGLUND AND A. MAXIMOV. Attack the Dragon. In S. Maitra, V. Madhavan, and R. Venkatesan, editors, *Progress in Cryptology – INDOCRYPT'05, Bangalore, India (acc. rate 21%)*, volume 3797 of *Lecture Notes in Computer Science*, pages 130–142. Springer-Verlag, 2005 [EM05].

During my time as a PhD student, I have also co-authored the following papers in the areas of Boolean functions and stream ciphers, which are not included in this thesis:

(i) M. HELL, T. JOHANSSON, A. MAXIMOV, AND W. MEIER. A Stream Cipher Proposal: Grain-128. To appear at the *IEEE International Symposium on Information Theory – 2006, Seattle, USA*.

(ii) A. MAXIMOV. Classes of Plateaued Rotation Symmetric Boolean Functions Under Transformation of Walsh Spectrum. In *Preproceedings of the Workshop on Coding and Cryptography – 2005, Bergen, Norway (acc. rate 21%)*, pages 325–334.

(iii) M. HELL, A. MAXIMOV, AND S. MAITRA. On Efficient Implementation of Search Stategy for RSBFs. In *Proceedings of the International Workshop on Algebraic and Combinatorial Coding Theory – 2004, Kranevo, Bulgaria (acc. rate 55%)*, pages 214–222.

(iv) A. MAXIMOV, M. HELL, AND S. MAITRA. Plateaued Rotation Symmetric Boolean Functions on Odd Number of Variables. In *Proceedings of the Workshop on Boolean Functions: Cryptography and Applications – 2005, Rouen, France*.

Acknowledgments

My Family is my fortress! I thank my Family for their fundamental support of my research activity. Whatever was happening, I knew there was a place where I could always come and be understood.

I infinitely appreciate the great opportunity that my supervisor, Professor Thomas Johansson, gave me as a scientist. In the beginning of my research I was like a blind puppy, and did not know where to go in the world of cryptography. Thomas was the person who showed me the spirit of creativity in science, and made me creative myself.

Obviously, no results could have been achieved without such a warm and cosy working surroundings at the Department of Information Technology. My great acknowledgment to my closest neighbours: to Martin Hell, for his patience in listening to my wrong ideas, and to Håkan Englund, for always being a good friend. I feel great respect for all the other PhD students, Maja, Marcus, Koraljka, Victor, Dmitry, Tomas, and others, partly for helping me with the proof reading of this thesis, but even more for being helpful, friendly, talkative and creative in fun events. Special thanks to the designer of the department's cosy couch in the kitchen; it has always been a great place to stretch out and relax. Thanks to Laila Lembke who never saw me sleeping there with books about cryptography in my hands. I also thank everyone else working at the IT department!

Perhaps I would never be able to feel happy without my great Russian-speaking friends, mostly also PhD students in physics, chemistry, microbiology, economics, maths, genetics, and other sciences. My limitless gratitude to Anton, for always having unexpected ideas for various activities, for telling me about bio-sensors and genetics, for playing a guitar and for talking about the meaning of life late at night. My infinite thanks to Yulia, Dmitry, Oleg, Svetlana, Ksenia, Ivan, Vaida, Valeria, Yurii, and many others, for keeping us all together, for the club of science, for skiing in mountains, for movies and cake parties — simply for being my best friends in Lund.

Thank you!!!

Lund, June 2006
Alexander Maximov

This research was partly supported by the Swedish Research Council under Grant VR 621-2001-2149; the European Commission through the IST Program under Contract IST-2002-507932 ECRYPT; the Graduate School in Personal Computing and Communication PCC++.

1

Introduction

"*Scytale*", an ancient device for encryption.

Human is a social creature! A need for communication has existed among human beings for thousands of years. Speech has always been an important peculiarity of human society, one that made us significantly different from other animals. In dinosaur times, when living in a cave one would need to tell the others where the meat is hidden, while that information had to be kept secret from another tribe, living in a neighbouring cave. That era can be regarded as the beginning of secrecy, the root of modern cryptography.

The creation of writing made the secrecy of communication even more important. Indeed, the lifetime of a written message is longer than that of a spoken one. Around 1900 B.C., ancient Egyptians started to modify symbols to ensure the privacy of a message. Later, in 600-500 B.C., ancient Jews created a cryptosystem, "Atbash", which is now known as a *substitution cipher*. The idea is simple: every letter is substituted with another letter from the same alphabet. One of the best known ancient encryption methods is the *Caesar cipher* from around 100-44 B.C., where every letter is substituted by the third next letter of the alphabet. The dictator of the Roman Empire, Gaius Julius Caesar, used this encryption method to protect his messages and orders from curious eyes. An interesting fact is that in the famous Indian book "Kama-Sutra", cryptography is mentioned as one of 64 arts, a must for study...

While some people were hiding their secrets, other curious creatures were breaking the codes. This "game" perhaps started with the Chinese manuscript "*Art of War*", written by Sunzi around 500 B.C. Besides the descriptions of tactics and strategies of a war, the book also gives the basic methods of information analysis. The science of analysis of codes is called *cryptanalysis*.

Evaluation of cryptography in the last 2000 years was significantly faster than before. In the XV century, Italian mathematician Leo Batista Alberti made the first mathematical model of cryptography. He also created the first machine for encryption, and future cryptographic systems were based on his ideas, before modern computers were invented. In the XVIII century, Thomas Jefferson, the third president of the USA and a scientist, invented a device in a cylindrical form for encryption. That mechanical device allowed application of tens of different encryption methods, and it was used until the second World War. In the XVIII century, the English spy agency started to use invisible ink based on milk. The information is revealed when the paper is heated. This kind of encryption was also used by Lenin–the father of the Russian Red Revolution in 1917. While in a prison, he was writing his messages by milk between texts in books, and then his revolutionary comrades could read them.

Figure 1.1: Thomas Jefferson, and his cylinders for encryption.

The history of modern cryptography perhaps began in the 1920s, when a group of German scientists created the device "*Enigma*"–this was, in fact, the first specialized computer for encryption. Three years later the mechanism of Enigma was discovered by a group of British scientists, secretly formed on purpose for this investigation. A few years later, Enigma was adopted for encryption for almost all correspondence of the German army, navy, air force, Gestapo, and other official departments. That device was widely used during the World War II. The fundamental break of the Enigma system was made in Warsaw, Poland, in 1932 by Marian Rejewski. This and

Figure 1.2: The encryption device "*Enigma*", used in World War II.

other results allowed breaking Enigma and cut the War by one-two years, and saved millions of lives. This historical example clearly shows the importance of cryptography and cryptanalysis in the XXI century.

1.1 Modern Cryptography in the Digital World

Nowadays, information technology and communication are a deeply integrated part of our life. People use mobile phones, Internet, banking systems, and other services. More generally, communication can exist wherever a *channel* can be established. A channel can be regarded as an information provider; it can be a wire, air, light, waves, etc. However, a channel can be *secure* or *insecure*, depending on who can "listen" to it. If the actual information going through the channel can be understood only by intended recipients, then the channel is *secure*. Otherwise, if someone else can understand the actual information, the channel is called *insecure*.

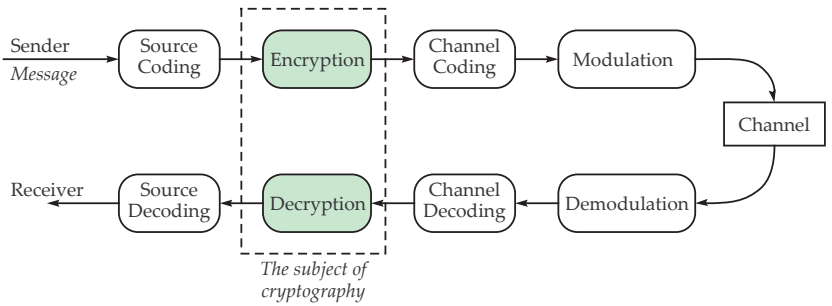


Figure 1.3: Place of cryptography in a communication system.

A typical path for information going through the channel is shown in Figure 1.3 and has four main stages:

1. *Source Coding* removes the redundancy from the source such that the information is *compressed*. Typical examples of source encoders are compression algorithms such as ZIP, ARJ, RAR. Source coding is also used to compress media in formats JPEG, AVI, and other applications. Decoding reverses the compression of the information.

2. After the redundancy is removed, *encryption* is performed to ensure the privacy of the communication. The encryption method, a *cipher*, takes the input *plaintext* and produces an encrypted message called *ciphertext*. Usually, the sizes of the plaintext and the ciphertext are the same. Encryption and decryption methods are typical subjects in cryptography.

3. In *Channel coding* some portion of redundancy is added to the input data stream for detecting and correcting digital errors that occur during the transmission. Error correcting codes are used, for example, in media compact discs, mobile communication, and other applications.

4. *Modulation* is the process wherein a radio frequency or light wave's amplitude, frequency, or phase is changed in order to transmit the information through the channel.

The role of cryptography is to make an insecure channel secure. The information transmitted through the channel is first *encrypted*, so that it can be understood only by those who are supposed to know a secret algorithm or a procedure that reverses the encryption. This procedure is also called *decryption*.

Any *cryptographic system* uses *primitives* to provide a set of *services*.

1.2 Cryptographic Services

Cryptography can be applied in many ways, providing different services. Below we describe the most important services that are used in real applications. They are: *data confidentiality, user authentication, data integrity,* and *non-repudiation of origin*.

Data Confidentiality

This service is, perhaps, the oldest and the best known. Every person has some sensitive information that he would not like to broadcast to everyone. For example, the number of one's credit card is better kept far away from

curious eyes. Confidentiality is very important in hospitals, where information in general is very sensitive. Banks have to keep records of their clients secure as well, since this information can be used by an intruder.

Data confidentiality guarantees access to the information *only* to authorized people.

User Authentication

When logging onto a computer one has to enter the login name and the password to make the operation system recognize him as a user. Or, when using an ATM machine to withdraw money, the client is usually asked to type his 4-digit pin code. In another situation, when two people want to establish communication, they often first need to prove to each other that they are indeed who they claim to be. All these situations require a special mechanism of *authentication*.

Authentication can be done in many ways. The person can identify himself if he *has* something (e.g., ID card, token), *knows* something (e.g., password, pin code), or *is* an identifier (e.g., fingerprints, eyes, other biometrics). The first two methods are widely used in common applications. Despite the fact that biometrics are very good for authentication, they are still used quite seldomly, since devices for biometric authentication are expensive and complicated.

Data Integrity

The data integrity service guarantees that the information sent through the channel is not modified. This service itself is useless unless the recipient knows who the sender is. Therefore, data integrity is usually combined with the *data origin authentication*, which guarantees that the person who claims his authority is really the sender of the message.

In a digital communication system one should be alerted to an intruder who listens to the channel and modifies the information that is going through (*an active attack*). For example, e-mail communication is usually not secure; one could generate an e-mail to some person A that looks as though it was sent from another person B.

Cryptography solves these problems as well. For the data integrity service *hash functions* and / or message authentication, codes as well as digital signatures can be used. Digital signatures additionally guarantee *non-repudiation of origin*, a very important concept in cryptology. It means that the author of the message cannot later deny his responsibility.

1.3 Cryptographic Primitives

Cryptographic primitives are those building blocks that are used to provide cryptographic services. In Figure 1.4 the taxonomy of cryptographic primitives is shown. Primitives can be divided into three categories: *unkeyed primitives*, *symmetric key primitives*, and *public key primitives*. In this thesis, we focus in detail on *symmetric key stream ciphers*, in particular on *stream ciphers*. They will be defined and described in the next section. Below we briefly describe security primitives.

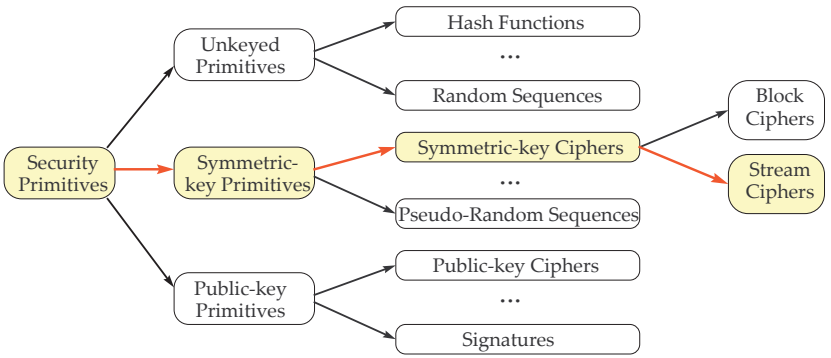


Figure 1.4: Taxonomy of cryptographic primitives.

Unkeyed Primitives

Unkeyed primitives are used to support such services as authentication and data integrity. These primitives are, for example, *one-way permutations*, *hash functions*, and others.

Hash functions or *compression functions* have received a lot of attention in modern cryptography. It takes a long input sequence of a message, and produces a short fixed length string, which is usually referred to as a *message digest*, *checksum* or a *digital fingerprint*. The term *hash* apparently comes from the physical term to *chop and mix*. Knuth noted that Hans Peter Luhn from IBM appears to be the first person to use this term, in January 1953. Indeed, in most hash algorithms the message is first "chopped" into words, and then "mixed" with the previous ones.

Hash functions must satisfy certain properties.

- *One-Way (OW) property* means that given a hash value it is difficult to

find a corresponding message.

- *Weak Collision Resistance (WCR):* it should be hard to find two messages with the same hash value.
- *Strong Collision Resistance (SCR):* given one message, it should be difficult to find another message with the same hash value.

One can note that there are relations among these criteria. For example, if a hash function is not SCR, then it is, consequently, not WCR.

It is hard to overestimate the importance of hash functions for cryptography. They provide services such as integrity and digital signing. The most widely used hash functions are, perhaps, MD5 [KR95] and SHA-1 [Rob94]. Recent results on cryptanalysis of hash functions MD4/5 [YWZW05], SHA-0 [WYY05b] and SHA-1 [WYY05a] reveal a weakness in the design of these functions. It has been shown that the strongest known compression function SHA-1 can be "broken" (a collision can be found), in around 2^{63} operations. These results gave hash functions top priority in discussions throughout the cryptographic community.

Symmetric Key Cryptography (SKC)

SKCs are used to provide services like data confidentiality and non-repudiation of origin. This class of primitives contains such primitives as block ciphers and stream ciphers – the main object of this thesis.

Public Key Cryptography (PKC)

PKC uses two keys, a *public key* and a *private key*, to implement an encryption algorithm that does not require a trusted third party. Thus, *public key primitives* can be used for *key exchange protocols*, *digital signing*, and other purposes.

The breakthrough in the concept of the PKC principles was achieved by Whitfield Diffie and Martin Hellman from Stanford University. Their ground-breaking paper "*New Directions in Cryptography*" was published in November 1976 in *IEEE Transactions on Information Theory* [DH76]. Their paper describes the key concepts of the PKC, including how to produce digital signatures, and gives some sample algorithms for implementation. This paper changed the world of cryptography research, which had been somewhat restrained up to that point by real and perceived government restrictions, and initiated dozens of researchers around the world to work on practical implementations of public key cryptography algorithms.

Two years later, Adi Shamir, Ron Rivest, and Leonard Adleman, from the Massachusetts Institute of Technology, published their first public key

encryption algorithm called RSA [RSA78]. It first appeared in the magazine *Scientific American* in the widely read column *Mathematical Games*. The article also invited people to include the RSA method in their e-mails to ensure privacy. The US National Security Agency (NSA) realized the power of the algorithm and feared that non-government elements could use it. NSA sought to stop distribution of the report, but could not provide legal bases for their demand. In February 1978, the university bravely published the RSA in the journal *Communications of the ACM*, and PKC went out to the world... In 1982, the inventors of RSA formed a company to market their PKC algorithm. Nowadays, PKC is one of the main parts of cryptography, widely used and investigated.

1.4 Recent History of Cryptography

For encryption purposes there exist, basically, two types of primitives, *block* and *stream* ciphers. Block ciphers are classical primitives that have been studied for years. Collected design techniques and cryptanalysis of block ciphers allowed to develop such a standard for encryption as Rijndael (AES). This cipher is widely accepted, and it has strong resistance against various kinds of attacks.

On the other hand, although the idea of stream ciphers appeared long ago, the open study and investigation of these primitives began only about 20 years ago. It is widely believed that stream ciphers can be smaller and much faster than block ciphers when implemented. Unfortunately, we still do not have enough knowledge about the design and cryptanalysis of stream ciphers.

Additive stream ciphers form a family of cryptographic primitives that has many properties suitable for use in various applications, including telecommunications. In a binary additive stream cipher, the keystream, the plaintext, and the ciphertext are sequences of binary digits. The keystream is generated from a keystream generator; it takes a secret key and the initial value (IV) as a seed, and produces a long pseudo-random sequence. In an additive stream cipher, the keystream depends neither on the plaintext nor on the ciphertext. The ciphertext is usually generated by bit-wise addition (modulo 2) of the keystream and the plaintext. Since the secret key is shared by the transmitter and the receiver, the receiver can decrypt, and obtain the message sequence by adding the keystream to the ciphertext. In general, the stream cipher does not have to be bit-oriented, but can also produce a number of bits (a word) at a time. This is the current trend in designing high-speed stream ciphers.

Thus, designing *high-speed* stream ciphers has been an important cryptographic topic for the last few years, motivated by the widespread belief that

such stream ciphers can be considerably faster than block ciphers in software, and possibly also simpler to implement in hardware. A fast stream cipher is obtained by designing a pseudo-random generator depending on a secret seed (key and IV), to produce the output stream as fast as possible.

Quite a few proposals for stream ciphers have been suggested. However, many of them suffer from a number of small cryptographic weaknesses. Well known and frequently used stream ciphers like A5/1 [BGW99] used in GSM, RC4 [Sma03], and E_0 [Blu03] used in Bluetooth are all susceptible to different attacks [BSW00, MJB04, EJ01, Gol97b, Gol99, Gol97a, FL01]. More recent proposals, like SOBER t16 [HR00a], SOBER t32 [HR00b], and SNOW 1.0 [EJ00], have been shown to be susceptible to so-called distinguishing attacks [CHJ02, EJ02a] that can distinguish the generated pseudo-random sequences from truly random sequences. In fact, in a recent European project called NESSIE [NES99] a call for primitives (including stream ciphers) was announced in 2000. After two phases of evaluation, none of the stream ciphers submitted were found to be completely free from security flaws. During this project, new techniques for cryptanalysis on stream ciphers were found, and many new proposals were broken.

After the NESSIE project was finished, we have seen additional proposals of stream ciphers, and some of them have not yet been thoroughly analysed. To mention a few, there are SNOW 2.0 [EJ02b], MUGI [WFY+02], Scream [HCJ02], Turing [RH03], Rabbit [BVP+03], Helix [FWS+03], VMPC [Zol04], and RC4A [PP04]. Most of these ciphers are significantly faster than for example AES, and if we could gain confidence in their security they would be very interesting alternatives for encryption. The primitives SNOW 2.0, Scream, VMPC, and RC4A are analysed in this thesis in Sections 4.4, 7, and 6.

The situation clearly requires the cryptographic community to devote more attention to the design and analysis of stream ciphers. For this reason, the European project eSTREAM (ECRYPT) [ECR05] announced a call for stream cipher primitives. 35 proposals were submitted to the project by April 2005, and most of them were presented at the workshop SKEW 2005 [SKE05] in May.

Cryptanalysis techniques discovered during the NESSIE project have made it possible to strengthen new designs greatly, and attacking new algorithms has become more difficult. There are many interesting submissions to eSTREAM, such as Dragon [CHM+05], Grain (V.0) [HJM05a], TRIVIUM [CP05], Roo [BS05], MOSQUITO [DK05], Phelix [WSLM05], and others. After around half a year of analysis, a new workshop (SASC 2006) was held in February 2006, where all proposals were submitted to more detailed discussion. In this thesis we analyse two of the candidates, Grain V.0 and Dragon, in Chapters 8 and 9, respectively.

All this development shows that work on stream ciphers is intense. We believe that this will result in a few good stream ciphers for future standardization. The next problem facing the crypto-community is the design and analysis of *hash functions*, but this is another story for the near future.

1.5 Thesis Outline

This section contains the outline of the thesis, as well as the material sources for each chapter. Next, we take a brief look at the contents.

Chapter 1 (*Introduction*) contains an overview of cryptography in general, where the place and role of cryptography in a communication system is given as well.

Chapter 2 (*Symmetric Primitives — Introduction to Stream Ciphers*) gives standard definitions and notations that are frequently used in cryptography. We introduce symmetric primitives, block ciphers and, in more detail, stream ciphers. Basic design principles are discussed, followed by a few examples.

Chapter 3 (*Techniques for Cryptanalysis*) provides an overview of different cryptanalysis methods in cryptography. We focus on linear cryptanalysis related to stream ciphers, and, more specifically, on correlation and distinguishing attacks when different scenarios are possible.

The content of the first three chapters is partly influenced by various Internet sources, like Wikipedia [Wik06b]; from general books on cryptography [Sma03, Sti95, Sch96], on probability theory and statistics [Dra67, CB90, Rao73, Gut95], on number and finite field theories [LeV77, Fra94], on information theory [CT91]; from PhD theses [Jen80, Jön02, Pas03, Dod03, Ekd03, Gup04]. The section devoted to distinguishing attacks is partly based on the paper [JM03].

Chapter 4 (*Tools for Cryptanalysis*) introduces a class of n bit functions, so-called pseudo-linear functions, which appear to be very useful for cryptanalysis evaluation. Efficient algorithms and data structures for multidimensional cryptanalysis are proposed; they enabled us to achieve many other results in cryptanalysis. One such an example is also given at the end of the chapter. This material is also presented in the paper [MJ05].

Chapter 5 (*Cryptanalysis of A5/1*) proposes a key-recovering attack on the stream cipher A5/1, which is used in GSM communication to encrypt conversations. This is a correlation attack originally published in the paper [MJB04].

Chapter 6 (*Cryptanalysis of VMPC and RC4A. Weakness of RC4-like Ciphers*) points out the general weakness in the structure of RC4-like stream ciphers. RC4 is a stream cipher that is widely used in, for example, Internet communication. This chapter also analyses two recent stream ciphers from this

family, namely VMPC and RC4A, and proposes two distinguishing attacks on them. The content of the chapter is derived from the paper [Max05].

Chapter 7 (*Cryptanalysis of "Scream"*) presents a linear distinguishing attack on the stream cipher Scream, which was developed by a group of researchers at IBM. These results are based on the paper [JM03].

Chapter 8 (*Cryptanalysis of the "Grain" Family of Stream Ciphers*) details the analysis of the stream cipher Grain. This cipher is a candidate to the eSTREAM project, and it is considered to be an interesting choice for the final portfolio. This analysis motivated the designers to tweak the design. Therefore, another version, Grain V.1., has appeared. That new version is immune to the discovered attack and appears to be quite strong against linear analysis. The results of this chapter are published in [Max06,BGM06].

Chapter 9 (*Statistical Analysis of "Dragon"*) presents one more analysis of another candidate to eSTREAM, the stream cipher Dragon. It reveals a statistical weakness of the keystream produced by this primitive. It proposes a distinguisher, which, however, has a very small advantage due to the resynchronisation policy of the design. However, the internal state of Dragon is huge, and the identified statistical weakness reveals some structural mistakes in the design of the cipher. The results of this chapter are published in the paper [EM05].

Finally, in Chapter 10 (*Concluding Remarks*) we highlight the results and draw some personal conclusions from the work presented in this thesis.

2

Symmetric Primitives — Introduction to Stream Ciphers

"I put my heart and my soul into my work,
and have lost my mind in the process"

Vincent Van Gogh

Symmetric primitives are important units in any cryptographic system: they are the tools for encryption of a data stream. Symmetric primitives are usually divided into *block ciphers* and *stream ciphers*, and both classes are important. In this chapter we describe symmetric primitives and their basics, and also give standard notations and definitions. We mainly focus on design principles of stream ciphers, although an introduction to block ciphers is also provided.

Let us start this chapter with basic definitions and notations. In our communication model we have three parties:

- **A***lice*, *the sender*, she wants to send some message to **B***ob*.
- **B***ob*, *the receiver*, he receives messages from **A***lice*.
- **E***ve*, *an intruder* or *cryptanalytic*, she wants to read messages sent by **A***lice* to **B***ob*.

REMARK: For notation purpose, if $x_1, x_2, \ldots, x_n$ are n variables from some alphabet $\mathcal{A}$, their sequence (or vector) is denoted in bold as $\mathbf{x} = (x_1, \ldots, x_n)$. When the number of components n is important, we denote it by $\mathbf{x}^n$. When $X_1, X_2, \ldots, X_n$ are variables from an extended alphabet $\mathcal{A}^b$, then this sequence is denoted as $\mathbf{X}^n$. Single symbols of each component X_i are addressed as $X_i = (X_i[1], \ldots, X_i[b])$. Short vectors with a fixed length, such as a key and an IV, are denoted by a capital letter Y, and its components are $Y = (y_1, y_2, \ldots)$. In some chapters this general notation rule can vary for a number of reasons.

We introduce more mathematical notions as follows.

Definition 2.1 (Alphabet, Plaintext, Ciphertext, Key):

- *Alphabet*, $\mathcal{A}$, is a set of symbols used for encryption. In most cases the alphabet consists of b bits words $\mathcal{A} = \{0,1\}^b$, for some $b = 1, 2, \ldots$.
- *Message* or *Plaintext*, $\mathbf{m}$, is the information to be sent through a secure channel from **A***lice* to **B***ob*. We often consider the plaintext as a sequence of n symbols from an alphabet $\mathcal{A}$, i.e.,

$$\mathbf{m}^n = m_1, m_2, \ldots, m_n, \quad m_i \in \mathcal{A}, i = 1, \ldots, n. \tag{2.1}$$

  Let the set of all possible plaintexts be denoted by $\mathcal{M}$.
- *Ciphertext*, $\mathbf{c}$, is the encrypted information sent through an insecure channel. The ciphertext is the result of an *encryption* algorithm, to be described later. The ciphertext is usually also a sequence of n letters from the alphabet $\mathcal{A}$ as follows.

$$\mathbf{c}^n = c_1, c_2, \ldots, c_n, \quad c_i \in \mathcal{A}, i = 1, \ldots, n. \tag{2.2}$$

  It is not always the case that the lengths and alphabets of the plaintext and the ciphertext are the same, but in most cases they are. Let us also denote the set of all possible ciphertexts as $\mathcal{C}$.
- *Key*, K, is the secret key that **A***lice* and **B***ob* exchange via a secure channel (or just secretly) in advance, before the communication is started. This secret key chooses a map from the set of plaintexts to the set of ciphertexts out of a set of maximum $|\mathcal{K}|$ possible maps, where $\mathcal{K}$ is the set of all possible keys K. We usually consider the key to be a sequence of l symbols from $\mathcal{A}$ and represent it as

$$K = k_1, k_2, \ldots, k_l, \quad k_i \in \mathcal{A}, i = 1, \ldots, l. \tag{2.3}$$

□

Symmetric primitives are algorithms that use the same key on both the sender and the receiver sides. Symmetric *ciphers* are ones of such algorithms. When given the plaintext, they produce the ciphertext in accordance to the secret key. On the receiver side, they are used to decrypt the ciphertext back into the plaintext.

Definition 2.2 (Encryption, Decryption, Cipher):

- *Encryption*, $E_{K_e}(M) : \mathcal{M} \times \mathcal{K} \rightarrow \mathcal{C}$, is a function or algorithm that receives the plaintext $\mathbf{m}$ and produces the ciphertext $\mathbf{c}$, according to the secret key K_e.
- *Decryption*, $D_{K_d}(M) : \mathcal{C} \times \mathcal{K} \rightarrow \mathcal{M}$, is an inverse function that receives the ciphertext $\mathbf{c}$ and recovers the plaintext $\mathbf{m}$, according to the secret key K_d.
- *Cipher*, is a pair of two functions, Cipher$= (E, D)$, such that for any plaintext $\mathbf{m} \in \mathcal{M}$, any encryption key $K_e \in \mathcal{K}$ and the corresponding decryption key $K_d \in \mathcal{K}$, we have

$$D_{K_d}(E_{K_e}(\mathbf{m})) = \mathbf{m}. \tag{2.4}$$

□

In a communication scheme where *symmetric primitives* are used (see Figure 2.1), both the sender and the receiver use the same *key* K,

$$K_e = K_d = K, \tag{2.5}$$

which is sent via some *secure channel* in advance before the communication has begun [1]. Therefore, these primitives are called *symmetric*.

It is not always possible to have a secure channel for key distribution. To overcome this problem, there exist various *key exchange protocols*, to set up a symmetric key. These protocols and methods are slow and likely to be used only for sending short messages, such as a key, and should not be used for encryption of large messages, e.g., large files, pictures, etc.

2.1 Definitions of Block and Stream Ciphers

Following Figure 2.2, an *encryption scheme* is a combination of a *symmetric primitive* and a *mode of operation*. Symmetric primitives are classified as *block*

[1] The realization of the secure channel in practice can be various. Many systems that use cryptography (GSM, banking cards, etc.) have secret key already inserted into devices during fabrication. Therefore, the *key distribution via a secure channel* for these devices is the process of writing the key on a trusted environment. Some of these systems also have another key, *administrative*, that is used for updating the secret key, which is then used for data encryption. Such separation prevents key-guessing attacks with chosen data.

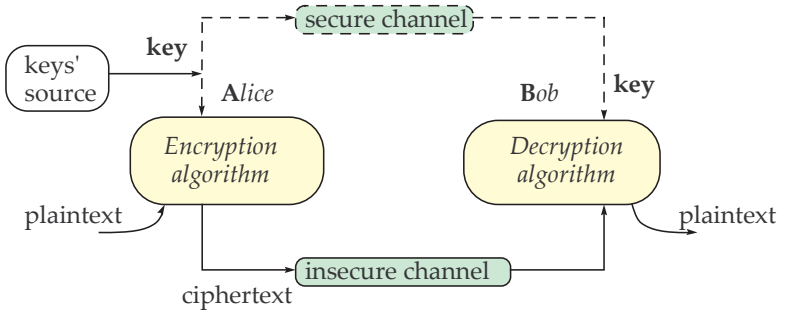


Figure 2.1: Communication scheme with a symmetric primitive.

ciphers and *stream ciphers*. Moreover, with some different *modes of operation* block ciphers can work as stream ciphers.

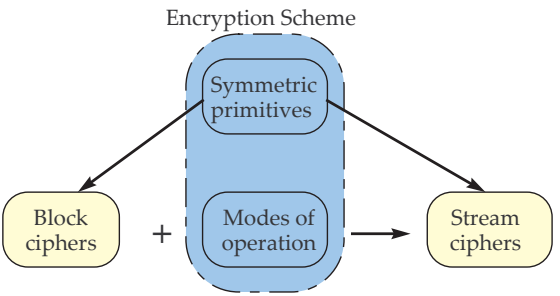


Figure 2.2: Components of an encryption scheme.

The basic key in defining block and stream ciphers is the type of operations used in the algorithm. We say that a cipher operates with symbols from alphabet $\mathcal{A}$ when all ground operations of the algorithm work with the symbols as whole. If this is not true, then there exists the smallest integer $c > 1$, such that $\mathcal{A} = \hat{\mathcal{A}}^c$, for some smaller set $\hat{\mathcal{A}}$, and the cipher can be represented as an algorithm that operates with symbols from $\hat{\mathcal{A}}$.

In the crypto community, the definition of these two classes is not yet precisely specified, and in the literature, different definitions can be found. Here we suggest our vision of how these two classes of ciphers can be distinguished.

Definition 2.3 (Block Cipher): A *block cipher* is a memory-less key-dependent permutation algorithm that takes *one* symbol from an alphabet $\mathcal{A}^b$, and outputs another symbol from the same alphabet. The decryption procedure is the inverse permutation algorithm. □

Although the ground operations in a block cipher work over the alphabet $\mathcal{A}$, a block cipher itself is a black box that works over an extension alphabet $\mathcal{A}^b$, which is a constitution of b symbols from $\mathcal{A}$. In *Advanced Encryption Standard (AES)* the length of the block is 128 bits, and the minimal key size is 128 bits. All operations are byte-oriented, and thus the input alphabet for AES is $\mathcal{A}^b = \mathbb{F}_{2^8}^{16}$. This means that **A***lice* can choose one out of 2^{128} permutations (the number of possible keys $|\mathcal{K}|$), although the total number of possible permutations is $2^{128}!$, since the block size is 128 bits.

It is more difficult to define a stream cipher. For example, Wikipedia gives us the common definition as follows.

Definition 2.4 (Stream Cipher – Common Definition): A *stream cipher* is a symmetric cipher in which the plaintext digits are encrypted one at a time, and in which the transformation of successive digits varies during the encryption. □

This definition is good, but it does not locate a precise borderline between block and stream ciphers. One can consider a block cipher as a stream cipher, for which a *digit* is from $\mathcal{A}^b$. Then, working in different modes of operation, a symbol transformation will be different each time, and the definition is satisfied. Do we then say that a construction of any stream cipher is a block cipher working in a mode of operation, or is it a wider concept? Are block and stream ciphers different, or is one a subclass of another? Should a stream cipher have memory or not?

Let us now give *our interpretation* for the definition of *a stream cipher*.

Definition 2.5 (Stream Cipher – Alternative Definition): A *stream cipher* is a key-dependent algorithm *with internal memory* that receives symbols of a message $\mathbf{m}$ one-by-one over the alphabet $\mathcal{A}$, and in parallel produces the ciphertext $\mathbf{c}$ over the same alphabet, perhaps, with some delay. □

For stream ciphers the alphabet $\mathcal{A}$ typically consists of $1/8/16/32/64$ bit numbers. In hardware, binary alphabets are most common, whereas stream ciphers in software are usually either *byte* (8 bits), or *word* (32 bits) oriented.

In our definition, block and stream ciphers are separate classes of primitives. One may ask whether a block cipher, working in a specific mode of operation, be regarded as a stream cipher? According to our definition above the answer is “no”; it will still remain “a block cipher working in a specific mode of operation”, since a stream cipher must have an internal state.

In the rest of this chapter we take a closer look at the design and criteria of symmetric primitives.

2.2 Designing a Perfect Cipher

From the time cryptography was formed as a science, one task has always been to find a perfect, unbreakable, cipher.

2.2.1 Unbreakable Cipher

The principle of *unconditional security* can be stated as follows.

Definition 2.6 (Perfect Secrecy): A cipher has *perfect secrecy* if

$$\Pr\{(\mathbf{m},\mathbf{c})\} = \Pr\{\mathbf{m}\}\cdot\Pr\{\mathbf{c}\}, \qquad \forall \mathbf{m}\in\mathcal{M}, \forall \mathbf{c}\in\mathcal{C}. \tag{2.6}$$

□

There is one well known perfectly secure cipher, which is called the *Vernam cipher* or the *one-time pad cipher (OTP)*. In the fundamental paper of 1949 [Sha49], Shannon showed that this cipher is unconditionally secure.

Definition 2.7 (One-Time Pad): Let the plaintext $m_1, m_2, \ldots, m_n \in \mathcal{A}$ contain n symbols from some alphabet $\mathcal{A}$. The secret key $k_1, k_2, \ldots, k_n \in \mathcal{A}$ must be of length $l = n$. The ciphertext $c_1, c_2, \ldots, c_n$ is then generated as

$$c_i = m_i + k_i, \qquad \text{for all } i = 1, \ldots, n, \tag{2.7}$$

where $+$ is a group operation over $\mathcal{A}$. □

In the following example we show how this cipher works.

EXAMPLE *2.1 (One-Time Pad (OTP)):* Let the alphabet be $\mathcal{A} = \{\mathtt{A}, \mathtt{B}, \mathtt{C}, \ldots, \mathtt{Z}\}$. Let the plaintext and the key be

$$\mathbf{m} = \mathtt{ONETIMEPAD},$$
$$K = \mathtt{SECRETKEYS}.$$

Let us introduce a map $\delta : \mathcal{A} \to \{0, 1, \ldots, 25\}$, which assigns a number from 0 to 25 to each of the letters, e.g., $\delta(\mathtt{A}) = 0$, $\delta(\mathtt{B}) = 1$, ... The '+' operation over $\mathcal{A}$ can be interpreted as

$$a + b = \delta^{-1}(\delta(a) + \delta(b) \mod 26), \quad \forall a, b \in \mathcal{A}. \tag{2.8}$$

Then, the ciphertext will be produced as follows.

$$\begin{array}{lcl}
\mathbf{m} = \mathtt{ONETIMEPAD} & \xrightarrow{\delta} & (14\ 13\ \ 4\ 19\ \ 8\ 12\ \ 4\ 15\ \ 0\ \ 3) \\
K = \mathtt{SECRETKEYS} & & (18\ \ 4\ \ 2\ 17\ \ 4\ 19\ 10\ \ 4\ 24\ 18) \\
 & & \qquad\qquad +\downarrow \\
\mathbf{c} = \mathtt{GRGKMFOTYV} & \xleftarrow{\delta^{-1}} & (\ 6\ 17\ \ 6\ 10\ 12\ \ 5\ 14\ 19\ 24\ 21).
\end{array}$$

□

The Vernam cipher has perfect secrecy only if the following requirements are satisfied.

- The secret key is chosen completely at random.
- The lengths of the secret key and the plaintext are the same.
- One key is used only once.

The "one-time pad" cipher is still used, for example, in the Russian navy. They have secret books with numbers, which are delivered to every ship in advance. When they want to encrypt something, they choose a page from the book, and use the numbers to pad the plaintext. The number of the page is broadcasted openly, and that page is never used again. If one of these books is lost or stolen, new books are printed and distributed. The weakest point here is the mechanism of distribution and storage of these books.

2.2.2 Alternative Principles for Designs

The one-time pad cipher is welcome and good. However, if the message is long, the required key is really large. It raises problems for the key exchange protocol, since the effort to establish a secure communication channel will require much more than the communication itself. Therefore, there must be another way of designing primitives. The solution is found in the principle of *conditional security*, which can be defined as follows.

Definition 2.8 (Conditional Security): A system is called *conditionally secure* if it can be broken in principle, but this requires more computing power than a realistic adversary would have. In this case its security is measured via *complexity theory*. □

Adopting this definition to cryptographic ciphers, we say that a cipher is called *conditionally secure* if the plaintext can be recovered from the ciphertext, but with the time and effort being no less than is required for an *exhaustive key search attack*, when every possible key from $\mathcal{K}$ is tested.

This principle means that the key can still be short, but long enough to make the *exhaustive key search attack* impossible for an intruder; i.e., to perform this attack will require an impossible combination of time and hardware resources.

2.2.3 Confusion and Diffusion

In his 1949 paper, Shannon identified two types of operations that a secure cipher should make use of:

- *Confusion* "refers to making the relationship between the key and the ciphertext as complex and involved as possible", – according to Shannon's original definition. The *substitution* types of operations are excellent examples of the term "confusion", such as S-boxes.

- *Diffusion* means the level of dependency between the input and output bits of a cipher. If a cipher has a good diffusion property, then flipping one bit of the input changes every bit of the output with a probability close to 1/2. *Permutation* or *transposition* operations are techniques for diffusion.

Classical building blocks are *substitution-permutation networks (SPN)* that are used in block ciphers (for example, in *Feistel ciphers*). To support principles of confusion and diffusion, a block cipher (or an SPN) consists of multiple rounds of similar operations, such as *bit-shuffling*, also known as *P-boxes*; *linear mixing*, usually using the *exclusive or* operation; and *nonlinear functions*, often known as *S-boxes*. The more rounds applied, the more secure is the cipher.

2.3 Overview of Block Ciphers

The first civilian block cipher was, perhaps, the cipher *Lucifer*, developed by IBM in the 1970s and mainly based on the work done by Horst Feistel. This cipher, with a few changes, was accepted by the US National Bureau of Standards (NBS) as the *Data Encryption Standard (DES)*, and publicly released in 1976. DES has a block size of 64 bits and a key size of 56 bits. Even from the beginning DES was criticized widely for its short size of the key. In 1998, *Electronic Frontier Foundation* developed a chip for breaking DES through a *brute force attack*. In parallel, a triple version of DES (*triple-DES*) was adopted to ensure privacy in communication applications. It has a 112 bit secret key and provides 80 bits of security level, and is still considered as secure.

On January 2, 1997, *US National Institute of Standards and Technology (NIST)* announced a competition for a new block cipher, an *Advanced Encryption Standard (AES)*. Two Belgian researchers, Vincent Rijmen and Joan Daemen, presented their cipher *Rijndael* in 1998, and after several rounds of evaluation it was chosen in 2001 as a new encryption standard — AES. AES has a block size 128 bits, and key sizes 128, 192, or 256 bits.

In Figure 2.3, a typical structure of encryption with a block cipher in the *Electronic Code Book (ECB)* mode is shown.

The encryption algorithm of a block cipher takes a message of size b bits, and produces a ciphertext of the same size. Thus, a block cipher is a bijective mapping on the space of 2^b possible inputs. The key chooses the

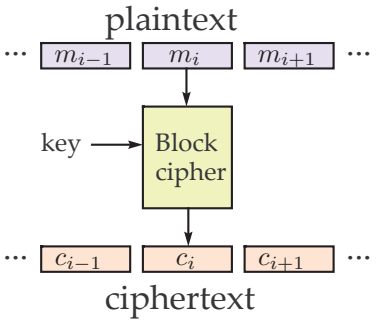
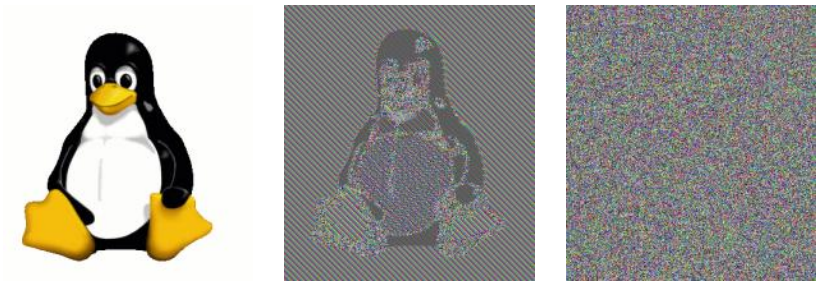


Figure 2.3: Model of a block cipher in the ECB mode.

permutation from 2^l possible permutations, although the total number of possible permutations is $2^b!$. The decryption algorithm works in a similar way.



(a) (b) (c)

Figure 2.4: Problems when encrypted with a block cipher: (a) original picture; (b) encrypted using ECB; (c) encrypted using CBC.

The problem with the encryption scheme in Figure 2.4 is that when one key is used to encrypt the same blocks, the ciphertext blocks are the same. Picture 2.4 borrowed from Wikipedia [Wik06a] shows this effect clearly.

Therefore, when the message is larger than a block size, different modes of operation are used as listed below. Now, $m_i, c_i \in \mathcal{A}^b$ are blocks of b symbols of the plaintext and the ciphertext, respectively.

- *Electronic Code Book (ECB).* This is the plain use of a block cipher, and is shown in Figure 2.3.
  Encryption: $c_i = E_K(m_i)$,
  Decryption: $m_i = D_K(c_i)$.

- *Cipher-Block Chaining (CBC).* A symbol from the plaintext is, before encryption, bit-wise XORed with the previous symbol of the ciphertext.
  Encryption: $c_i = E_K(m_i \oplus c_{i-1}), \quad c_0 = IV,$
  Decryption: $m_i = D_K(c_i) \oplus c_{i-1}, \quad c_0 = IV.$

- *Cipher Feedback (CFB).* A block cipher in the CFB mode can be regarded as a *self-synchronizing stream cipher* (see Section 2.4.1.2). Flipping one bit in the ciphertext flips the same bit in the plaintext, thus revealing the principle of OTP.
  Encryption: $c_i = m_i \oplus E_K(c_{i-1}), \quad c_0 = IV,$
  Decryption: $m_i = c_i \oplus E_K(c_{i-1}), \quad c_0 = IV.$

- *Output Feedback (OFB).* A block cipher in the OFB mode can be regarded as a *synchronous stream cipher* (see Section 2.4.1.1).
  Encryption: $c_i = m_i \oplus o_i, \quad o_i = E_K(o_i), \quad o_0 = IV.$

- *Counter Mode (CTR).* As CFB and OFB modes, the counter mode turns a block cipher into a stream cipher. The encryption process is
  Encryption: $c_i = m_i \oplus E_K(\text{nonce}||\text{counter}_i),$
  where $||$ denotes concatenation. The counter is known and the *nonce* value is used as a constant during encryption with one key. It should be guaranteed that the same nonce does not appear twice.

Let us give two classical examples of block ciphers. The first uses the principle of *substitution*, and the second uses the principle of *permutation*. The fundamental differences between these two ciphers were, perhaps, first discovered and mentioned by *Giovanni Battista Porta* [Kah67] in 1563, and these differences are important because they reveal basic principles in the design of modern primitives. Both examples will be used in the chapter on cryptanalysis when standard cryptanalytic tools will be defined.

Definition 2.9 (Substitution Cipher): Let the alphabet be $\mathcal{A}$, and the key space $\mathcal{K}$ consist of all possible permutations on $|\mathcal{A}|$ symbols, i.e., $|\mathcal{K}| = |\mathcal{A}|!$. Then for any key $K \in \mathcal{K}$, the encryption and decryption functions are defined as
Encryption: $c_i = K(m_i),$
Decryption: $m_i = K^{-1}(c_i).$ □

EXAMPLE *2.2 (Substitution Cipher):* Let the alphabet $\mathcal{A} = \{0, 1, \ldots, 25\}$ correspond to English letters, and the key K be the following permutation.

x	a	b	c	d	e	f	g	h	i	j	k	l	m
$K(x)$	C	I	V	A	T	Y	M	P	S	F	L	E	X
x	n	o	p	q	r	s	t	u	v	w	x	y	z
$K(x)$	O	H	Z	G	B	U	W	Q	N	K	J	R	D

In this table, small letters denote plaintext symbols, and capital letters are ciphertext symbols. Assume that we want to encrypt the following message (spaces are omitted)

$$\mathbf{m} = \texttt{thiscipherisnotverystrong}.$$

This will be encrypted as

$$\mathbf{c} = \texttt{WPSUVSZPTBSUOHWNTBRUWBHOM}.$$

□

Definition 2.10 (Permutation Cipher): Let us have an alphabet $\mathcal{A}$ and some fixed integer $r \geq 1$. Let the key space $\mathcal{K}$ consist of all possible permutations of r values, i.e., $|\mathcal{K}| = r!$. Encryption of the plaintext is considered block-wise, where each block $(m_1, m_2, \ldots, m_r)$ consists of r consecutive symbols of the plaintext. For a chosen key $K \in \mathcal{K}$, the encryption of one such block is thus done as

Encryption: $(c_1, c_2, \ldots, c_r) = (m_{K(1)}, m_{K(2)}, \ldots, m_{K(r)})$,
Decryption: $(m_1, m_2, \ldots, m_r) = (c_{K^{-1}(1)}, c_{K^{-1}(2)}, \ldots, c_{K^{-1}(r)})$.

□

EXAMPLE *2.3 (Permutation Cipher):* Let us again have the alphabet $\mathcal{A} = \{0, 1, \ldots, 25\}$, in correspondance to English letters. Let the secret key $K \in \mathcal{K}$ be a permutation on a block of $r = 6$ symbols, i.e., $\mathcal{B} = \mathcal{A}^6$ can be regarded as an extended alphabet for encryption. The permutation K is

x	1	2	3	4	5	6
$K(x)$	5	4	1	6	2	3

Assume that we want to encrypt the following plaintext,

$$\mathbf{m} = \texttt{thepermutationcipherisnotchallenging}.$$

First, we split it into blocks of size r.

$$\mathbf{m} = \texttt{theper|mutati|onciph|erisno|tchall|enging}.$$

Then we apply the permutation K, and merge the ciphertext symbols together.

$$\mathbf{c} = \texttt{EERHTPTTIUMACPHNOIINORESHLLCTAGNGNEI}.$$

□

2.4 Stream Ciphers in a Nutshell

Stream ciphers are important primitives for ensuring privacy in communication. For one example, they are widely used in telecommunication applications. It is believed that stream ciphers can be secure, efficient, and small in implementation, and better than block ciphers in these aspects. However, the security of stream ciphers has not been studied in sufficient detail. The current effort of many cryptographers is to make design and analysis of stream ciphers more understandable.

2.4.1 General Structure of Stream Ciphers

By Definition 2.5 the structure of a stream cipher can be viewed as *a finite state machine (FSM)* with *internal state (IS)* and *update function (UF)* for the internal state. From a cryptographic point of view, the strength of almost any stream cipher is based on the results produced by Shannon on the security of the one-time pad cipher (recall Section 2.2.1).

Commonly, a stream cipher receives a key K and an IV, and generates a long *keystream* $\mathbf{z}$, also known as a *pseudo-random sequence*. The definition of the *keystream* can be stated as follows.

Definition 2.11 (Keystream): The finite state machine of almost any stream cipher working over some alphabet $\mathcal{A}$ produces a long sequence $\mathbf{z}$ of symbols $z_1, z_2, \ldots, z_n$ from the same alphabet $\mathcal{A}$, i.e.,

$$\mathbf{z}^n = z_1, z_2, \ldots, z_n, \quad z_i \in \mathcal{A}, \;\; i = 1, 2, \ldots, n. \tag{2.9}$$

This sequence is called the *keystream*, and is combined with the plaintext to produce the ciphertext, thus revealing the principal of OTP. □

The combining function of the plaintext and the keystream is usually just a simple *exclusive or (XOR, $\oplus$)* operation.

If the keystream would be completely random, such a stream cipher is "unbreakable". However, that can only happen if we use OTP. Stream ciphers instead try to generate a keystream $\mathbf{z}$ that looks as random as possible, seeded with the secret key K.

Most of stream ciphers can be classified as *synchronous* and *self-synchronizing*. However, the classification of stream ciphers is not limited to these two classes (for example, Helix [FWS+03] belongs to another type). More details can be found in, e.g., [MvOV96].

2.4.1.1 Synchronous Stream Ciphers

A typical structure of a synchronous stream cipher is shown in Figure 2.5. Formally, such a structure can be defined as follows.

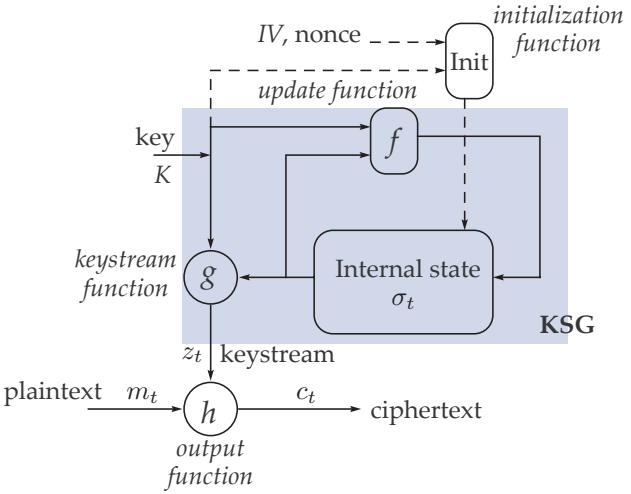


Figure 2.5: General structure of a synchronous stream cipher.

Definition 2.12 (Synchronous Stream Cipher): A *synchronous stream cipher (SSC)* is a finite state machine, the update function that receives the secret key K, but independent of the plaintext and the ciphertext. A SSC consists of

- *Internal state*, σ_t, denotes the value of the internal state at time t.
- *Initialisation function*, Init$(\cdot)$, is applied to set up the initial value of the IS σ_0.
- *Update function*, $f(\cdot)$, is the function updating the IS in the FSM, perhaps, depending on the key K, i.e.,

$$\sigma_{t+1} = f(\sigma_t, K), \quad \text{at time } t = 0, 1, \ldots. \tag{2.10}$$

- *Keystream function*, $g(\cdot)$, decides the output symbol as a function of the internal state, perhaps, depending on the key K, i.e.,

$$z_t = g(\sigma_t, K), \quad \text{at time } t = 0, 1, \ldots. \tag{2.11}$$

- *Output function*, $h(\cdot)$, is the function combining the keystream and the plaintext, resulting in the ciphertext, i.e.,

$$c_t = h(m_t, z_t), \quad \text{at time } t = 0, 1, \ldots. \tag{2.12}$$

The part that produces the keystream is called a *keystream generator (KSG)*.

□

Note that on the decryption side the keystream is generated in the same way as for the encryption, and it is only required that the output function h is invertible.

Synchronous stream ciphers have a problem with synchronisation in communication. If one symbol is lost or inserted during the transmission, then all the consecutive message symbols will be decrypted wrong. This problem can be solved by using *frames* of the ciphertext, with a synchronisation process between the frames, usually involving a special *tag sequence*. The size of one frame is fixed and the plaintext is encrypted portionally, one frame after another, as shown in Figure 2.6. One frame usually consists of a frame number and encrypted block of data. Before the encryption of one frame of information a stream cipher is reinitialised with the secret key *and* an *initial value*, which can be derived from the *frame counter* via some publicly known function.

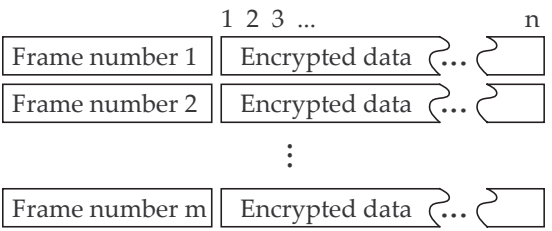


Figure 2.6: The use of frames for resynchronisation.

Definition 2.13 (Frame, Frame Counter, Initial Value, Nonce):

- A *frame* is a block of a ciphertext of a fixed length. A synchronisation procedure is included between the frames with the purpose to prevent loss of information in the communication channel.
- An *Initial value (IV)* is the same as *nonce* or *frame counter*. This is a publicly known parameter for the initialisation procedure of a cipher, directly calculated from the frame counter, or set up publicly by agreement of the communication parties. An IV value should be used only once for a fixed key.

To establish a secure communication channel **A***lice* and **B***ob* may have to share a secret key via some key exchange protocol. This exchange procedure might take a long time. Therefore, the initialisation process should additionally accept the initial value as a parameter. In this way we can generate many keystreams from the same key.

A typical example of a synchronous stream cipher is A5/1 [BGW99] used in GSM communication in the European part of the world. The size of each frame is 228 bits, and the IV is derived directly from the frame counter. Description of A5/1 and its cryptanalysis are given in Chapter 5.

2.4.1.2 Self-Synchronizing Stream Ciphers

The synchronisation problem involved with synchronous stream ciphers can also be solved with the use of *self-synchronization stream ciphers (SSSC)*.

Definition 2.14 (Self-Synchronizing Stream Ciphers): A *self-synchronizing* stream cipher is one in which the keystream is generated as a function of the key and a fixed number of previous ciphertext digits. □

The main property of an SSSC is that the internal state is fully determined from d consecutive symbols of the ciphertext. Thus, mistakes like loss or insertion of a symbol will only have an influence on the following d symbols, and then continue to encrypt the message correctly. The general structure of an SSSC is shown in Figure 2.7.

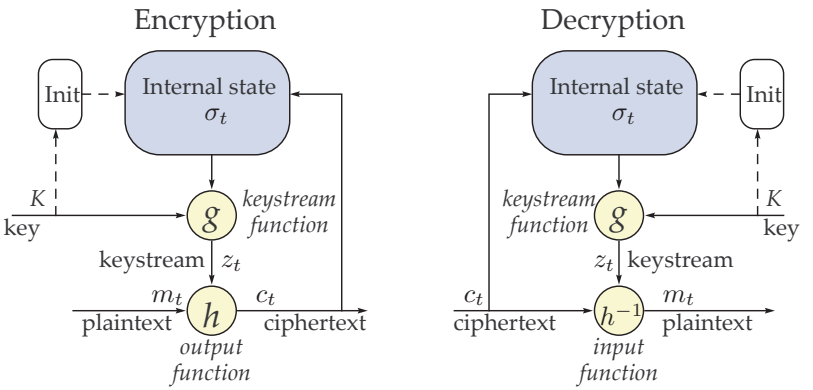


Figure 2.7: General structure of a self-synchronizing stream cipher.

2.4.2 Pseudo-Random Number Generators

The design of a good synchronous stream cipher usually means designing a good *pseudo-random number generator (PRNG)*, which is then used as a keystream generator. One common model of a synchronous stream cipher

with a PRNG is shown in Figure 2.8. The similarity of this structure to the one shown in Figure 2.5 is as follows. The structural part of a stream cipher works independently (from the key, the plaintext, the ciphertext and the keystream) after the initialisation process, and generates the keystream without the use of the secret key, and is called a *pseudo-random number generator*. Thus, a PRNG is equivalent to a *keystream generator*.

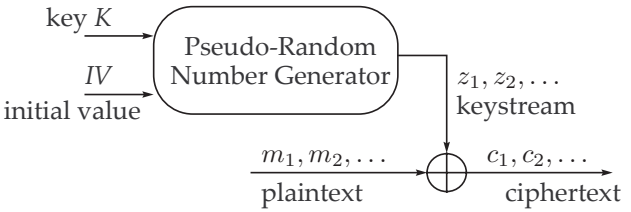


Figure 2.8: Pseudo-random number generator as the keystream function.

Definition 2.15 (Truly Random Number Generator): A *truly random number generator (TRNG)* is an *oracle* that produces independent numbers (or *samples*) from some alphabet $\mathcal{A}$ in an *unpredictable and non-repeatable* way. □

There are many good physical devices that generate random numbers with properties close to a TRNG. These include such phenomena as *radioactive decay* [2], *thermal noise* [3], and others. However, although these generators are good for simulations in statistics, they do not satisfy the following important and required property for a random number generator to be used in cryptography: *the random sequence must be possible to repeat.* This statement basically means that the process must be deterministic in some way, i.e., repeatable.

Definition 2.16 (Pseudo-Random Number Generator (PRNG)): A *pseudo-random generator (PRNG)* is a deterministic algorithm that attempts to produce samples from some alphabet $\mathcal{A}$ that *looks* independent and uniformly distributed. It tries to behave as close to a TRNG as possible. A PRNG has a *seed* as its initialisation parameter, and always produces the same sequence of numbers for the same seed. □

[2] *Radioactive decay* is the process wherein atomic nuclides emit subatomic particles. This is a *random* process, i.e., it is impossible to predict the decay of an atom.

[3] *Thermal noise*, also known as *Johnson-Nyquist noise*, is the noise that appears due to equilibrium fluctuations of the electric current inside of an electrical conductor, since the thermal motion of electrons is *random*.

However, the practice of design and analysis shows that it is not so easy to make such a good PRNG. A good PRNG should meet a set of statistical requirements:

1. *Period length.* Since PRNG is a finite state machine, it means that sooner or later it will reach a state where it has already been before (not necessarily the state that we have started from). The deterministic procedures of generating the next value of the internal state will assure that PRNG will produce a sequence that has a *period* of some length. One important requirement is that this period should be large enough.

2. *Statistical Properties.* Output symbols should be uniformly distributed, i.e., for binary symbols the probability of 0 and 1 should be $1/2$ for both. Moreover, the joint distribution of two or more bits in a window of the keystream should be the uniform distribution. Finally, any linear combination of output bits should be from the uniform distribution as well. There are a variety of statistical tests addressing these issues, such as *the frequency test*, *the serial test*, *the poker test*, *the gap test*, *Diehard tests*, *Maurer's universal statistical test*, *the autocorrelation test*, and others.

EXAMPLE *2.4 (A Simple PRNG):* Let us look at a very simple PRNG of the form

$$N_{i+1} = (A \cdot N_i + B) \mod C, \tag{2.13}$$

where N_i is the output symbol and the internal state of the generator, and A, B and C are parameters. Let

$$A = 2, \quad B = 3, \quad C = 19.$$

Then we have the following periodical sequence of numbers, which is supposed to "look random":

$$[0, 3, 9, 2, 7, 17, 18, 1, 5, 13, 10, 4, 11, 6, 15, 14, 12, 8]^{\infty}.$$

It is easy to see that the period of such a generator is 19, but what about its statistical property? We have:

$$\begin{aligned} N_{i+2} &= (A \cdot N_i + B) \cdot A + B = A^2 \cdot N_i + (1 + A)B \mod C, \\ &\vdots \\ N_{i+t} &= A^t \cdot N_i + (1 + A + A^2 + \ldots + A^{t-1})B \mod C. \end{aligned} \tag{2.14}$$

Any output symbol N_{i+t} of the stream can be expressed via the preceding number N_i (as well as subsequent ones). If we let $i = 0$ and set $N_0 = 0$, then we have the formula

$$N_t = \frac{A^t - 1}{A - 1} \cdot B \mod C. \tag{2.15}$$

For example, $N_5 = \frac{2^5-1}{2-1} \cdot 3 = 93 \equiv 17 \mod 19$ – the value is easy to predict when the formula above is found. Obviously, this PRNG is too simple to be used in cryptography. □

Perhaps the best random generator that is cryptographically strong is the *Blum-Blum-Shub (BBS)* PRNG, proposed by *Lenore Blum, Manuel Blum*, and *Michael Shub* in 1986. The calculations are done in a number field, and its security strength is based on the *computational difficulty of integer factorization problem*. To attack BBS one needs to factorize the modulus M, which is itself a product of two large primes $M = pq$. Distinguishig the output bits from random will be at least as difficult as factoring M. Thus, the quality of BBS PRNG can easily be increased by choosing a larger M.

However, although this generator is good, it is slow, and, again, cannot be used in symmetric cryptography.

Underlying the above sections, a PRNG plays a role as a keystream generator in synchronous stream ciphers, where the pair of a key and an IV of the cipher is used as a seed for the PRNG.

2.5 Stream Cipher Building Blocks

The following subsections comprise the introduction to various building blocks for keystream generators in brief.

2.5.1 Boolean Functions

Boolean functions have always been important pieces in cipher design. A proper choice of a Boolean function may significantly increase the resistance to different kinds of attacks. This fact inspired many scientists to study this subject in detail. When designing cryptographically significant Boolean functions, many requirements have to be fulfilled, such as *balancedness*, *nonlinearity*, *algebraic degree*, *correlation immunity*, *algebraic immunity*, and others. Some of them may contradict each other, e.g., *bent functions*, which have the highest possible nonlinearity, can not be balanced. Getting the best possible trade-off among these parameters has always been a challenging task (see [Gup04, P. 04, SM00a, SM00b] and references in these papers).

REMARK: In this section the notation will be slightly different from the previous ones. A function f will be on n variables $x_1, x_2, \ldots, x_n$ with *resiliency* m, *degree* d and *nonlinearity* σ.

Definition 2.17 (Boolean Function (BF)): A *Boolean function (BF)* f is a mapping $\mathbb{F}_2^n \to \mathbb{F}_2$, where n is the number of input variables $x_1, x_2, \ldots, x_n \in \mathbb{F}_2$, and $\mathbb{F}_2 = \{0, 1\}$. □

There exist *four* Boolean functions for *one* variable, and 16 different functions for *two* variables. Truth tables of the most important functions on one and two variables are shown in Table 2.1.

BF on two variables				BF on one variable	
Inputs	*OR*, $\vee, \vert$	*AND*, $\wedge, \cdot, \&$	*XOR*, $\oplus, +$	Input	*NOT*, $\overline{x}, !$
0 0	0	0	0	0	1
0 1	1	0	1	1	0
1 0	1	0	1		
1 1	1	1	0		
Priority	*low*	*medium*	*low*		*high*

Table 2.1: The most used Boolean functions on two variables, and their truth tables.

Perhaps the most important representations of a Boolean function for cryptography are as follows.

Definition 2.18 (Truth Table (TT), Algebraic Normal Form (ANF)):

- A Boolean function $f(x_1, \ldots, x_n)$ can be defined through its *truth table (TT)*, i.e., a binary string of length 2^n,

$$\mathrm{TT}(f) = [f(0, 0, \ldots, 0), f(1, 0, \ldots, 0), f(0, 1, \ldots, 0), \ldots, f(1, 1, \ldots, 1)]. \tag{2.16}$$

- A Boolean function has a *unique* representation as a polynomial over $\mathbb{F}_2$, called the *algebraic normal form (ANF)*,

$$f(x_1, \ldots, x_n) = a_0 \oplus \bigoplus_{1 \le i \le n} a_i x_i \oplus \bigoplus_{1 \le i < j \le n} a_{ij} x_i x_j \oplus \ldots \oplus a_{12\ldots n} x_1 x_2 \ldots x_n, \tag{2.17}$$

  where the coefficients $a_0, a_{ij}, \ldots, a_{12\ldots n} \in \{0, 1\}$.

□

EXAMPLE *2.5 (A Boolean function):* Let us consider a Boolean function on $n = 5$ variables,

$$f(x_1, x_2, x_3, x_4) = \overline{x_1} \& x_2 \& x_4 \vee x_1 \& x_2 \& x_4 \vee x_1 \& \overline{x_2} \& x_4. \tag{2.18}$$

The function is given in a so-called *Disjunctive Normal Form (DNF)*. After a few simplifications of the function, we derive its unique ANF:

$$f(x_1, x_2, x_3, x_4) = x_1x_2 + x_1x_2x_3 + x_1x_4 + x_2x_4. \tag{2.19}$$

Its truth table is

$$\mathrm{TT}(f) = [0001000001110110]. \tag{2.20}$$

□

There is a class of BFs that are called *linear* and *affine* functions, depending on the form of their ANF.

Definition 2.19 (Degree, Linear BF, Affine BF):

- The *algebraic degree*, $d = \deg(f)$, is the number of variables in the highest order term with non-zero coefficient.
- A Boolean function is called *affine* if there exists no term of degree > 1 in the ANF. The set of all affine functions is denoted by $A(n)$.
- An affine function with constant term equal to zero is a *linear* function. The set of all linear functions is denoted as $L(n)$. The general form of a linear function f_L is

$$f_L = \omega_1 \cdot x_1 + \omega_2 \cdot x_2 + \ldots + \omega_n \cdot x_n, \quad \omega_i \in \mathbb{F}_2,\ i = 1, 2, \ldots, n. \tag{2.21}$$

□

Some properties of a Boolean function, such as *balancedness* and *correlation immunity*, can be described through *Hamming weight* and *Hamming distance*, and we give the definitions below.

Definition 2.20 (Hamming Weight, Hamming Distance):

- The *Hamming weight* of a binary sequence S, denoted $w_H(S)$, is the number of ones in S.
- The *Hamming distance* between two binary sequences S_1 and S_2 of the same length, denoted $d_H(S_1, S_2)$, is the number of positions where they differ. The relation $d_H(S_1, S_2) = w_H(S_1 \oplus S_2)$ is valid.

□

A Boolean function f is called *balanced* if the truth table contains an equal number of ones and zeros, i.e., $w_H(f) = 2^{n-1}$. Many other properties of a Boolean function can be expressed via the *Walsh transform*.

Definition 2.21 (Walsh Transform, Walsh Spectrum): Let $x = (x_1, \ldots, x_n)$ and $\omega = (\omega_1, \ldots, \omega_n)$, $x, \omega \in \{0,1\}^n$ and let $x \cdot \omega = x_1\omega_1 \oplus \ldots \oplus x_n\omega_n$. Let $f(x)$ be a Boolean function on n variables. Then the *Walsh transform* of $f(x)$ is a real valued function over $\{0,1\}^n$, defined as

$$W_f(\omega) = \sum_{x \in \{0,1\}^n} (-1)^{f(x) \oplus x \cdot \omega}, \tag{2.22}$$

where the integer valued vector W_f is called the *Walsh spectrum*. □

Here $\omega \cdot x$ is the linear function in inner product notation. Thus, if we think of ω as a linear function in sense of (2.21), then the value $W_f(\omega)$ corresponds to the number of inputs x to the functions f and ω where they are equal, minus the number of points where they differ. Thus,

$$d_H(f, \omega) = \frac{2^n - W_f(\omega)}{2}. \tag{2.23}$$

For example, if the function f is *balanced* then $w_H(f) = d_H(f, 0)$ must be 2^{n-1}, which implies $W_f(0) = 0$.

Boolean functions met wide applications in cryptography. They play a central role in designing *S-boxes*, *combining functions*, *generating functions* of *nonlinear feedback shift registers*, and other functional blocks. For each of the applications, a Boolean function should satisfy a set of specific properties, which could be in a trade-off relation. The list of typical properties of Boolean functions and their definitions is as follows.

Definition 2.22 (Properties of a Boolean Function):

- *Balancedness*, as mentioned above, implies

$$w_H(\mathrm{TT}(f)) = 2^{n-1} \quad \Leftrightarrow \quad W_f(0) = 0. \tag{2.24}$$

- The *nonlinearity*, $nl(f)$, of an n variable function f is the minimum distance to the set of all n variable affine functions, i.e.,

$$nl(f) = \min_{g \in A(n)} (d_{\mathrm{H}}(f, g)) \quad \Leftrightarrow \quad nl(f) = 2^{n-1} - \frac{1}{2} \max_{\omega \in \mathbb{F}_2^n} |W_f(\omega)|. \tag{2.25}$$

  This criterion was introduced by Meier and Staffelbach [MS90]. BFs used in ciphers must often have high nonlinearity to prevent linear attacks [DXS91, Mat94].

- An n variable BF is called m^{th} *order correlation immune (CI)* if for any m-tuple of i.i.d. random variables $X_{i_1}, X_{i_2}, \ldots, X_{i_m}$ we have

$$I(X_{i_1}, X_{i_2}, \ldots, X_{i_m}; Y) = 0, 1 \leq i_1 < \ldots < i_m \leq n, \tag{2.26}$$

  where $Y = (X_1, X_2, \ldots, X_m)$, and $I(X;Y)$ denotes the mutual information [CT91]. If f is additionally balanced, then it is called m*-resilient*. Thus, a function is m*-resilient* (respectively m^{th} *order correlation immune*) iff its Walsh transform satisfies

$$W_f(\omega) = 0, \text{ for } \forall \omega : 0 \leq w_{\text{H}}(\omega) \leq m \text{ (respectively, } 1 \leq w_{\text{H}}(\omega) \leq m). \tag{2.27}$$

- The *strict avalanche criterion (SAC)* [For88, Llo92] is important in designing both S-boxes and combining functions [4]. A simple SAC means that if one bit of the input is changed, the output should change with probability 1/2. A function f satisfies to the t^{th} order SAC if, when t bits of the input are kept constant, the output bits change with probability 1/2 when one of the remaining input bits is flipped.
- The term of *algebraic immunity degree (AI)* was recently introduced [MPC04] to estimate the complexity of an *algebraic attack*. For the attack to be successful, there must exist an *annihilator function* h (a non zero function s.t. $f \cdot h = 0$), with a small degree. The minimum degree of an annihilator is called *algebraic immunity degree*. The best results for finding the value of AI and construction of BF with AI of a high-degree are, perhaps, in [DGM06].

□

Following the notation in [P. 04, SM00a, SM00b] we use (n, m, d, σ) to denote an n-variable, m-resilient function with degree d and nonlinearity σ. By $[n, m, d, \sigma]$ we denote an unbalanced n-variable, m^{th} order correlation immune function with degree d and nonlinearity σ.

2.5.2 Finite Fields

Many topics in stream ciphers are based on finite fields.

Definition 2.23 (Field): A *field* is a triple $< F, \cdot, + >$ where $< F, + >$ is an abelian additive group with the identity 0, $< F \setminus \{0\}, \cdot >$ is an abelian multiplicative group with the identity 1, and $< F, \cdot, + >$ satisfies the distributive law. □

[4] A cipher can be viewed as an S-box. For differential cryptanalysis, the attacker will select pairs of inputs, x_1 and x_2 such that they satisfy to a particular difference $\Delta x = x_1 \oplus x_2$, knowing that a particular difference of outputs $\Delta y = y_1 \oplus y_2$ occurs with high probability. This leads to a differential attack on the cipher. Using S-boxes satisfying *SAC*(k) with higher value of k decreases the propagation ration, and, hence, can resist differential cryptanalysis.

For a field with multiplicative identity 1, a *field characteristic* is the smallest positive integer p such that $\underbrace{1+1+\ldots+1}_{p \text{ times}} \equiv 0$. A field $< F, \cdot, + >$ is usually denoted by $\mathbb{F}_{p^d}$, where p is the field characteristic, and p^d is the number of elements in the field.

Consider the case when $\mathbb{F}_p = \mathbb{Z}_p = \{0, 1, \ldots, p-1\}$. It can be shown that $\mathbb{F}_p$ is a field only if p is prime. Let us consider some polynomial over the field $\mathbb{F}_p$ of a fixed degree d in the form

$$f(x) = \sum_{i=0}^{d} b_i \cdot x^i, \quad b_i \in \mathbb{F}_p \;\; b_d \neq 0. \tag{2.28}$$

All polynomials from $\mathbb{F}_p[x]$ taken modulo the polynomial $f(x)$ also form a ring, where every element $\alpha(x)$ is a polynomial of degree $< d$. This ring is denoted as $\mathbb{F}_p[x]/f(x)$, where $f(x)$ is called the *generating polynomial*. All elements of the field are possible *residues* when taking an abstract polynomial from $\mathbb{F}_p[x]$ modulo $f(x)$.

A polynomial $f(x)$ over $\mathbb{F}_p$ of degree d generates the finite field consisting of *all* polynomials of degree $< d$ over $\mathbb{F}_p$ only if $f(x)$ is *irreducible*, i.e., it cannot be factorised over $\mathbb{F}_p$. This field id denoted as $\mathbb{F}_{p^d}$. For example, $f(x) = x^4 + x + 1$ over $\mathbb{F}_2$ is irreducible, whereas $f(x) = x^4 + 1$ over $\mathbb{F}_2$ is reducible since $x^4 + 1 \equiv (x^2 + 1)^2$, i.e., it can be factorised. To show that $f(x)$ is the generating polynomial, sometimes we write $f(\alpha) = 0$, for some $\alpha \in \mathbb{F}_{p^d}$.

REMARK: We distinguish between the notations $\mathbb{F}_{16}$, $\mathbb{F}_2^4$, and $\mathbb{F}_{2^4}$ as follows. $\mathbb{F}_{16}$ is a set of integers from 0 to 15; $\mathbb{F}_2^4$ is a set of 4-dimensional binary vectors; $\mathbb{F}_{2^4}$ is a finite field, whose generating polynomial is of degree 4, and the ground field is $\mathbb{F}_2$.

EXAMPLE *2.6 (Finite Fields and Representations):* Let the generating polynomial be $f(x) = x^4 + 2x^3 + 2x^2 + x + 1$ over $\mathbb{F}_3$, which is irreducible. Then, any element $\alpha(x)$ of $\mathbb{F}_{3^4}$ is of the form

$$\alpha(x) = a_3x^3 + a_2x^2 + a_1x^1 + a_0x^0, \quad a_0, a_1, a_2, a_4 \in \mathbb{F}_3 = \{0, 1, 2\}, \tag{2.29}$$

i.e., the total number of elements (polynomials) in this field is $3^4 = 81$. For simplicity, people just write α, instead of $\alpha(x)$, when it is clear that we are working in the field. Any element α of the field $\mathbb{F}_{3^4}$ can be represented as a vector of coefficients, such as

$$\alpha = (a_3a_2a_1a_0), \quad a_i \in \mathbb{F}_3. \tag{2.30}$$

Thus, if $\alpha = 2x^4 + x + 1$, and $\beta = x^4 + x^3 + 2$, then their sum in the vector form will be calculated in a component-wise fashion modulo 3, and would look like

$$\alpha + \beta = (2011) + (1102) = (0110) = x^2 + x. \tag{2.31}$$

Multiplication is a bit more complicated:

$$\begin{aligned}\alpha \cdot \beta &= (2011) \cdot (1102) \mod (12212) = (2210122) \mod (12212) \\ &= 2 \cdot (12212) + (101022) \mod (12212) \\ &= 1 \cdot (12212) + (20112) \mod (12212) \\ &= 2 \cdot (12212) + (1200) \mod (12212) = x^3 + 2x^2. \end{aligned} \tag{2.32}$$

□

More complicated *extension fields* can also be constructed. Thus, the generating polynomial $f(x)$ can itself be over some other *extension field*. For example, the polynomial $g(y) = \sum_{j=0}^{e} c_j y^j$ over the field $\mathbb{F}_{p^d}$, with the generating polynomial $f(x)$ defined in (2.28), could be the generating polynomial for an extension field $\mathbb{F}_{(p^d)^e}$, in case $g(y)$ is irreducible. The number of elements in such a field will then be $(p^d)^e$.

The ground field, the parameter p in our formulas, must be a *prime number* (otherwise, not all elements have an inverse), and it is called *the characteristic of the field*. In communication theory, finite fields with characteristic 2 are the most important ones, since the model of a communication channel is binary.

Since the multiplicative group of any finite field is cyclic [Dic58], it must have a *generator*. If the element $\alpha = x$ generates the multiplicative set of the field, then the generating polynomial of this field is called *primitive polynomial*.

EXAMPLE *2.7 (Binary Finite Field and Its Generator):* Let the ground field be the binary field $\mathbb{F}_2$, and the generating polynomial be

$$f(x) = x^3 + x^2 + 1 \quad \text{over } \mathbb{F}_2. \tag{2.33}$$

It will contain 8 elements. Let us take $\alpha = x \in \mathbb{F}_{2^3}$. Then, the elements of the field can be generated by α, as shown in Table 2.2. Here, $\alpha = x$ appears to be the *generator* of the finite field, and $f(x)$ is thus a *primitive* polynomial. □

As we can see, every non-zero element of the field can be represented as a power of α. We can then define the *discrete logarithm* to base α as follows.

$$t = \log_\alpha(\beta), \quad \text{such that} \quad \alpha^t = \beta \in \mathbb{F}_{p^d}. \tag{2.34}$$

For $\alpha = x$			
i	α^i	vector	polynomial
1	α^1	(010)	x
2	α^2	(100)	x^2
3	α^3	(101)	$x^2 + 1$
4	α^4	(111)	$x^2 + x + 1$
5	α^5	(011)	$x + 1$
6	α^6	(110)	$x^2 + x$
7	α^7	(001)	1
8	α^8	(010)	x

Table 2.2: Elements of the field $\mathbb{F}_{2^3}$ with $f(x) = x^3 + x^2 + 1$.

2.5.3 Linear Feedback Shift Registers

A typical block in a design of a stream cipher is a *linear feedback shift register (LFSR)*, the general structure of which is shown in Figure 2.9. Important LFSR-based stream ciphers include A5/1 [BGW99], E0 [Blu03], SNOW 2.0 [EJ02b], and others.

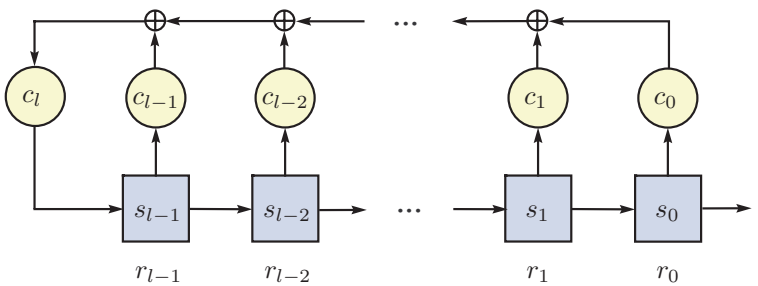


Figure 2.9: General structure of a linear feedback shift register.

An LFSR is a finite state machine that operates over some finite field $\mathbb{F}_q$ with some characteristic p, where $q = p^e$ for some $e \geq 1$. An LFSR consists of l *memory cells* $r_0, r_1, \ldots, r_{l-1}$ each containing one value from $\mathbb{F}_q$. At any time instance t the content of the register is called the *state* of the LFSR at time t, and denoted as $S_t = (s_{t+l-1}, s_{t+l-2}, \ldots, s_t)$. The state at time zero, S_0, is called the *initial state* of the LFSR.

The state S_{t+1} is derived from S_t as follows. When the control unit of the FSM is clocked, then the value of the cell r_0 goes to the *output*, while the

remaining cells are shifted as $r_i = r_{i+1}, i = 0, 1, \ldots, l-2$, and the last cell r_{l-1} is loaded with a new value s_{t+l}, calculated as

$$s_{t+l} = c_l \cdot \sum_{i=0}^{l-1} c_i \cdot s_{t+i} \quad \text{over } \mathbb{F}_q. \tag{2.35}$$

This relation is called a *linear recurrence relation*, where the constants $c_0, c_1, \ldots, c_l \in \mathbb{F}_q$ are called *feedback coefficients*, the connection at c_l is called the *feedback position*, and the connections at $c_0, \ldots, c_{l-1}$ are called *tap positions*. These constants constitute the *generating polynomial of the LFSR*

$$g(x) = c_l x^0 - c_{l-1} x^1 - c_{l-2} x^2 - \ldots - c_0 x^l \quad \text{over } \mathbb{F}_q[x]. \tag{2.36}$$

The state transition S_{t+1} from S_t can be represented in matrix form over the field $\mathbb{F}_q$ as follows,

$$\underbrace{\begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{l-2} \\ s_{l-1} \end{pmatrix}_{t+1}}_{S_{t+1}} = c_l \cdot \underbrace{\begin{pmatrix} 0 & c_l^{-1} & 0 & \ldots & 0 \\ 0 & 0 & c_l^{-1} & \ldots & 0 \\ & & & \vdots & \\ 0 & 0 & 0 & \ldots & c_l^{-1} \\ c_0 & c_1 & c_2 & \ldots & c_{l-1} \end{pmatrix}}_{M} \cdot \underbrace{\begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{l-2} \\ s_{l-1} \end{pmatrix}_{t}}_{S_t}. \tag{2.37}$$

Thus, any state S_t can be expressed through the bits of the initial state S_0 via the *transition matrix* M as

$$S_t = M^t \cdot S_0, \tag{2.38}$$

where the power t of the matrix M can be evaluated in logarithmical time $O(\log t)$.

The total number of possible states is q^l, and since the LFSR is a FSM, it has a period for every possible initial state S_0. For example, if $S_0 = (0\ 0\ \ldots\ 0)$, then, obviously, the period will be 1.

If the generating polynomial $g(x)$ is irreducible, then the internal state S_t can be regarded as an element of an extension field $\mathbb{F}_{q^l}$ with the generating polynomial $g(x)$. If, additionally, $g(x)$ is primitive, then the transitions of the states are

$$S_t = \alpha^t \cdot S_0, \quad \forall t, \tag{2.39}$$

where $\alpha = x$ is the generator of the extension field $\mathbb{F}_{q^l}$, and S_0, S_t are elements of the field given in the vector form (see Example 2.6). The following theorem is directly derived from Legendre's theorem, since $ord(\alpha) = 2^l - 1$, and the size of the multiplicative group is $2^l - 1$ as well.

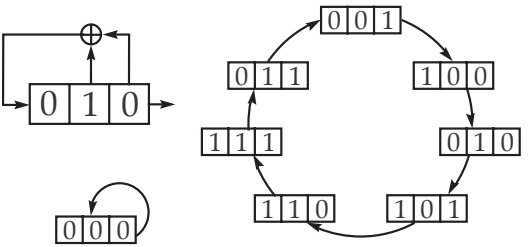


Figure 2.10: LFSR with $g(x) = x^3 + x^2 + 1$ over $\mathbb{F}_2$, and its state transitions.

Theorem 2.1: If the generating polynomial $g(x)$ over $\mathbb{F}_q$ for the LFSR is primitive and has degree l, then for every non-zero initial state S_0 the period of the LFSR is $T = q^l - 1$. □

Thus, an LFSR with a primitive generating polynomial is called a *maximum length LFSR*.

REMARK: From now on, we will consider only LFSRs with primitive generating polynomials $g(x)$, as such LFSRs are of great interest in cryptography.

For a *maximum length* LFSR over $\mathbb{F}_2$, tap positions for the generating polynomial should be relatively prime. An example of an LFSR is given below.

EXAMPLE *2.8 (Linear Feedback Shift Register):* Consider the primitive polynomial $g(x) = x^3 + x^2 + 1$ over $\mathbb{F}_2$ from Example 2.7. The corresponding LFSR and two cycles are shown in Figure 2.10.

□

If the length l of the LFSR is known, it is enough to have only l symbols at known positions from the output stream $s_0, s_1, \ldots$, to recover the initial state S_0 of the LFSR. However, if we are given a sequence of n symbols $\mathbf{s}^n = (s_0, s_1, \ldots, s_{n-1})$, a different problem is to find the *shortest LFSR* that generates this sequence. An efficient algorithm to find the shortest LFSR for a given sequence is the Berlekamp-Massey algorithm [Mas69].

Definition 2.24 (Linear Complexity (LC)): For a sequence of n symbols $\mathbf{s}^n = (s_0, s_1, \ldots, s_{n-1})$, the length of the shortest LFSR that generates the sequence is called the *linear complexity (LC)* of the sequence $\mathbf{s}^n$. □

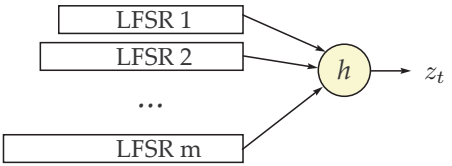


Figure 2.11: Keystream as a function on the outputs of several LFSRs, combined with the function $h(\cdot)$, or a *combination keystream generator*.

The linear complexity of any sequence produced by an LFSR is at most the length of the LFSR, i.e., l. If the sequence is $\mathbf{s}^n = (0, 0, 0, \ldots)$, its linear complexity is zero. On the other hand,

the linear complexity of a truly random number generator is $+\infty$.

One task of a keystream generator is to produce the keystream with a linear complexity that is very large. LFSRs have the following properties that are attractive in cryptography.

⊕ The period is large, and grows exponentially along with the size of the LFSR.

⊕ LFSR sequences have good statistical properties. If we consider a part of the sequence $s_t, \ldots, s_{t+n}$, its distribution is very close to the uniform distribution.

⊕ LFSR sequences are easy to construct and implement both in software and hardware.

⊖ It has a low linear complexity.

To increase the linear complexity of a sequence produced by an LFSR, several techniques can be applied. Let us further consider only LFSRs over $\mathbb{F}_2$.

In the first basic technique, several LFSRs are working in parallel, and the keystream is the output of the *combining Boolean function* $h(\cdot)$, the inputs to which are the outputs of the LFSRs. Note that the lengths of the LFSRs can be different. This scheme is depicted in Figure 2.11.

The linear complexity of this scheme is a well-known result, and the details of the proof can be found in [MvOV96].

Theorem 2.2: Assume we have a combiner as shown in Figure 2.11. If the function h is written in ANF, then the linear complexity of the keystream sequence $\mathbf{z}$ is

$$LC(\mathbf{z}) = h(l_1, l_2, \ldots, l_m), \tag{2.40}$$

where l_is are the lengths of the LFSRs, and the function h is evaluated without taking the modulo operation (i.e., performed in $\mathbb{Z}$). □

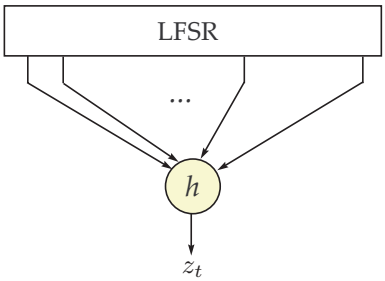


Figure 2.12: Keystream as a function on the state of an LFSR, or a *nonlinear filter keystream generator*.

Another method is to take some bits from the LFSR register, and produce the keystream as a function on these bits. This scheme is called *a nonlinear filter generator*, and shown in Figure 2.12. In [Key76], Key proved an upper bound for the linear complexity of the keystream for this generator.

Theorem 2.3: Assume that we have a *nonlinear filter generator* as shown in Figure 2.12. The linear complexity of the keystream $\mathbf{z}$ is then bounded as

$$LC(\mathbf{z}) \leq \sum_{i=1}^{d} \binom{l}{i}, \tag{2.41}$$

where l is the length of the LFSR, and $d = \deg(h)$. □

There exist many other clocking / combining methods for LFSRs, such as *Geffe's generator* [Gef73]; *Jennings's generator* [Jen80, Jen82]; *the Stop-and-Go generator*, proposed by Beth-Piper [BP84]; *the Double-sided Stop-and-Go generator* [ZYR89]; *self-decimated generators*, one proposed by Rainer Rueppel [Rue87], and another proposed by Bill Chambers and Dieter Gollmann [CG88a]; *the multi-speed scalar product generator*, proposed by L. Massey and R. Rueppel [MR84]; *Gollmann's cascade* [CG88b]; *Shrinking generators*, proposed by D. Coppersmith et al. [CKM93]; *Self-Shrinking generators*, proposed by W. Meier et al. [MS94]; and others...

2.5.4 Nonlinear Feedback Shift Registers

A competitive building block named *nonlinear feedback shift register (NLFSR)* has begun to appear in recent designs of stream ciphers, such as Dragon [CHM+05], TRIVIUM [CP05], Grain [HJM05a], and others. The general structure of a NLFSR is presented in Figure 2.13.

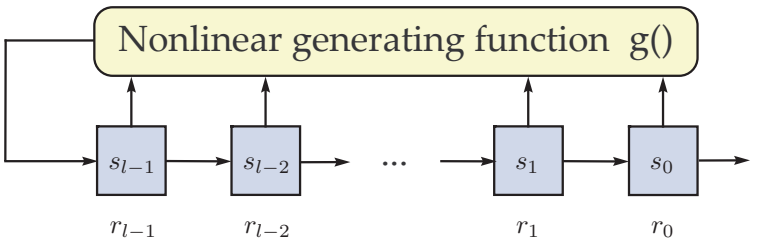


Figure 2.13: General structure of a nonlinear feedback shift register.

An NLFSR is similar to an LFSR, but the function $g(\cdot)$, used to generate a new value for the memory cell r_l, is not linear. The following pros and cons of an NLFSR can be given:

⊕ Easy to implement, especially in hardware.

⊕ The output sequence is very hard to predict, simply because *there is not much known about the construction and analysis of NLFSRs.*

⊕ The linear complexity is high, and algebraic attacks are (usually) not applicable. Many variables are included, through recursion, into algebraic expressions, that quickly grow in complexity and degree.

⊖ Hard to make the output sequence be balanced, and nobody can guarantee good statistical properties.

⊖ The maximum period of the sequence can be less than expected in average.

⊖ The sequence could look "random" for some time, and then fall into some static state, in case when updating function is not reversible.

In a hypothetical design of a stream cipher, one could take some random NLFSR with a huge internal state. Nobody could guarantee any properties of the design, but the output will have a large period with high probability.

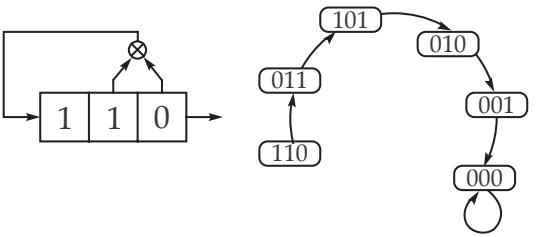


Figure 2.14: Example of an NLFSR and its state transitions.

For example,TRIVIUM [CP05] is such a design. Its internal state is 288 bits, and the authors hope the period is at least 2^{128}, which is quite *likely* true.

Another possible solution for preventing an NLFSR from falling into a spurious state is presented in the design of *Grain* [HJM05a], where the next state of the NLFSR depends on the output from a separate LFSR; thus, the NLFSR is acting as a filter.

EXAMPLE *2.9 (Nonlinear Feedback Shift Register):* Let us consider an NLFSR with 3 cells over $\mathbb{F}_2$ as shown in Figure 2.14. The recurrence relation is

$$s_{t+3} = s_t \cdot s_{t+1}. \tag{2.42}$$

We can see that if, for example, the initial state is $S_0 = (1\ \ 1\ \ 0)$, then the output sequence will be $\mathbf{z} = (0\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ \ \ldots)$.

□

Some additional research on FSRs can be found in, e.g., [CG86, Gol82]. We conclude that we need to learn more about NLFSRs, and derive techniques for their construction and analysis.

2.5.5 S-boxes and P-boxes

As we already saw in previous sections, combining functions play a central role in the security of a cipher. Criteria for combining functions will be discussed in the chapter on cryptanalysis techniques. In this section we present two more building blocks used in both block and stream ciphers, socalled *S-boxes* and *P-boxes*.

2.5.5.1 S-boxes

Definition 2.25 (S-box): An S-box is a mapping $\mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$, not necessarily invertible. Let $X \in \mathbb{F}_2^n$ be the input to the S-box, and $Y \in \mathbb{F}_2^m$ be its output.

The m bit output $Y = (y_1, y_2, \ldots, y_m)$ can be represented as a vector of m Boolean functions on n variables such as

$$Y = S[X] \quad \Rightarrow \quad \begin{cases} y_1 & = f_1(x_1, x_2, \ldots, x_n), \\ y_2 & = f_2(x_1, x_2, \ldots, x_n), \\ \vdots & \\ y_m & = f_m(x_1, x_2, \ldots, x_n). \end{cases} \tag{2.43}$$

□

Thus, S-boxes carry out the *confusion* concept (see Section 2.2.3), "shuffling" the bits of a single internal n bit word. The larger n, the better the confusion can be. During the encryption or decryption process, an S-box can be *fixed* or *dynamic*, depending on whether it can be regarded as a constant function, or if it changes along with the process. Moreover, an S-box can be *known* or *secret*. A known S-box is just a fixed map, whereas a secret S-box is a key dependent map.

Depending on where an S-box will be applied, it should satisfy different criteria. Common criteria are *balancedness*, *high nonlinearity*, and *correlation immunity*, defined as follows [Nyb92, Pas03, SZZ94].

Definition 2.26 (Properties of an S-Box): Let an S-Box be given by its Boolean functions $(f_1, f_2, \ldots, f_m)$.

- An S-box is *balanced* if any non-zero linear combination of the functions is balanced.
- The *algebraic degree* is the minimum algebraic degree out of all non-zero combinations of the functions, i.e.,

$$\deg(S) = \min_{(r_1,\ldots,r_m)\in\mathbb{F}_2^n\setminus\{\mathbf{0}\}} \deg\left(\sum_{i=1}^{m} r_i f_i\right). \tag{2.44}$$

- The *nonlinearity* is the minimum nonlinearity out of all non-zero combinations of the functions, i.e.,

$$nl(S) = \min_{(r_1,\ldots,r_m)\in\mathbb{F}_2^n\setminus\{\mathbf{0}\}} nl\left(\sum_{i=1}^{m} r_i f_i\right). \tag{2.45}$$

□

Clearly, a good S-box should have high algebraic degree and nonlinearity as well as being balanced, otherwise various attacks can be applied. Generally, designing S-boxes is a difficult and challenging problem, and for more information we refer to [Pas03, Gup04].

2.5.5.2 P-boxes

P-boxes are *permutation boxes* or *permutation rules*, applied on *symbols* or *words* (could be just bits). A P-box is used to support the *diffusion* concept (see Section 2.2.3), and it is used in many ciphers, for example, in DES.

Definition 2.27 (P-Box): A P-Box is a permutation on n ordered words. □

The number of permutations on n words is $n!$.

EXAMPLE *2.10 (P-Box):* Let $V \in \mathcal{A}^6$ be a vector of six symbols from $\mathcal{A} = \{A, B, \ldots, Z\}$,

$$V = (C, R, Y, P, T, O).$$

Let the permutation ρ be written as follows

$$\rho = (3, 2, 6, 5, 4, 1).$$

This means that the position 1 goes to the position 3, position 2 goes to the position 2, and so on. The application of the permutations ρ to the vector v one and two times gives us two new vectors

$$\rho(V) = (O, R, C, T, P, Y), \quad \rho^2(V) = (Y, R, O, P, T, C).$$

□

If we track the position of the first letter, then sooner or later it will come back to position 1. The path where the first letter went is called *an orbit*. All other letters on this path will go the same way, around the orbit. In our example, the first letter '*C*' will go to $1 \to 3 \to 6 \to 1 \to \ldots$, i.e., its orbit contains the positions $(1, 3, 6)$. The next letter, not included in the first orbit, is '*R*' – it will go as $2 \to 2 \to 2 \ldots$, hence its orbit is just (2). Continuing in this was, any permutation can be written as a product of orbits. In our example we have

$$\rho = (1, 3, 6)(2)(4, 5).$$

The *length of an orbit*, τ, is the number of positions in the orbit. Thus, in our example we have $\tau_1 = 3, \tau_2 = 1, \tau_3 = 2$.

Theorem 2.4 (Period of a permutation): Let a permutation ρ consist of k orbits $\sigma_1, \ldots, \sigma_k$ each of length $\tau_1, \ldots, \tau_k$, respectively. The period of the permutation ρ is

$$\text{Period}(\rho) = \gcd(\tau_1, \tau_2, \ldots, \tau_k). \tag{2.46}$$

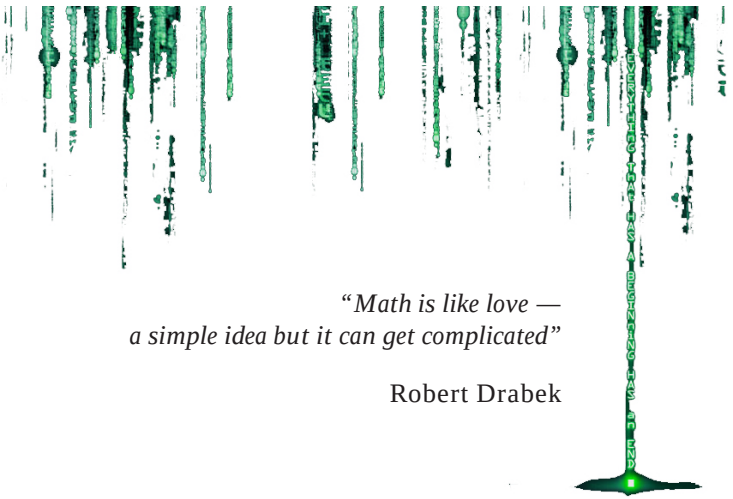
□

Some designs of stream ciphers, such as RC4 [Sma03], VMPC [Zol04], are based on a dynamic permutation, i.e., one that is changed dynamically in the process of keystream generation.

In this chapter we summarized techniques for the construction of block and stream ciphers, and briefly introduced building blocks like Boolean functions and finite fields. We continue with cryptanalysis techniques in the next chapter.

3

Techniques for Cryptanalysis



"Math is like love —
a simple idea but it can get complicated"

Robert Drabek

Long time ago, when people started to encrypt their secrets, on the other side of barricades curious creatures started to analyse ciphertexts in attempts to reveal the hidden information. Since that time, cryptology and cryptanalysis always walk along together, helping people to understand cryptography better, and design excellent algorithms for privacy.

In the ancient time during the golden age of the Islamic civilization, many foreign manuscripts were brought to Baghdad and placed at the great Arab libraries. Some of the manuscripts were encrypted, which motivated work to break the ciphers and reveal the information within.

Perhaps, the first significant cryptanalysis that gave birth to the science of *code breaking* is the attack on the substitution cipher from the 9^{th} century, carefully described by the scientist *Abu Yusuf Ya 'qub ibn Is-haq ibn as-Sabbah ibn 'omran ibn Ismail al-Kindi* in his manuscript "*On Deciphering Cryptographic Messages*" (see Figure 3.1), rediscovered in 1987 in the *Sulaimaniyyah Ottoman Archive* in Istanbul; this manuscript describes the oldest cryptanalysis method – *frequency analysis*.

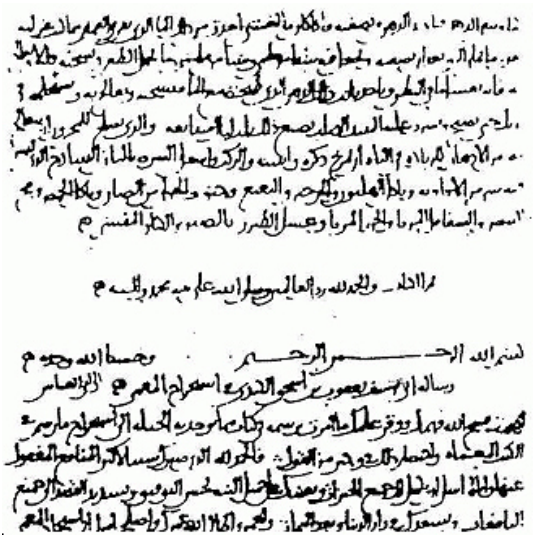


Figure 3.1: The first page from al-Kindi's manuscript "*On Deciphering Cryptographic Messages*".

In the Arabic language the most frequent letters are 'a' and 'l', and in English they are 'e', 't', and 'a'. If the message is encrypted with the substitution cipher, then the encrypted letters will have the same statistic as their original letters.

Throughout history, people was sure that hiding the algorithm of a cryptosystem would ensure the privacy. However, history tells us that this is not a good basis for security. Many algorithms were revealed even when they were hidden, such as the historical *Enigma*, and the more modern example of *A5/1*. There is an important principle for a cryptanalysis called *Kerchoff's Principle*.

> *The security of the encryption scheme must depend only on the secrecy of the key, and not on the secrecy of the algorithm.*

This principle is the first starting point for cryptanalysis. Analysis of a cryptosystem begins with the assumption that the encryption algorithm is known.

3.1 Introduction

3.1.1 Attack Scenarios

An adversary, **E***ve*, can potentially have one of, but is not limited to, the following four scenarios of an attack.

- *Ciphertext only attack.* In this scenario **E***ve* has only the ciphertext and tries to analyse it to receive some information from it. It could be a proof or a disproof for some hypothesis stated, or even the secret key recovering. This attack is the most common to think about when one is talking about code breaking.
- *Known plaintext attack.* **E***ve* knows both the ciphertext and the corresponding plaintext. This is the most common scenario in modern cryptanalysis.
- *Chosen plaintext attack.* In this case **E***ve* chooses the messages to be sent, and then gets the corresponding ciphertext.
- *Chosen ciphertext attack.* In this scenario **E***ve* has temporary access to the decryption device, and chooses herself ciphertext to be decrypted receiving the corresponding plaintext.

3.1.2 Success Criteria

The results of cryptanalysis could be one of the following.

- *Total break*, (or a *key-recovering* attack), when the secret key is recovered, and then the rest of the message is known as well. This is the most desirable result, but could sometimes be difficult to achieve.
- *Partial break*, which helps to reduce the search space for the secret key.
- *Information deduction*, which allows **E***ve* to get some partial information about the plaintext.
- A *distinguishing algorithm*, means that we can show that the corresponding cipher has a significant bias, when comparing with a *perfect* cipher. When applied to stream ciphers, the keystream is analysed. In this case we say a distinguisher can recognize whether the keystream

comes from the cipher or is a random sequence. For example, **A***lice* wants to send to **B***ob* one of two pictures X or Y. The keystream can be achieved from the plaintext (X or Y) and the known ciphertext. Thus, **E***ve* can take the picture X and test, whether the guess is correct or not, applying a distinguisher for that cipher to the keystream. Other attack scenarios can be applied. Section 3.4 gives a more detailed overview and analysis of these attacks.

3.1.3 Complexity Issues

When we are talking about cryptanalysis of a cipher, we often compare attacks. This comparison is basically depends on the complexity of an attack. Here three categories can be examined.

- *Time complexity*, C_T, is the number of simple operations that have to be performed to complete the attack, in average. We say that an attack is successful if C_T is less than the complexity of a *brute-force attack*.
- *Memory complexity*, C_M, is the amount of operation and storage memory required to fullfil the attack.
- *Data complexity*, C_D, is the amount of data (keystream, ciphertext) that is required for the attack.

We assume standard inequalities of these complexities

$$\begin{aligned} C_T &\leq C_D, \\ C_T &\leq C_M. \end{aligned} \tag{3.1}$$

Sometimes, we distinguish between two phases of an attack: the *precomputation* and the *evaluation* phases. For the precomputation phase, time complexity larger or close to the complexity of the brute-force attack can be allowed.

In academic sense, a cipher is said to be "*broken*" if it does not satisfy the advertised *security level*. The *security level* is often chosen to be the complexity of a brute-force attack, i.e., time complexity around 2^l, where $l = \log_2 |\mathcal{K}|$ is the size of the secret key in bits.

To show the complexity size, the O-notation is frequently used.

Definition 3.1 (O-notation [CLRS01]): For a given function $g(n)$, we denote by $O(g(n))$ the set of functions $\{f(n)\}$ such that:

$$O(g(n)) = \{f(n) : \exists c_1, c_2, n_0 > 0 \Rightarrow 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0\}. \tag{3.2}$$

□

If for all $n \geq n_0$ the function $f(n)$ is equal to $g(n)$ within a constant factor, then we say that $g(n)$ is an *asymptotically tight bound* for $f(n)$.

3.2 Generic Attacks

Independently of how good an encryption algorithm is, there always exists a set of attacks, known as *generic attacks*. These attacks can be applied to *any* cipher, which is represented as a *black box*.

3.2.1 Brute-Force Attack

This is perhaps the simplest to launch. Assume the size of the secret key K is l bits. Then the total key space is 2^l different keys.

Definition 3.2 (Brute-force Attack): In a *brute-force attack* (or an *exhaustive search*) the intruder tries all possible keys from the key space, which is of size 2^l. The average time complexity is

$$C_T = 2^{l-1}, \tag{3.3}$$

operations, where one operation is the verification test function for one key. □

This attack is important since its complexity is often the threshold in academic sense for success in cryptanalysis of a cipher.

3.2.2 Time-Memory Trade-Offs

Most *time-memory trade-off attacks (TMTO)* are based on the so-called *birthday paradox*.

3.2.2.1 Birthday Paradox

EXAMPLE *3.1 (Birthday Paradox):* Let us have 23 people in a room. The chance that two of them have birthday the same day is around 57%! □

We have started this section with an example, which fake our intuition and requests a proof. However, pure calculations will show that it is true.

Theorem 3.1 (Birthday Paradox): Let us have n random i.i.d. variables $X_1, X_2, \ldots, X_n$ drawn from a discrete uniform distribution with range $[1, d]$. Let $p(n, d)$ be the probability that at least two random variables have the same value. Then

$$p(n,d) = \begin{cases} 1 - \prod_{k=1}^{n-1}(1 - \frac{k}{d}), & 1 < n \leq d, \\ 1, & n > d. \end{cases}$$

$$p(n,d) \approx 1 - e^{-(n(n-1))/2d}. \tag{3.4}$$

□

For the example above, we can calculate the value of $p(23, 365)$, which is around 0.57.

A direct application of this paradox is the problem of finding collisions for hash functions. If an algorithm produces hash tags of size n bits, then the number of tags we need to observe before we can find a collision is around $1.2\sqrt{2^n}$, although the total number of possible tags is 2^n.

3.2.2.2 Basic Time-Memory Trade-Off

Let $S_t \in \mathcal{S}$ be the state of a stream cipher at time t, and let the total number of different possible states be $|\mathcal{S}| = 2^s$. Each time one bit of the keystream is produced, as shown in the model of a *binary* stream cipher in Figure 3.2.

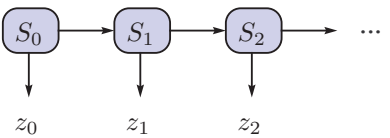


Figure 3.2: A model of a binary stream cipher.

A basic exhaustive attack on the internal state requires at least s (because of the size of the internal state) consecutive bits to be able to recognize the internal state uniquely. However, this attack would take time $O(2^s)$.

Assume that we have generated 2^r independent internal states and for each state we record the first s bits of the keystream. Let us call this list as L_r. Assume a given keystream from an unknown state S_0 of length $2^m + s - 1$, so that we can collect 2^m overlapping s bit sequences from that keystream. Call this list L_m. Now we look at the two lists, one is precomputed of size 2^r with known internal states, and the other one is of size 2^m generated from an unknown internal state. The lists contain s bit sequences. If we can find an s bit sequence that appears in both lists, then the internal state of the given keystream at the corresponding time will be the same as in the list L_r. It can give us the possibility to recover the state S_t of the cipher at some time t. By backward reversing, the initial state S_0 can be achieved.

From the birthday paradox, for such a match to be likely we need that $m+r \approx s$. This attack has the time complexity $O(rs2^m)$ – we need to test 2^m length r sequences, and one logarithmical search in the list L_r takes time r; and memory complexities $O(s2^r)$ – for storing the list L_r of s bit sequences. Thus, if we take $m = r \approx s/2$, the time and memory attack will approximately be $O(2^{s/2})$, which could be much faster than the exhaustive search.

Other examples of TMTO attacks with different scenarios can be found in, e.g., [Bab95, BS00, BSW00, Mih96, Saa02, Gol97a].

3.2.3 Relationship Between the Size of the Key, the IV, And the Internal State

Because of the basic TMTO attack presented in the previous section, it has always been assumed that to prevent tradeoff attacks the internal state size must be at least twice as large than the key size l. Then, the basic TMTO attack would have complexity at least $O(2^l)$, which is comparable to the exhaustive search.

Recently, Jin Hong and Palash Sarkar published a paper [HS05] where they discuss the meaning of IV in stream ciphers, and propose a new model for a stream cipher. Let the key be of l bits and let the IV be of v bits. The attacker observes 2^d frames, uses precomputation time 2^p, and mounts an online attack with time 2^t and memory 2^m to recover the secret key K of one frame. Then, the new TMTO attack would satisfy the following constraints.

$$\begin{cases} p + d = l + v, \\ t \geq 2d, \\ t + m = l + v. \end{cases} \tag{3.5}$$

Suppose we generate keystream sequences for 2^m random pairs Key / IV and store them in a list. Then we observe 2^d keystreams that we are trying to break. From the birthday paradox it follows that one of these keystreams will be broken when $m = d = (l + v)/2$. It basically means that the size of an IV v should be around l bits, otherwise the attack will be faster than the brute-force. However, the claims from J. Hong and P. Sarkar are discussible, as their model is different from the standard one.

3.3 Hypothesis Testing

Hypothesis testing is an important tool in linear cryptanalysis. Many scenarios in cryptanalysis are based on methods of hypothesis testing. In this section we introduce general view on this topic, introduce the notation and give standard definitions. We also present different methods for hypothesis testing, followed by some simple examples.

3.3.1 Basic Definitions

If P is a distribution over some finite domain $\mathcal{X}$ and x is an element of $\mathcal{X}$, then by $P(x)$ we denote probability mass of x according to P. For any subset $S \subseteq \mathcal{X}$ we define $P(S)$ as $P(S) = \sum_{x \in S} P(x)$. When two distributions P_0 and P_1 are given one can consider a distance between them, measured in two different ways.

Definition 3.3 (Statistical and Variational Distances): Let P_0 and P_1 be two distributions over the domain $\mathcal{X}$. We define two distances between them.

(i) The *statistical distance* between P_0 and P_1, denoted by $|P_0 - P_1|$, is defined as

$$|P_0 - P_1| = \frac{1}{2} \sum_{x \in \mathcal{X}} |P_0(x) - P_1(x)|. \tag{3.6}$$

(ii) The *divergence* or *variational distance* between P_0 and P_1, denoted by $\rho(P_0||P_1)$, is defined as

$$\rho(P_0||P_1) = \sum_{x \in \mathcal{X}} P_0(x) \log \frac{P_0(x)}{P_1(x)}. \tag{3.7}$$

□

We note that the statistical distance is always between 0 and 1. However, the distance to the uniform distribution is always between 0 and 0.5.

Lemma 3.2: For two distributions P_0 and P_1, there exist a set $S \in \mathcal{X}$ such that

$$|P_0 - P_1| = P_0(S) - P_1(S). \tag{3.8}$$

This set can be defined as $S = \{x \in \mathcal{X} : P_0(x) \geq P_1(x)\}$. □

The important relation between divergence and statistical distance between two distributions is given by the following lemma.

Lemma 3.3 (Lemma 12.6.1 from [CT91]):

$$\rho(P_0||P_1) \geq \frac{2}{\ln 2} |P_0 - P_1|^2. \tag{3.9}$$

□

Let $X_1, X_2, \ldots, X_n$ be n i.i.d. random variables from $\mathcal{X}$ with distribution P, and $x_1, x_2, \ldots, x_n$ be a realization. The vector $\mathbf{X}^n = (X_1, X_2, \ldots, X_n)$ is an *n-dimensional random vector*, and $\mathbf{x}^n = (x_1, x_2, \ldots, x_n)$ is the *sample*, where n is called the *size of the sample*.

Definition 3.4 (Type, $P_{\mathbf{x}}$): The *type* $P_{\mathbf{x}}$ (or *empirical probability distribution*) for the sequence $\mathbf{x}^n$ is the relative proportion of occurrences for each symbol in $\mathcal{X}$, i.e.,

$$P_{\mathbf{x}}(a) = \frac{\text{number of times } a \text{ occurs in } \mathbf{x}^n}{n}, \quad \forall a \in \mathcal{X}. \tag{3.10}$$

□

3.3.2 Hypothesis Testing in General

In a hypothesis testing problem one has the *null hypothesis*

$$H_0 : \theta \quad \mathcal{R}_0 \quad \theta_0, \tag{3.11}$$

where

- θ is a generic *parameter* of interest of a random variable $\mathbf{x}^n$, e.g., the *mean* μ, variance σ^2 in the one sample case; differences $\mu_1 - \mu_2$, ratio σ_1^2/σ_2^2 in the two sample case.
- θ_0 is a *conjectured value* of the parameter θ in the null-hypothesis. In the two sample case for the mean it is usually zero, as we are trying to detect *any statistically significant* difference between the two groups, at some predetermined significance level p_α. For the ratio of variances it is usually one, to test for equivariance.
- $\mathcal{R}_0$ is the relation between θ and θ_0, such as $=, \leq, \geq$.

Once a suitable *random sample* $\mathbf{x}^n$ is selected, the observed data can be used to compute a *point estimate* $\hat{\theta}$, that *approximates* the parameter θ above. The fundamental question is "*At some pre-determined confidence level (error probability)* p_α*, does the sample estimator* $\hat{\theta}$ *provides sufficient experimental evidence to reject the null hypothesis that the parameter* θ *is in the relation* $\mathcal{R}_0$ *to the fixed value* θ_0?" If not, then we support an *alternative hypothesis* H_1

$$H_1 : \theta \quad \mathcal{R}_1 \quad \theta_0. \tag{3.12}$$

The complete set of possible values of the sample $\mathbf{x}^n$ is of size $|\mathcal{X}|^n$. One can define the *decision rule* δ as a function that decides which hypothesis is most likely to be true.

$$\delta(\mathbf{x}^n) = \begin{cases} H_0, & \text{if } \mathbf{x}^n \in A_c \\ H_1, & \text{otherwise,} \end{cases} \tag{3.13}$$

where the set A_c is called the *acceptance region*.

Related to the region of acceptance A_c one can distinguish error probabilities of two types:

		TRUTH	
		H_0 is True	H_0 is False
DECISION	Retain H_0	Correct Retention	Type II Error
	Reject H_0	Type I Error	Correct Rejection

If we compare a type I error probability to a *false positive alarm* ("an alarm without a fire"), a type II error is a *false negative alarm* ("a fire without an alarm"). The probabilities are as follows

$$\begin{aligned} \Pr\{\text{type I error}\} &= p_\alpha, \\ \Pr\{\text{type II error}\} &= p_\beta. \end{aligned} \tag{3.14}$$

The probability of *not* making a type I error is called *confidence level* (p_α). The probability of *not* making a type II error is called *power level* (p_β). Thus,

$$\begin{aligned} \Pr\{\text{avoiding a type I error}\} &= 1 - p_\alpha, \\ \Pr\{\text{avoiding a type II error}\} &= 1 - p_\beta. \end{aligned} \tag{3.15}$$

Conventional levels for *confidence* are $0.90, 0.95,$ and 0.99. Conventional levels for *power* are $0.80, 0.90,$ and 0.95.

To test H_1 against H_0 one has to fix the level of testing, i.e., the error probability threshold p_α. We *convert the test statistics to a p-value* by placing the test statistic $\hat{\theta}$ on its appropriate probability distribution and determine the area under the curve beyond. Therefore, the decision rule δ is then simplified to

$$\delta(\mathbf{x}^n) = \begin{cases} H_0, & \text{if } p > p_\alpha, \\ H_1, & \text{if } p \leq p_\alpha. \end{cases} \tag{3.16}$$

Thus, a hypothesis testing is a combination of four steps:

1. Define the null H_0 and alternative H_1 hypothesis.
2. Decide on the significance level p_α (only in fixed-level testing).
3. Calculate a test statistic and compare to the probability distribution for the sake of deriving a probability statement.
4. Make the decision of acceptance between two hypotheses.

In Appendix A a set of classical techniques for hypothesis testing, the most useful in cryptanalysis, is given. They are as follows.

(i) One sample point inference (in A.1): *z-statistics, t-statistics.*

(ii) One sample tests with multiple parameters (in A.2): *chi-square test, Kolmogorov-Smirnoff test.*

(iii) Convergence in distribution (in A.3): *information-theoretical approach, through the relation to χ^2 test for γ-flat distributions.*

3.4 Distinguishing Attacks

One of the possible tools to analyse a stream cipher is the *linear statistical distinguisher* approach introduced by J. Golić at Asiacrypt'94 [Gol94], and later also used at, e.g., [Gol00, Gol99, Gol04]. His work contains the basic notion, the basic mathematical results, a method for finding distinguishers called Linear Sequential Circuit Approximation (LSCA), as well as their applications to many known types of stream ciphers at that time. The LSCA method also was previously proposed in [Gol93].

This approach has inspired people to perform attacks on ciphers like SOBER [EJ02a], SNOW [WBC03], and can be regarded as a generalisation of linear cryptanalysis [Mat94].

The purpose of a distinguishing attack is to provide evidence that the generated keystream sequence is not completely random. Informally, we build a distinguisher for a generator X, which can be described as a black box that takes a sequence of symbols as input and produce one out of two answers, either "*The sequence was generated by generator* X" or "*The sequence is completely random*". If the distinguisher can give us correct answers much more often than pure guessing, the attack is successful.

In order to show the usefulness of a distinguishing attack, consider a situation with unknown plaintext for which we would like to give an answer to the following question: "*Does the ciphertext correspond to a given plaintext or not?*" To answer the question, we would xor the ciphertext to the plaintext we want to test against, obtaining a sequence that could be the correct keystream sequence (if we guessed the plaintext correctly). We then feed this sequence as input to the distinguisher. If the distinguisher gives a positive answer, our guessed plaintext was probably the correct one. Otherwise, if the distinguisher tells us that the input sequence was purely random, we probably made a wrong guess about the plaintext. These arguments require some independence assumptions that we do not consider in detail.

Finally note, a distinguishing attack is not as strong as a key-recovery attack, but can provide some undesired information leakage to the adversary. Note that a key-recovery attack is also a distinguishing attack, so if we want to make things simple we can simply state that a good stream cipher should be resistant to distinguishing attacks. The strength of this required resistance, i.e., the required computational complexity, memory, and length of the keystream sequence for a successful distinguisher, is an issue of debate. For example, an n bit block cipher used in any of the most common modes of operation can be distinguished from random using around $2^{n/2}$ output blocks. Note that our model assumes a fixed secret key for all produced keystream. However, if keystream from many different keys are available, there might exist a stronger attack of a different kind.

3.4.1 General Idea and Scenario

The structure shown in Figure 2.8 is also called *an additive stream ciphers*. The keystream generator X accepts a short secret key K and generates a long keystream $\mathbf{z} = z_1, z_2, \ldots$ called as keystream. Encryption and decryption on both transmitter and receiver sides are done similarly. The ciphertext $\mathbf{c} = c_1, c_2, \ldots$ is obtained as the xor of the keystream $\mathbf{z}$ and the plaintext $\mathbf{m} = m_1, m_2, \ldots$, i.e., $\mathbf{c} = \mathbf{m} \oplus \mathbf{z}$.

In most attack scenarios on stream ciphers we assume that a plaintext and the corresponding ciphertext of an appropriate length n are given, i.e., we deal with *a known plaintext attack*. One could sometimes consider stronger assumptions, like chosen plaintext attacks, but on an additive stream cipher (like Scream) this is equivalent to a known plaintext attack. Observe that the bit-wise xor of plaintext and ciphertext gives the adversary the keystream sequence, so when we consider attacks on additive stream ciphers, we generally just assume the keystream sequence to be known. In Figure 3.3 the typical scenario for a linear distinguishing attack is shown.

Known:	Generator X is used
Given:	The ciphertext $\mathbf{c}$, and a *possible* message $\mathbf{m}'$
Question:	Is the original message was $\mathbf{m} = \mathbf{m}'$ or something else?
Solution:	Apply the distinguisher D_X to the sequence $\mathbf{z}' = \mathbf{c} \oplus \mathbf{m}'$: if $D_X(\mathbf{z}') =$ "CIPHER" then $\mathbf{m} = \mathbf{m}'$, otherwise $\mathbf{m} \neq \mathbf{m}'$.

Figure 3.3: A linear distinguishing attack scenario.

To construct a distinguisher, one basic idea is to introduce linear approximations of all nonlinear operations in a specific "path" of the cipher. The path should be such that it connects some known values, which in this case must be keystream symbols. If the linear approximation is true, this leads to a linear relationship L among the known keystream symbols $\mathbf{z}$. That linear relation is usually time-invariant, and can be expressed as

$$L_t(\mathbf{z}) = \sum_{i=1}^{|I|} z_{t+I(i)} \quad \forall t, \tag{3.17}$$

where I is some fixed array of integer numbers. On the other hand, if the linear approximation is not true, we can often think of the error introduced by the linear approximation that behaves as a truly random noise. In summary, some linear combination of keystream symbols corresponding to the linear relationship discussed above can be viewed as a sample from a very

noisy (but not uniform) distribution $P_{\mathcal{C}}$, also called as the *noise distribution*. By collecting many such samples, we can eventually distinguish the distribution they are drawn from, from the *random distribution* $P_{\mathcal{R}}$, which is usually the uniform distribution $P_{\mathcal{U}}$. This results in a successful distinguishing attack, in the sense that it shows that the samples are not from a truly random generator. A typical structure of a distinguisher is shown in Figure 3.4.

Known:	Generator X is used, and the set of integers I is known. The *noise* $P_{\mathcal{C}}$ and *random* $P_{\mathcal{R}}$ distributions are also known.
Given:	The keystream $\mathbf{z}'$, which is *possibly* from X
Question:	Is $\mathbf{z}'$ from X?
Solution:	1. Construct the type $P_{\mathbf{x}}$ (see Def. 3.4) from the following sequence of *samples*: $(L_1(\mathbf{z}'), L_2(\mathbf{z}'), L_3(\mathbf{z}'), \ldots)$. 2. Use statistical methods to decide: · if $P_{\mathbf{x}} \sim P_{\mathcal{C}}$ then output $\leftarrow$ "`CIPHER`": $\mathbf{z}'$ is from X, · if $P_{\mathbf{x}} \sim P_{\mathcal{R}}$ then output $\leftarrow$ "`RANDOM`": $\mathbf{z}'$ is not from X.

Figure 3.4: The typical structure of a distinguisher.

3.4.2 Assumptions

Because the model of a distinguishing attack is idealised and perfect, it cannot include all particulars and dependencies that we meet in real life. Therefore, a set of standard assumptions are usually accepted, and they are described as follows.

1. We assume that after each approximation of a nonlinear part of a cipher we introduce a new *independent noise random variable*. However, if two parts of the cipher are approximated then in the real life these two new noise variables will be dependent, since they both come from the same source (the cipher). Usually, these kinds of dependencies are rather small, and often can be discarded from the model of the attack. Thus, in the model we usually treat new random variables as *independent*.

2. Samples are collected from the keystream sequence, and they form a type (an empirical distribution). We assume that *all samples are independent*. However, this is not true in the real life. We think that we sample from a local distribution $P_{\mathcal{N}}$, but the samples are, indeed, dependent. If, for example, at time t we approximate some parts A_0, A_1, and A_2, and in time $t+1$ approximated parts are A_1, A_2, and A_3 – obviously, the two consecutive samples can be dependent.

However, this assumption is usual in linear cryptanalysis as well. Between two consecutive samples their dependency is determined by the inherent operations of the cipher, which is supposed to scramble the information[1].

3. If the distinguisher consists of a set of subdistinguisher, we treat these subdistinguishers (introduces later in Section 3.4.7) as independent as well.

3.4.3 Distinguishing via Hypothesis Testing

A linear distinguishing attack is usually based on a classical hypothesis testing. Thus, one is given a type $P_{\mathbf{x}}$ constructed from n independent samples $x_i \in \mathcal{X},\ i = 1, \ldots, n$.

We introduce the notation $P_{\mathbf{x}} \leftarrow P$, meaning that the considered samples are drawn according to P. The samples are drawn either according to $P_{\mathcal{C}}$ or according to $P_{\mathcal{R}}$ and one has to decide which is the case, i.e., there exist two hypothesis:

$$\begin{cases} H_{\mathcal{C}}: & P_{\mathbf{x}} \leftarrow P_{\mathcal{C}}, \\ H_{\mathcal{R}}: & P_{\mathbf{x}} \leftarrow P_{\mathcal{R}}. \end{cases} \tag{3.18}$$

One defines a *decision rule*

$$\delta : \mathcal{X}^n \rightarrow \{H_{\mathcal{C}}, H_{\mathcal{R}}\}, \tag{3.19}$$

which constructs a type $P_{\mathbf{x}}$ from the n samples, and defines what should be the guessed hypothesis for any possible $P_{\mathbf{x}}$. Associated to this decision rule the error probabilities are:

$$\begin{aligned} p_\alpha &= \Pr\{\delta(P_{\mathbf{x}}) = H_{\mathcal{R}} | P_{\mathbf{x}} \leftarrow P_{\mathcal{C}}\}, \\ p_\beta &= \Pr\{\delta(P_{\mathbf{x}}) = H_{\mathcal{C}} | P_{\mathbf{x}} \leftarrow P_{\mathcal{R}}\}. \end{aligned} \tag{3.20}$$

When the *a priori* probabilities $\Pr\{P_{\mathbf{x}} \leftarrow P_{\mathcal{C}}\}$ and $\Pr\{P_{\mathbf{x}} \leftarrow P_{\mathcal{R}}\}$ are known, the *total error probability* is then calculated as

$$P_e = \Pr\{P_{\mathbf{x}} \leftarrow P_{\mathcal{C}}\} \cdot p_\alpha + \Pr\{P_{\mathbf{x}} \leftarrow P_{\mathcal{R}}\} \cdot p_\beta. \tag{3.21}$$

A common characteristic of a distinguisher is the value of the *bias*, which is the distance

$$\epsilon = |P_{\mathcal{C}} - P_{\mathcal{R}}|. \tag{3.22}$$

For a fixed number of samples n, the error probability decreases as the *bias* increases.

[1] In the case when two consecutive samples are very much dependent one can skip a few samples before accepting one, making the dependency as small as necessary.

REMARK: When we deal with distinguishers, people often say that a type $P_{\mathbf{x}}$ is constructed from n *samples* $x_1, x_2, \ldots, x_n$, whereas in hypothesis testing it is *a sample* $\mathbf{x}^n$ *of size* n.

3.4.4 Distinguisher and Advantage

Definition 3.5 (Distinguisher): A *distinguisher* D is a probabilistic function $D : \mathcal{X}^n \rightarrow \{H_{\mathcal{C}}, H_{\mathcal{R}}\}$, which for a given stream of n samples from $\mathcal{X}$ decides between the following two hypotheses:

- $H_{\mathcal{C}}$ is when the sequence is actually produced from the given cipher.
- $H_{\mathcal{R}}$ is when the given sequence is completely random.

□

Basically, a distinguisher is an algorithm which exploits hypothesis testing, and has probabilities of errors p_α and p_β. Note that for *any* stream cipher there always exists a *random distinguisher*.

Definition 3.6 (Random Distinguisher): The *random distinguisher* gives the answers $H_{\mathcal{R}}$ and $H_{\mathcal{C}}$ with probabilities $1/2$, independently of the given sequence of samples. I.e., its error probabilities are $p_\alpha = p_\beta = 1/2$. □

This simple distinguisher gives the correct answer with probability $1/2$. Of course, such a distinguisher is useless. For a distinguisher which provides another probability, we can measure the *advantage* of the distinguisher.

Definition 3.7 (Distinguisher Advantage): The success of a distinguisher D to distinguish a cipher from a random generator is determined by two probabilities:

- The probability p_0 of answering "CIPHER" when the given stream is from the cipher (a correct answer).
- The probability p_1 of answering "CIPHER" when the given stream is from a random generator (an incorrect answer).

The overall ability of a distinguisher D to distinguish two functions is measured by the *advantage*, denoted $\mathtt{Adv}_D$, which is expressed as

$$\mathtt{Adv}_D = |p_0 - p_1|. \tag{3.23}$$

In terms of p_α and p_β, the advantage can also be expressed as

$$\mathtt{Adv}_D = |1 - p_\alpha - p_\beta|. \tag{3.24}$$

□

The advantage is always between 0 and 1. For the random distinguisher its advantage is 0.

3.4.5 The Case When Both Noise and Random Distributions are Known

In this section we consider the case when two distributions $P_{\mathcal{C}}$ (*noise distribution*) and $P_{\mathcal{R}}$ (*random distribution*) over some probability space $\mathcal{X}$ *are known,* and the samples x_i from the given sequence $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ are produced from one of these two distributions. The type $P_{\mathbf{x}}$ is constructed from the samples $\mathbf{x}$. The optimal test between the two distributions is given by the *Neyman-Pearson likelihood test* [CT91].

Definition 3.8 (Neyman-Pearson Optimal Hypothesis Testing): To test $P_{\mathbf{x}} \leftarrow P_{\mathcal{R}}$ against $P_{\mathbf{x}} \leftarrow P_{\mathcal{C}}$ we need to check the *likelihood ratio*

$$\begin{aligned} I &= \rho(P_{\mathbf{x}}||P_{\mathcal{C}}) - \rho(P_{\mathbf{x}}||P_{\mathcal{R}}) \\ &= \sum_{x \in \mathcal{X}} P_{\mathbf{x}}(x) \log_2 \frac{P_{\mathcal{C}}(x)}{P_{\mathcal{R}}(x)}. \end{aligned} \tag{3.25}$$

Then the decision rule is

$$\delta(P_{\mathbf{x}}) = \begin{cases} H_{\mathcal{C}}, & \text{if } I > 0 \\ H_{\mathcal{R}}, & \text{if } I \leq 0 \end{cases} \tag{3.26}$$

□

In this case the probability of error P_e for the decision rule δ given n samples is asymptotically bounded by

$$P_e = p_\alpha \cdot \Pr\{H_{\mathcal{C}}\} + p_\beta \cdot \Pr\{H_{\mathcal{R}}\} \leq 2^{-n \cdot C(P_{\mathcal{C}}, P_{\mathcal{R}})}, \tag{3.27}$$

where $C(P_{\mathcal{C}}, P_{\mathcal{R}})$ is the Chernoff information between the two distributions. The Chernoff information is defined as

$$C(P_{\mathcal{C}}, P_{\mathcal{R}}) = -\min_{0 \leq \lambda \leq 1} \log_2 (\sum_{x \in \mathcal{X}} P_{\mathcal{C}}^{\lambda}(x) P_{\mathcal{R}}^{1-\lambda}(x)). \tag{3.28}$$

For the fixed probability of error P_e and known λ the approximate value of n can be evaluated.

The value of λ is sometimes difficult to obtain. Therefore, in most cases we just take $\lambda = 0.5$ and get an upper bound on the probability of error, which is sufficient for most situations.

In real cryptanalysis work people often operate with the distance between $P_{\mathcal{C}}$ and $P_{\mathcal{R}}$, i.e., $\epsilon = |P_{\mathcal{C}} - P_{\mathcal{R}}|$. The number of samples n required to distinguish with sufficiently high probability of success is roughly

$$n = O(1/\epsilon^2), \tag{3.29}$$

which is frequently used to estimate the complexity of a distinguishing attack.

Assume the random distribution $P_{\mathcal{R}}$ is actually the uniform disribution $P_{\mathcal{R}} = P_{\mathcal{U}}$, and the noise variable $N \sim P_{\mathcal{C}}$ is the sum of k independent variables $N = N_1 + N_2 + \ldots + N_k$, with $\epsilon_i = |P_{\mathcal{R}} - P_{N_i}|, \ \forall i = 1, \ldots, k$. Then we have the following relation.

$$\epsilon = |P_{\mathcal{C}} - P_{\mathcal{R}}| \leq \prod_{i=1}^{k} \epsilon_i. \tag{3.30}$$

3.4.6 Special Case – Binary Distributions

Consider the binary case with some events e and $\overline{e}$. Then we have $P_{\mathcal{R}}(e) = p$, and $P_{\mathcal{C}}(e) = p(1+\epsilon)$, for some valid values of p and ϵ. Then the required number of samples for a successful distinguisher is given by the following lemma.

Lemma 3.4 (From [MS01]): Let an event e happen in $P_{\mathcal{R}}$ with probability p and in $P_{\mathcal{C}}$ with probability $p(1+\epsilon)$. Then for small p and ϵ,

$$n = O(1/p\epsilon^2) \tag{3.31}$$

samples suffice to distinguish $P_{\mathcal{R}}$ from $P_{\mathcal{C}}$ with a constant probability of success. □

Assume we have k noise variables. Then the bias of their sum is given by the following lemma from M. Matsui.

Lemma 3.5 (Piling-Up Lemma [Mat94]): Let us have k independent binary random variables $X_1, X_2, \ldots, X_k$, for which $\Pr\{X_i = 0\} = \frac{1}{2}(1+\epsilon_i), \ \forall i = 1, \ldots, k$. Then we have

$$\Pr\{\sum_{i=1}^{k} X_i = 0\} = \frac{1}{2}\left(1 + \prod_{j=1}^{k} \epsilon_j\right). \tag{3.32}$$

□

3.4.7 X-Distinguisher for Unknown Noise Distribution

In some primitives a typical situation is when the cipher has key-dependent nonlinear blocks in its algorithm. In this case, the distribution of the noise $P_{\mathcal{C}}$, when approximating these nonlinear blocks, is unknown. Therefore, the collected type can be either the random distribution $P_{\mathcal{R}}$, or the unknown

distribution $P_{\mathcal{C}}$. The distributions $P_{\mathcal{R}}$ and $P_{\mathcal{C}}$ will be at some unknown distance $\epsilon = |P_{\mathcal{R}} - P_{\mathcal{C}}|$.

Approximation of nonlinear parts of the cipher can be done in many ways. For each approximation, the distribution of the noise variable is (in most cases) different. The distance ϵ also varies, but its typical value can be estimated by simulations.

Let us introduce a *distinguisher with* **U***nknown noise distribution*, U-distinguisher, defined as follows.

Definition 3.9 (U-Distinguisher): Assume that two distributions $P_{\mathcal{R}}$ and $P_{\mathcal{C}}$ have distance $\epsilon = |P_{\mathcal{C}} - P_{\mathcal{R}}|$, and one collect n samples for the type $P_{\mathbf{x}}$ either from $P_{\mathcal{R}}$ or $P_{\mathcal{C}}$. The decision rule δ for the U-distinguisher is defined as follows

$$\delta(P_{\mathbf{x}}) = \begin{cases} H_{\mathcal{C}} : \text{“}P_{\mathbf{x}} \leftarrow P_{\mathcal{C}}\text{”}, & \text{if } |P_{\mathbf{x}} - P_{\mathcal{R}}| \geq \epsilon_{\text{thr}}, \\ H_{\mathcal{R}} : \text{“}P_{\mathbf{x}} \leftarrow P_{\mathcal{R}}\text{”}, & \text{if } |P_{\mathbf{x}} - P_{\mathcal{R}}| < \epsilon_{\text{thr}}, \end{cases} \tag{3.33}$$

for some threshold ϵ_{thr}, such that $0 < \epsilon_{\text{thr}} < \epsilon$. □

Note that for such a distinguisher it is not necessary to know the distribution $P_{\mathcal{C}}$, but only the distribution $P_{\mathcal{R}}$, the distance $\epsilon = |P_{\mathcal{R}} - P_{\mathcal{C}}|$, and the decision threshold ϵ_{thr}, which is usually taken as $\epsilon_{\text{thr}} = \epsilon/2$.

Unfortunately, the value of ϵ is unknown for us, but its distribution can be estimated by simulations. Therefore, an *e***X***tended distinguisher* is introduced.

Definition 3.10 (X-Distinguisher): This *extended distinguisher* is a construction from an appropriate number m of *subdistinguishers* $\texttt{SD}_i, i = 1, 2, \ldots, m$.

Each subdistinguisher $\texttt{SD}_i$ uses a randomly chosen linear approximation of the nonlinear parts of the cipher and the key K is the same for all subdistinguishers. Let $P_{\mathcal{C}}^{(i)},\ i = 1, 2, \ldots, m$ denote the corresponding unknown noise distributions, and let $\epsilon_i = |P_{\mathcal{R}} - P_{\mathcal{C}}^{(i)}|,\ i = 1, 2, \ldots, m$, which are also unknown. From the keystream $\mathbf{z}$, each $\texttt{SD}_i$ constructs its own type $P_{\mathbf{x}}{}^{(i)}$, according to its corresponding linear combination of $\mathbf{z}$. We denote the distances between the types and the random distribution as $\tau_i = |P_{\mathbf{x}}{}^{(i)} - P_{\mathcal{R}}|$.

For some appropriately chosen *threshold* value ϵ_{thr} each $\texttt{SD}_i$ is a U-distinguisher with the decision rule

$$\texttt{SD}_i(Z) = \begin{cases} H_{\mathcal{C}} : \text{“}\texttt{OUTSIDE}\text{”}, \text{ if } \tau_i = |P_{\mathbf{x}}{}^{(i)} - P_{\mathcal{R}}| \geq \epsilon_{\text{thr}}, \\ H_{\mathcal{R}} : \text{“}\texttt{INSIDE}\text{”}, \text{ otherwise.} \end{cases} \tag{3.34}$$

We use the notation

$$\epsilon_{\max} = \max\{\epsilon_1, \epsilon_2, \ldots, \epsilon_m\}. \tag{3.35}$$

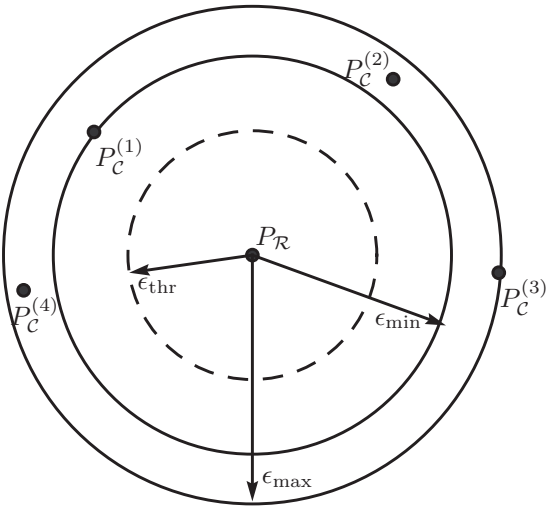


Figure 3.5: The noise distribution $P_{\mathcal{C}}$ is unknown, but on some distance $\epsilon = |P_{\mathcal{R}} - P_{\mathcal{C}}|$ from the known random distribution $P_{\mathcal{R}}$.

The value of ϵ_{thr} should be chosen such that for the m subdistinguishers we have

$$\Pr\{\epsilon_{\text{thr}} \leq \lambda \epsilon_{\text{max}}\} \text{ close to } 1, \tag{3.36}$$

for some fixed $\lambda \in (0, 1)$ (as we mentioned before, the usual value for λ is $1/2$). The overall X-distinguisher is then defined as follows,

$$\delta(\mathbf{z}) = \begin{cases} \text{“}\texttt{CIPHER}\text{”}, & \text{if } \texttt{SD}_i(Z) = \text{“}\texttt{OUTSIDE}\text{”} \text{ for } \textsc{At Least One } i = 1, 2, \ldots, m \\ \text{“}\texttt{RANDOM}\text{”}, & \text{if } \texttt{SD}_i(Z) = \text{“}\texttt{INSIDE}\text{”} \text{ for } \textsc{All } i = 1, 2, \ldots, m. \end{cases} \tag{3.37}$$

□

Figure 3.5 illustrates the case when an X-distinguisher can be applied. If the given stream is completely random, then *all types* $P_{\mathcal{C}}^{(1)}, P_{\mathcal{C}}^{(2)}, \ldots, P_{\mathcal{C}}^{(m)}$ *should be inside* the sphere with radius ϵ_{thr}, otherwise, at least one should be outside of the sphere. We need to try m different linear approximations to ensure that at least for one the distance $|P_{\mathcal{R}} - P_{\mathcal{C}}^{(i)}|$ will be as large as possible. If the stream is from the cipher, then *some* i^{th} *type will be outside* the sphere with radius ϵ_{thr}.

Below we give calculations for the error probabilities of such a distinguisher. For the U-distinguisher, its probability of errors are denoted by

$$\begin{aligned} p^{\epsilon}_{U\alpha}(\epsilon_{\rm thr}) &= \Pr\{H_{\mathcal{R}} | P_{\mathbf{x}} \leftarrow P_{\mathcal{C}}\}, \\ p^{\epsilon}_{U\beta}(\epsilon_{\rm thr}) &= \Pr\{H_{\mathcal{C}} | P_{\mathbf{x}} \leftarrow P_{\mathcal{R}}\}, \end{aligned} \tag{3.38}$$

where the *unknown* distance is $\epsilon = |P_{\mathcal{R}} - P_{\mathcal{C}}|$, and the decision threshold is $\epsilon_{\rm thr}$.

Theorem 3.6: For the X-distinguisher, the probability of error of the first kind is the case when none of the m constructed subdistinguishers break the threshold $\epsilon_{\rm thr}$. This probability equals to

$$\begin{aligned} p_\alpha = \Pr\{\delta(\mathbf{z}) = H_{\mathcal{R}} | P_{\mathbf{x}} \leftarrow P_{\mathcal{C}}\} &= \prod_{i=1}^{m} p^{\epsilon_i}_{U,\alpha}(\epsilon_{\rm thr}) \\ < \min_i \{p^{\epsilon_i}_{U\alpha}(\epsilon_{\rm thr})\} &= p^{\epsilon_{\max}}_{U\alpha}(\epsilon_{\rm thr}). \end{aligned} \tag{3.39}$$

The probability of error of the second kind is

$$p_\beta = \Pr\{\delta(\mathbf{z}) = H_{\mathcal{C}} | P_{\mathbf{x}} \leftarrow P_{\mathcal{R}}\} = 1 - \prod_{i=1}^{m} \left(1 - p^{\epsilon_i}_{U\beta}(\epsilon_{\rm thr})\right). \tag{3.40}$$

□

For the X-distinguisher there are three parameters one has to choose appropriately, m (the number of subdistinguishers), n (the number of samples), and $\epsilon_{\rm thr}$ (the decision threshold for subdistinguishers). The principles of choosing these parameters are as follows. The probability $\Pr\{\lambda \cdot \epsilon_{\max} \geq \epsilon_{\rm thr}\}$ should be very close to 1, and this probability can usually be estimated by simulation. For this purpose we choose a random key K, and make a loop; each step of the loop randomly chooses an approximation functions R for nonlinear blocks of the cipher, and calculate the distance $\epsilon = |P_{\mathcal{C}}(K, R) - P_{\mathcal{R}}|$. Note that the noise distribution depends on the key K and an approximation function R. The set of distances created by the loop allows us to estimate the probability $\Pr\{\lambda \cdot \epsilon_{\max} \geq \epsilon_{\rm thr}\}$ and derive an appropriate value of $\epsilon_{\rm thr}$ (when $\lambda \in (0, 1)$ is fixed).

The remaining part is the analysis of the class of U-distinguishers, in order to receive the relation between error probabilities p_α and p_β, number of samples n, the maximum achievable distance $\epsilon_{\max}$, and the decision threshold $\epsilon_{\rm thr}$.

3.4.8 Analysis of the U-Distinguisher

The number of required samples of such a distinguisher depends on the distance ϵ. The success and error probabilities depend on the number of samples n. To derive the relation between these parameters we need to derive expressions for error probabilities p_α and p_β. The probability of error p_β depends only on $P_\mathcal{R}$ and $\epsilon_{\rm thr}$. However, to find the bound for p_α the value ϵ is important. We use the following Chernoff bound to derive the later expressions.

Theorem 3.7: (Chernoff Bound) Suppose $x_1, \ldots, x_n$ are independent random variables such that for all i, $\Pr\{x_i = 1\} = p$ and $\Pr\{x_i = 0\} = 1 - p$. Let $\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$. Then, for any $\delta > 0$,

$$\begin{aligned} \Pr\{\overline{x} \geq p + \delta\} &\leq \exp(-2n\delta^2), \\ \Pr\{\overline{x} \leq p - \delta\} &\leq \exp(-2n\delta^2). \end{aligned} \tag{3.41}$$

□

Corollary 3.8: Let a random variable $x \in \mathcal{X}$ be distributed as P_0. Let $S \subseteq \mathcal{X}$, and $\Pr\{x \in S\} = p$. Assume that we construct a type $P_{\mathbf{x}}$ from n independent samples $x_i, i = 1, \ldots, n$ distributed as $P_\mathcal{R}$. Then, for any $\delta > 0$, we have

$$\begin{aligned} \Pr\{\text{at least } (p+\delta)n \text{ samples are not in } S\} &\leq \exp(-2n\delta^2), \\ \Pr\{\text{at most } (p-\delta)n \text{ samples are in } S\} &\leq \exp(-2n\delta^2). \end{aligned} \tag{3.42}$$

Proof: Consider the sequence y_i, where $y_i = 1$ if the sample x_i is in S, otherwise $y_i = 0$, i.e., $\Pr\{y_i = 1\} = p$. Obviously,

$$\overline{y} = \{\text{number of samples in } S\}/n, \tag{3.43}$$

therefore, Theorem 3.7 is directly applied. □

Let us for simplicity denote the error probabilities of a U-distinguisher by $p_\alpha = p^{\epsilon}_{U\alpha}$ and $p_\beta = p^{\epsilon}_{U\beta}$, to the end of this section.

Theorem 3.9: Let the random distribution $P_\mathcal{R}$ be a γ-flat distribution (see Def.A.1), for some negligible $\gamma \geq 0$, and the number of samples n is large ($n \gg 35|\mathcal{X}|$). Then, for the U-distinguisher, the error probabilities are asymptotically bounded as follows,

$$p_\alpha < \exp(-2n(\epsilon - \epsilon_{\rm thr})^2). \tag{3.44}$$

$$p_\beta \leq 1 - Q_{\chi^2_{|\mathcal{X}|-1}}(4n\epsilon_{\text{thr}}^2). \tag{3.45}$$

Here $Q_{\chi^2_{|\mathcal{X}|-1}}(x)$ is the cumulative density function for χ^2.

Proof: The proof for the case p_β is straight forward from Appendix A.3.2 and the resulting equation (A.36). Therefore, we only need to prove the case when the hypothesis $H_\mathcal{C}$ is true.

Recall Definition 3.9, the type $P_\mathbf{x}$ is drawn from $P_\mathcal{C}$ but we measure the distance to $P_\mathcal{R}$, and then compare it with the threshold value ϵ_{thr}.

Recall also Definition 3.3, for any two distributions $P_\mathcal{C}$ and $P_\mathcal{R}$ there exist a set $S \in \mathcal{X}$ such that $\epsilon = |P_\mathcal{C} - P_\mathcal{R}| = P_\mathcal{C}(S) - P_\mathcal{R}(S)$, where $P_\mathcal{C}(S) = p + \epsilon$ and $P_\mathcal{R}(S) = p$, for some $0 \leq p < 1$. Then, for the U-distinguisher, the success probability is lower bounded as

$$\Pr\{\delta(P_\mathbf{x}) = H_\mathcal{C} | P_\mathbf{x} \leftarrow P_\mathcal{C}\} \geq \Pr\{\text{at least } (p+\epsilon_{\text{thr}})n \text{ samples of } P_\mathbf{x} \text{ are in } S\}. \tag{3.46}$$

This argument is easy to check. Let $(p+\epsilon_{\text{thr}})n$ samples be in S, then we have

$$\begin{aligned} |P_\mathbf{x} - P_\mathcal{R}|_S &= \sum_{x \in S} |P_\mathbf{x}(x) - P_\mathcal{R}(x)| \geq \left| \sum_{x \in S} P_\mathbf{x}(x) - P_\mathcal{R}(x) \right| \\ &= |P_\mathbf{x}(S) - P_\mathcal{R}(S)| = \frac{1}{n}(p+\epsilon_{\text{thr}})n - p = \epsilon_{\text{thr}}, \end{aligned} \tag{3.47}$$

similarly $|P_\mathbf{x} - P_\mathcal{R}|_{\mathcal{X}/S} \geq \epsilon_{\text{thr}}$, which means that the final distance $|P_\mathbf{x} - P_\mathcal{R}|$ is at least ϵ_{thr}, and, according to the decision rule (3.33), the correct null hypothesis $H_\mathcal{C}$ will be accepted. Of course this is not the only case when $H_\mathcal{C}$ is accepted, therefore, inequality (3.46) holds.

Because $P_\mathcal{C}(S) = p + \epsilon$, then from Corollary 3.8 we have

$$\begin{aligned} &\Pr\{\text{at most } ((p+\epsilon) - (\epsilon - \epsilon_{\text{thr}}))n \text{ samples of } P_\mathbf{x} \text{ are in } S\} \\ &\quad \leq \exp(-2n(\epsilon - \epsilon_{\text{thr}})^2), \end{aligned} \tag{3.48}$$

i.e., by taking the compliments of the probabilities,

$$\begin{aligned} &\Pr\{\text{at least } (p+\epsilon_{\text{thr}})n \text{ samples of } P_\mathbf{x} \text{ are in } S\} \\ &\quad \geq 1 - \exp(-2n(\epsilon - \epsilon_{\text{thr}})^2). \end{aligned} \tag{3.49}$$

Thus,

$$\begin{aligned} 1 - p_\alpha &= \Pr\{\delta(P_\mathbf{x}) = H_\mathcal{C} | P_\mathbf{x} \leftarrow P_\mathcal{C}\} \\ &\geq 1 - \exp(-2n(\epsilon - \epsilon_{\text{thr}})^2). \end{aligned} \tag{3.50}$$

I.e.,

$$p_\alpha < \exp(-2n(\epsilon - \epsilon_{\text{thr}})^2). \tag{3.51}$$

□

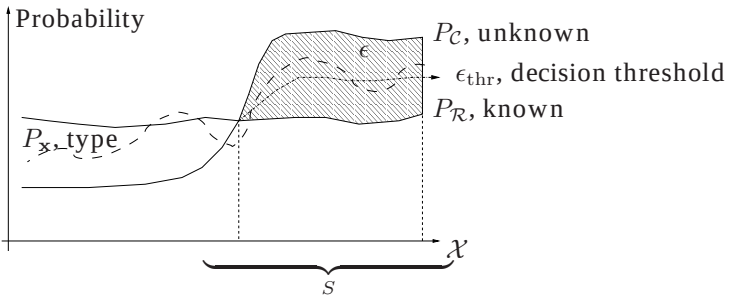


Figure 3.6: U-distinguisher and hypothesis testing.

The idea for the proof in Theorem 3.9 above is illustrated in Figure 3.6. Two distributions $P_{\mathcal{C}}$ and $P_{\mathcal{R}}$, and the set $S \in \mathcal{X}$ is such that $P_{\mathcal{C}}(x) \geq P_{\mathcal{R}}(x)$ for all $x \in S$. The distance $|P_{\mathbf{x}} - P_{\mathcal{R}}|$ is compared with the threshold value ϵ_{thr}, and then the decision is made. The theorem above gives us an upper bound for the error probability. This type of distinguishers is useful when one of two distributions is unknown.

Note that the bounds for the error probabilities p_α and p_β are in a trade-off relation. When p_α grows the probability p_β decreases, and vice versa.

However, the variation of p_β is more sensitive to the choice of parameters. Therefore, when the desired error probability p_α is fixed, we would like to choose parameters according to the following system of inequalities,

$$\begin{cases} n & \geq \frac{|\mathcal{X}|}{2\epsilon_{\text{thr}}^2}, \qquad \leftarrow \textit{to be sure } p_\beta \textit{ is negligible} \\ p_\alpha & < \exp(-2n(\epsilon - \epsilon_{\text{thr}})^2). \end{cases} \tag{3.52}$$

Corollary 3.10: Let $P_{\mathcal{R}} = P_{\mathcal{U}}$ and let $\epsilon = |P_{\mathcal{R}} - P_{\mathcal{C}}| \ll 1$ be very small. If the desired error probability p_α is fixed, then the optimal value for the parameter $0 < \lambda < 1$ is the solution of the equation

$$\lambda^2 \ln p_\alpha + (1-\lambda)^2 |\mathcal{X}| = 0, \tag{3.53}$$

which is

$$\lambda = \frac{|\mathcal{X}| - \sqrt{-|\mathcal{X}| \ln p_\alpha}}{|\mathcal{X}| + \ln p_\alpha}. \tag{3.54}$$

The values n and ϵ_{thr} are then derived as

$$\begin{aligned} \epsilon_{\text{thr}} &= \lambda \cdot \epsilon \\ n &= |\mathcal{X}|/(2\epsilon_{\text{thr}}^2). \end{aligned} \tag{3.55}$$

□

EXAMPLE *3.2 (U-Distinguisher):* Let $|\mathcal{X}| = 256$, and desired probability of error is $p_\alpha = 2^{-10}$. The solution of (3.52) is $\lambda \approx 0.835$.

If, additionally, we assume that $\epsilon = 2^{-50}$, then the threshold will be $\epsilon_{\text{thr}} \approx 0.835 \cdot 2^{-50}$ and the number of samples required is $n \approx 2^{107.52}$.

To be precise, the error probability p_β is calculated as

$$p_\beta = 1 - Q_{\chi^2_{255}}(\underbrace{2^{2+107.52-100.52}}_{512}) \ll 2^{-100}. \tag{3.56}$$

□

3.4.9 On Distinguishers and Resynchronisation

Assume we have a distinguisher D on some cipher. Assume that we are given n samples, when the distance between $P_{\mathcal{R}}$ and $P_{\mathcal{C}}$ is ϵ. Following (3.27), the error probability P_e will be of size around $O(2^{-c \cdot n\epsilon^2})$, up to a constant c in the expression.

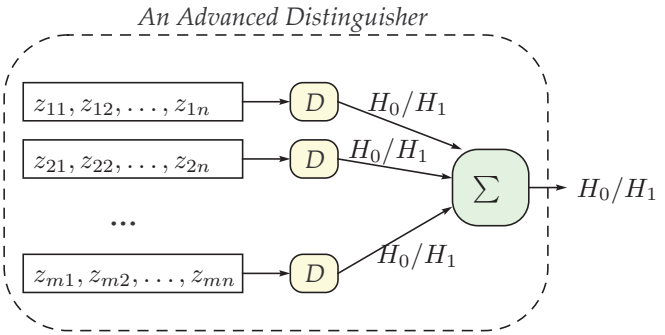


Figure 3.7: An advanced distinguisher as a set of subdistinguishers.

Assume that the probability of error P_e is very close to 1, but the number of samples n cannot be increased due to, for example, a resynchronisation process. However, still, such a distinguisher would have an advantage larger than 0. So we can apply D to m frames, each of size n symbols, instead. Each subdistinguisher gives the correct answer with the probability $1/2 + (1/2 - P_e)$, where $0 \leq P_e \leq 1/2$. Thus, testing a sufficient number of frames m and collecting answers from subdistinguishers, we can make the error probability as small as we wish. In Figure 3.7 the structure of such a distinguisher is shown.

The advanced distinguisher receives the correct answers with the bias $\epsilon' = 1/2 - P_e$. To be short, the number of samples in this case and the number of frames m should be around $1/\epsilon'^2$. Thus, we have:

$$m \approx 1/\epsilon'^2 = \frac{4}{(1-2^{-c\cdot n\epsilon^2})^2}. \tag{3.57}$$

Recall the series expansion of an exponent

$$2^x = e^{x\ln 2} = 1 + x\ln 2 + \underbrace{\frac{(x\ln 2)^2}{2!} + \frac{(x\ln 2)^3}{3!}}_{\text{is always} > 0 \text{ for } \forall x:|x|\leq 1} \ldots \tag{3.58}$$

As we assume that the error probability is large, then it means that $2^{-c\cdot n\epsilon^2}$ is very close to 1, i.e., $-c\cdot n\epsilon^2$ is almost 0. Therefore, in the expansion of the exponent, we can accept the first two terms, and omit the rest as follows.

$$\begin{aligned} m &\approx \frac{4}{(2^{-c\cdot n\epsilon^2}-1)^2} < \frac{4}{(1-c\cdot n\epsilon^2\ln 2 - 1)^2} \\ &\approx 4/((c\ln 2)^2\cdot n^2\epsilon^4) \sim O\left(\frac{1}{n^2\epsilon^4}\right). \end{aligned} \tag{3.59}$$

Assume that each subdistinguisher have time complexity

$$C_T(D) = n\cdot r, \tag{3.60}$$

where $r \geq 1$ is the time for possible "guesses" of some bits, or, perhaps, multiple runs through the keystream during the individual subdistinguisher evaluation. Then, when the accessed number of samples before the resynchronisation is fixed to n, the total time and data complexities of the advanced distinguisher will be as follows.

$$\begin{aligned} C_T &\sim O(m\cdot nr) = O\left(\frac{r}{n\epsilon^4}\right), \\ C_D &\sim O(m\cdot n) = O\left(\frac{1}{n\epsilon^4}\right). \end{aligned} \tag{3.61}$$

EXAMPLE *3.3 (Advanced Distinguisher):* Assume that the bias is $\epsilon = 2^{-50}$. If we would have infinite number of samples, then after $n \approx 2^{100}$ samples the cipher will be distinguished.

Let the cipher is resynchronised after each $n = 2^{80}$ output symbols. Then, the probability of success for our distinguisher would be around 2^{-20}, which is not very good. Let us take $m \approx (n^2\epsilon^4)^{-1} = 2^{40}$ of such frames, then our distinguisher will be successful, but require $m\cdot n = 2^{80+40} = 2^{120}$ samples. □

3.5 Correlation Attacks

Correlation attacks is a huge class of attacks from linear cryptanalysis, the result of which is usually a recovered secret key K. We start this section with possible formulas for estimation of just one bit, and then move to more complex structures.

3.5.1 Bit Estimation

Assume we have an *unknown binary constant generator* $X \in \{0,1\}$, and a binary random variable N with the distribution $\Pr\{N=0\} = p > 1/2$ and $\Pr\{N=1\} = 1-p$. We observe n samples of $X+N$. This scheme can be regarded as a sequence of constants X passing through the *binary symmetric channel (BSC)*, and then the observed sequence is $z_1, z_2, \ldots, z_n$, as shown in Figure 3.8. I.e., we observe the correct value of X with probability p, and wrong with probability $1-p$.

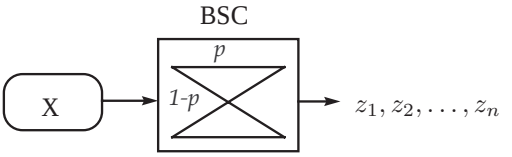


Figure 3.8: Binary symmetric channel.

From the sequence $\mathbf{z}^n$ we would like to estimate the value of X. The likelihood decision rule is as follows. If the number of ones is more than the number of zeros, then we conclude that $X=1$, otherwise $X=0$.

Methods to calculate success and error probabilities for such a decision rule are given in Appendix A.4.1.

3.5.2 Correlation Attacks on LFSRs with Combiners

This classical attack in modern cryptanalysis was introduced by T. Siegenthaler in 1984 [Sie84, Sie85]. Consider the scheme in Figure 3.9.

In this scheme m LFSRs are combined with the Boolean function h, the output of which is the keystream $\mathbf{z}$. This function has to be at least balanced, to have the keystream bits as close to the uniform distribution as possible. If we would apply an exhaustive search attack, then it would take $O(2^{l_1+l_2+\ldots+l_m})$ number of operations, which is quite much.

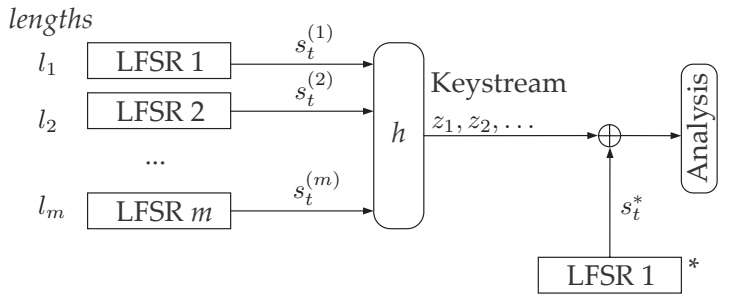


Figure 3.9: Principle of the Siegenthaler's attack.

Siegenthaler suggested to divide the tasks. Let us guess the initial state of the LFSR-1, then, adding the corresponding output stream s_t^* to the keystream we will actually observe the following samples

$$x_t = h(s_t^{(1)}, s_t^{(2)}, \ldots, s_t^{(m)}) \oplus s_t^*. \tag{3.62}$$

We can have two situations in principle:

(a) *The guess is not correct*. In this case the samples x_t will look random, because then s_t^* and $s_t^{(1)}$ are not correlated.

(b) *The guess is correct*. Then, actually, we have $s_t^* = s_t^{(1)}$.

- If $h(\cdot)$ is not a correlation immune function, then x_t *is biased*.
- Otherwise, we will still get a random sequence.

If the function $h(\cdot)$ is not immune against correlation, then we can guess the initial states of the LFSRs just one by one. The complexity of this attack will be $O(\sum_{i=1}^{m} 2^{l_i})$, which is much faster than exhaustive search.

In the case when $h(\cdot)$ has k^{th} order of correlation immunity, then we should guess $k+1$ LFSRs at once, in order to have a biased sample sequence for the right guess. Note also that the number of samples that we need to observe depends on the nonlinearity of the function for x_t. Afterwards, the standard hypothesis testing between two binary distributions is performed (see Section 3.4.6).

3.5.3 LFSR Reconstruction via General Decoding Problem

Meier and Staffelbach in 1988 proposed a slightly different model for the correlation attack [MS88, MS89]. The problem of reconstructing the LFSR

can be viewed as a decoding problem [Sie85, MG90, CS91], and the attack from Siegenthaler is actually the subcase of the *general decoding problem (GDP)*. The proposed scenario is shown in Figure 3.10.

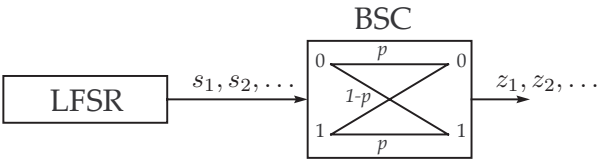


Figure 3.10: The keystream as the noisy output from an LFSR.

In the decoding problem we observe the output $z_1, z_2, \ldots, z_n$ of length n, the noisy version of the LFSR sequence $\mathbf{s}$, and we wish to reconstruct the initial state of the LFSR $s_0, \ldots, s_{l-1}$. There is a set of solutions for this problem, and the best overview of the methods can be found in Fredrik Jönsson's PhD Thesis [Jön02].

Several techniques can be applied, depending on the particular case. When the probability p is very close to $1/2$, then it is hard to recover the state of the LFSR. Earlier ideas allowed to perform this attack when p is quite low. However, the later results allow to operate even when p is large (e.g., $p \approx 0.45$). Obviously, when $p = 1/2$ then this problem has no solution.

For simple coding arguments [Sha49], to recove the LFSR state uniquely, the number n of samples should be around

$$n \approx \frac{l}{1 - h(p)}, \tag{3.63}$$

where $h(p)$ is the *binary entropy function* defined as

$$h(p) = -(p \log_2 p + (1-p) \log_2 (1-p)). \tag{3.64}$$

There exist many algorithms related to the GDP. Most algorithms consist of two phases: *finding parity check equations* or *precomputation phase*, and *decoding algorithm* or *active attack phase*. Algorithms for both parts are selected according to the particular parameters: generating polynomial is of *low weight* or *arbitrary*, the length of the LFSR is *short* or *long*, the accessible length of the sequence is *short* or *long*, and others.

In Appendix A.4.2 and A.4.3 we present best known techniques to recover the state of the LFSR in such scenario.

3.6 Other Attacks

In the previous two sections *distinguishing* and *correlation* attacks were presented more or less in detail. However, there exist many more attacks, and some of them we mention in this section.

3.6.1 Differential Cryptanalysis

The original idea for the *differential cryptanalysis* comes from Eli Biham and Adi Shamir, who published a number of papers in 1980s on cryptanalysis against various block ciphers and hash functions, including weaknesses in DES, e.g., in [BS90]. Surprisingly it has appeared that DES is quite resistant against differential attack, although a small change in it makes it weaker. It means that designers of DES did know about this kind of attack in 1970s. The NSA introduces some small changes in the original DES, to prevent the cipher from attacks. This was one of the reason to keep the design evaluation secret. Within IBM, this kind of attack was known as "*Tickling attack*". However, the differential attack on DES needs to have 2^{47} chosen plaintexts.

A *differential attack* works in the scenario when plaintext can be chosen. The idea behind is that pairs of plaintext are chosen in such a way, that the corresponding pairs of ciphertext (or, perhaps, keystreams) have some noticeable properties. A pair of a chosen plaintext m_1 and m_2 usually makes up a *difference* Δ, which can be defined in many ways, but usually just a XOR: $\Delta = m_1 \oplus m_2$. The attacker then computes the difference in the ciphertext, trying to determine some pattern in its statistics and reveal anomalities in the distribution of ciphertexts' pairs.

The way how to find the distribution of the differences in ciphertexts is not known exactly. The general way is to trace the initial plaintexts difference through the stages of the cipher, achieving a *difference characteristic* on each stage.

Resynchronisation attack is an extended differential cryptanalysis. You are given a set of keystream produced with multiple unknown keys and known IVs. In this scenario a distinguisher for the cipher can be found.

Other specialized types of the differential attack that can be mentioned are as follows: truncated differential cryptanalysis [KB96]; impossible differential cryptanalysis [BF00]; boomerang attack [Wag99]; higher order differential analysis [Knu94]; differential power analysis [KJJ99]; and others...

3.6.2 Algebraic Attacks

The idea of an *algebraic attack* takes its roots from the paper by Kipnis and Shamir [KS99]. It says that the keystream generated from some PRNG can be described by a system of Boolean equations of some degree, which hold

with probability 1 (or, in some cases, slightly less than 1, when applicable). The first stage, therefore, simply requires to find equations where only the secret key bits are involved. Afterwards, solve the equations. However, these equations are usually *nonlinear* and of a *high degree*, but *overdefined*. For example, it can contain 8000 equations on 1600 variables. Fast methods from analysis are generally not applicable. There is sometimes also a problem to find the equations, and sometimes non-trivial for stream ciphers.

The first paper that proposes the basis for solving such overdefined complex systems of equations is the paper from 2000 by Nicolas Courtois et al. [CKPS00]. They propose an algorithm called *XL*. The idea behind is simple (although, many people do not believe that it works). We are good to solve linear systems, but unable to solve system with quadratic terms. We can then us the *linearisation technique*. Each term $x_i x_j$ is then replaced with a new variable $y_{i,j}$, and treat them as independent from other variables. Then we receive only linear equations. If the number of equations N_{eq} is related to the number of variables N_{var} as $N_{\text{eq}} \leq N_{\text{var}}^2$, then we can solve such a system, perhaps, after a few guesses.

Assume we want to solve a system of equations on n variables $x_1, x_2, \ldots, x_n$, and let us have a set of equations $\{Q_j\}$. Pick a value $d \geq 2$ and let $x^k = \{\prod_{j=1}^{k} x_{i_j}\}_{i_j}$ is the set of all monomials of degree less than k. Then we can generate new equations $\left(\prod_{j=1}^{k} x_{i_j}\right) Q_i = 0$ for all $\prod_{j=1}^{k} x_{i_j} \in x^k$, for all $k \leq d-2$. We then treat each monomial in x_i's as an independent new variable and solve the system by Gaussian elimination as usual. The higher the value d the more equations we can derive, but the number of variables increases rapidly. If d is small then it will not give us enough equations either. It is a trade-off between the parameters.

For other recent results on algebraic attacks we would refer to, e.g., [CM03, DKDM04, MPC04, DGM06, CKPS00, CP02, MHLL02, CP03, Cou03, Moh01].

3.6.3 Side-Channel Attacks

Side-channel attacks (SCA) usually refer to attacks based on some weakness in a particular hardware (or, even in software in some cases) implementation of a cipher algorithm. Despite the analysis of the keystream, in SCA a particular side-channel information is used, for example, *timing* or *power usage* is analysed in specific points of a chip, or a smart card device.

To mount a SCA an *electromagnetic emission* technique can be used. It has been shown that these attacks are efficient for block ciphers and various public-key cryptography. In 1999, Kocher, Jaffe and Jun presented their result on power analysis of the cipher DES on a specific implementation. They have shown how 6 bits of the secret Key can be guessed and verified independently.

In a timing attack one can measure the actual time for different blocks of the algorithm, which can also give the knowledge about what happens inside. For example, timing analysis can easily distinguish whether the operation was XOR or a multiplication. Even during a multiplication, one can say how many summations were done at the end.

There exist several subcategories of the side-channel attacks, such as: timing analysis [DKL+98, Koc96]; power analysis attack [KJJ99]; acoustic cryptanalysis [ST06]; and others...

An implementation of a cipher can avoid these attack by, for example, buffering the output sequence. For more on side-channel attacks we would refer to, e.g., [OI05, LALM04].

In above sections we gave a brief overview of cryptanalysis techniques. However, the list of existing attacks is not limited; other attacks can also be considered: slide attacks [BW99,BW00,Pha05]; integral cryptanalysis [KW02].

4

Tools for Cryptanalysis

"Everything should be made as simple as possible, but not simpler"

Albert Einstein

As we saw in the previous section, linear cryptanalysis is one of the most powerful cryptanalysis techniques. It is, for example, the fastest known attack on DES. More recently, we have seen that linear cryptanalysis also plays a major role in the area of stream ciphers. Many recent proposals have been analysed through the idea of replacing nonlinear operations by linear ones, hoping that obtained linear equations are correct with a probability slightly larger than otherwise expected. Actually, the best known attacks on many recent stream cipher proposals are linear attacks. This includes stream ciphers like Scream [HCJ02], SNOW [EJ00, EJ02b], SOBER [HR00a, HR00b], RC4 [Sma03], A5/1 [BGW99], and many others.

A large part of research in linear cryptanalysis on block ciphers is based on bit-wise linear approximations. In short, the process consists of the following steps. We find a sum of certain plaintext bits, ciphertext bits and key bits such that this sum is zero with a probability $1/2 + \epsilon$, where ϵ is usually small. By getting access to a large number of different plaintext/ciphertext

pairs we can eventually find out the value of the sum of key bits. This results in a key recovery attack.

In linear attacks on stream ciphers, it is mostly the case that a linear approximation will give us a set of keystream symbols that sum to zero with probability $1/2 + \epsilon$. Since no key bits are involved in the expression, this gives us a distinguishing attack. In some linear attacks on stream ciphers, one has moved from the binary alphabet to instead consider a sum of variables defined over a larger set. For example, we can consider a sum of different bytes from keystream sequence if it is byte-oriented. Distinguishers based on symbols from a larger alphabet have been dealt with in, for example, [JM03, EJ02a, GH05].

It is clear that moving to a larger alphabet gives improved results. However, the computational complexity of finding the result increases. To be a bit more specific, assume for example that the operation $X_1 \boxplus X_2$ is replaced by $X_1 \oplus X_2$, where $\boxplus$ denotes mod 2^n addition. The usefulness of such an approximation is given by the distribution $\Pr\{(X_1 \boxplus X_2) \oplus (X_1 \oplus X_2) = \gamma\}$. However, the complexity of computing this distribution can be large. For example, for $n = 32$ bits a straight forward approach would require complexity 2^{64}, which is an impossible size to implement.

In several previous papers related problems were studied. For example, in [LM02] differential properties of addition, such as $\mathtt{DC}^+(\alpha, \beta \rightarrow \gamma) := \Pr\{(x \boxplus y) \oplus ((x \oplus \alpha) \boxplus (y \oplus \beta)) = \gamma\}$, were studied in details, including different useful and efficient computational algorithms. There are a few other results where different classes of similar functions (mostly related to differential properties) were achieved, e.g., in [LWD04, Max04, Lip02], and others. However, these papers the focus is only on a small class of functions, which can be regarded as a subclass of the functions studied in this chapter, referred to as *pseudo-linear functions*. Moreover, our main concern is the *algorithms using large distribution tables*, i.e., providing a *practical tool for cryptanalysis over large distributions* (or a large alphabet). When, for example, the probability space is $|\Omega| = 2^{32}$, our algorithms and data structures allow us to store and perform the most common operations over such huge distributions, in a reasonable time on a usual PC.

Consider $X_1, X_2, \ldots, X_k$ to be independent n bit random variables. If they have *arbitrary* distributions, we show how to compute distributions like $\Pr\{X_1 \oplus X_2 \oplus \cdots \oplus X_k\}$ and $\Pr\{X_1 \boxplus X_2 \boxplus \cdots \boxplus X_k\}$ in complexity $O(kn2^n)$. For example, we compute the distribution $\Pr\{(X_1 \boxplus X_2) \oplus (X_1 \oplus X_2) = \gamma\}$ in complexity $2^{37} \cdot c$ for some small c. The presented algorithms make use of techniques from *Fast Fourier Transform* and *Fast Hadamard Transform*. Although some of these techniques were also mentioned in a recent paper [GM04], we include the full approach for completeness. We show how they can be performed when more complicated data structures are used,

introduced due to a high memory complexity.

Moreover, in cases when $X_1, X_2, \ldots, X_k$ are *uniformly* distributed we demonstrate a large class of functions $F(X_1, X_2, \ldots, X_k)$, for which the distribution $\Pr\{F(X_1, X_2, \ldots, X_k) = \gamma\}$ can be efficiently computed. Here, the algorithms are based on performing a combinatorial count in a bit-wise fashion, taking the "carry depth" into account. These results give us efficient methods of calculating distributions of *certain* functions $F(X_1, X_2, \ldots, X_k)$. Fortunately, this includes many functions that appear in linear analysis of ciphers.

4.1 Pseudo-Linear Functions Modulo 2^n

In this chapter we denote n bit variables by a capital letter X, and 1 bit variables by a small letter x. Individual bits of X in a vector form are represented as $X = \overline{x_{n-1} \ldots x_1 x_0}$. By $X[a:b]$ we denote an integer number of the form $\overline{x_b \ldots x_{a+1} x_a}$. If $Y = \overline{y_{m-1} \ldots y_0}$, then $X||Y = \overline{x_{n-1} \ldots x_0 y_{m-1} \ldots y_0}$ is another integer number (*concatenation*). We use '$\boxplus$' and '$\boxminus$' to denote arithmetical addition and subtraction modulo 2^n, respectively. However, when the inputs to a function $F(\cdot)$ are from the ring $\mathbb{Z}_{2^n}$, we assume '$+$' to be an addition in the ring as well. Matrix multiplication is denoted as '$\times$'. When '$\cdot$' is applied to two vectors, it denotes element-by-element multiplication of corresponding positions from the vectors.

4.1.1 A Pseudo-Linear Function Modulo 2^n

Let $\mathcal{X}$ be a set of k uniformly distributed n bit (nonnegative) integer random variables $\mathcal{X} = \{X_1, \ldots, X_k\}$, $X_i \in \mathbb{Z}_{2^n}$. Let $\mathcal{C}$ be a set of n bit constants $\mathcal{C} = \{C_1, \ldots, C_l\}$. Let T_i be some symbol or expression on $\mathcal{X}$ and $\mathcal{C}$. We define *arithmetic*, *Boolean*, and *simple terms* as follows.

Definition 4.1 (Algebraic, Boolean, and Simple Terms): Given $\mathcal{X}$ and $\mathcal{C}$ we say that:

(1) $\mathcal{A}$ is an *arithmetic term*, if it has only the arithmetic $+$ operator between the input terms (e.g., $\mathcal{A} = T_1 + T_2 + \ldots$).

(2) $\mathcal{B}$ is a *Boolean term* if it contains only bit-wise operators such as NOT, OR, AND, XOR, and others (e.g., $\mathcal{B} = (\overline{T_1} \oplus T_2)|T_3 \& \overline{T_4} \ldots$).

(3) $\mathcal{S}$ is a *simple term* if it is a symbol from either $\mathcal{X}$ or $\mathcal{C}$ (e.g., $\mathcal{S} = X_i$).

□

Next, we define a *pseudo-linear function modulo* 2^n.

Definition 4.2 (Pseudo-Linear Function): $F(X_1, \ldots, X_k)$ is called a *pseudo-linear function modulo* 2^n *(PLFM)* on $\mathcal{X}$ if it can recursively be expressed in

arithmetic ($\mathcal{A}$), Boolean ($\mathcal{B}$), and simple ($\mathcal{S}$) terms [1]. We also assume the number of $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{S}$ terms to be a, b, and s, respectively. □

Note, if a given function contains a subtraction $\boxminus$, then it can easily be substituted by $\boxplus$ using

$$X \boxminus Y \equiv X \boxplus (\texttt{NOT}\ Y) \boxplus 1 \mod 2^n, \tag{4.1}$$

which is valid in the ring of integers modulo 2^n. Note that the number of $\mathcal{A}$-terms does not grow during the substitution.

As an example, let us consider a linear approximation of a modulo sum of the kind '$X_1 \boxplus X_2 \boxplus X_3 \rightarrow X_1 \oplus X_2 \oplus X_3 \oplus N$', where N is the noise variable introduced due to the approximation. The expression for the noise variable is a PLFM:

$$N = F(X_1, X_2, X_3) = (X_1 + X_2 + X_3) \oplus X_1 \oplus X_2 \oplus X_3. \tag{4.2}$$

Finding the distribution of such an approximation could be the bottleneck in cryptanalysis. The trivial algorithm for solving this problem would be as follows.

```
1.    Loop for all (X_1, X_2, X_3) ∈ Z_{2^n}^3
2.        T[(X_1 ⊞ X_2 ⊞ X_3) ⊕ X_1 ⊕ X_2 ⊕ X_3] + +.
```

After termination of the algorithm we have $\Pr\{N = \gamma\} = T[\gamma]/2^{3n}$. The complexity of this classical solution when the variables are 32 bit integers, is $O(2^{96})$, infeasible for a common PC. Instead, we suggest another principle to solve this problem, as follows.

```
1.    for γ = 0...2^n − 1
2.        T[γ] = some combinatorial function.
```

In the following section we will show how this combinatorial function is constructed.

4.1.2 Algorithm for Calculating the Distribution for a PLFM

The problem we are considering in this subsection is the following. Given a PLFM $F(X_1, X_2, \ldots, X_k)$ on $\mathcal{X}$ and $\mathcal{C}$, we want to calculate the probability $\Pr\{F(X_1, X_2, \ldots, X_k) = \gamma\}$, for a fixed value γ, in an efficient way.

[1] Note that a PLFM is a T-function [KS03], but not vice versa.

Let some arithmetic term $\mathcal{A}$ have k^+ operators '+', i.e., $\mathcal{A} = T_0 + T_1 + \ldots + T_k$, where T_j are some other terms, possibly $\mathcal{B}$ or $\mathcal{S}$. Then, considering 1 bit inputs, the evaluation of the $\mathcal{A}$ term can, potentially, produce the *local* maximum carry value $\omega_{\max}$ as

$$\omega_{\max} = \lfloor \frac{k^+ + 1}{2} \rfloor. \tag{4.3}$$

This carry value at some bit t can influence the next bits of the sum at positions $t+1, t+2$, etc. Therefore, the maximum carry value $\sigma_{\max}$ at every bit t of the sum for $\mathcal{A}$ is then derived as the minimum integer solution for the equation

$$\sigma_{\max} = \lfloor (k^+ + 1 + \sigma_{\max})/2 \rfloor. \tag{4.4}$$

Thus, for every arithmetic term $\mathcal{A}_i$ the *maximum local carry value*, denoted by $\sigma_{i\max}$, is

$$\sigma_{i\max} = k_i^+, \tag{4.5}$$

where k_i^+ is the number of additions in $\mathcal{A}_i$.

For any t bit truncated input tuple $(X_1, \ldots, X_k)$ to the function $F(\cdot)$ we can define *a tuple of local carry values* for each of the $\mathcal{A}_i$-terms, as follows:

$$\Psi|_t = (\sigma_1, \sigma_2, \ldots, \sigma_a)|_t, \tag{4.6}$$

where σ_i is the corresponding local carry value for the $\mathcal{A}_i$-term, when the inputs are t bit truncated, and it can also be expressed as

$$\sigma_i|_t = \left(\sum_{j=0}^{k_i^+} (T_{i,j}(X_1, \ldots, X_k) \mod 2^t) \right) \operatorname{div} 2^t, \tag{4.7}$$

when $\mathcal{A}_i = T_{i,0} + \ldots + T_{i,k_i^+}$.

Assume there is an oracle $R_t(\Psi_0, \gamma)$ that can tell us the number of choices of the tuple $(X_1[0:t-1], \ldots, X_k[0:t-1])$ out of $2^{t \cdot k}$ possible combinations, such that for each choice the function F produces a required vector of local carry values $\Psi|_t = \Psi_0$, and the condition $F(X_1, \ldots, X_k) = \gamma \mod 2^t$ is satisfied, i.e. $F(X_1, \ldots, X_k)[0:t-1] = \gamma[0:t-1]$. The probability we are seeking can now be written as

$$\Pr\{F(X_1, \ldots, X_k) = \gamma\} = \frac{1}{2^{k \cdot n}} \sum_{\Psi} R_n(\Psi, \gamma). \tag{4.8}$$

It remains to show how to construct the oracles $R_t(\Psi_0, \gamma)$. Let us assume that $R_t(\Psi_0, \gamma)$ is known for every Ψ_0. When $\Psi|_t = \Psi_0$ is fixed, then, by trying all combinations for t^{th} bits of the inputs, i.e., testing each k bit vector

$(X_1[t:t],\ldots,X_k[t:t])$, we can calculate the exact value of $F(X_1,\ldots,X_k)[t:t]$, as well as the exact resulting local carries vector $\Psi|_{t+1}$. Clearly, the oracle $R_{t+1}(\Psi',\gamma)$ makes calls to $R_t(\Psi_0,\gamma)$, for various values of Ψ_0. That relation is linear, and can easily be represented in a matrix form. For this purpose, let us introduce a one-to-one *index mapping function*

$$\texttt{Index}(\Psi):(\sigma_1\times\sigma_2\times\ldots\times\sigma_a)\rightarrow\theta\in[0\ldots\theta_{\max}-1], \tag{4.9}$$

as follows.

$$\begin{aligned}&\texttt{Index}(\Psi)=((\sigma_1\cdot(\sigma_{2\max}+1)+\sigma_2)\cdot(\sigma_{3\max}+1)+\sigma_3)\cdot\ldots,\\&\theta_{\max}=\prod_{j=1}^{a}(\sigma_{j\max}+1)=\prod_{j=1}^{a}(k_j^{+}+1).\end{aligned} \tag{4.10}$$

Now, $R_t(\Psi,\gamma)$ for all Ψ can be regarded as a vector

$$\Big(R_t(\texttt{Index}^{-1}(0),\gamma),\ldots,R_t(\texttt{Index}^{-1}(\theta_{\max}-1),\gamma)\Big), \tag{4.11}$$

also referred for simplicity as R_t, for all the consecutive valid tuples Ψ. The transformation from R_t to R_{t+1} is a linear function, i.e., it can be written as

$$R_{t+1}=M_{\gamma_t|t}\times R_t, \tag{4.12}$$

where $M_{\gamma_t|t}$ is some fixed *connection matrix* of size $(\theta_{\max}\times\theta_{\max})$, which, in general, is different for different ts. It depends on the t^{th} bits of the constants involved in $F(\cdot)$, as well as on the value of the t^{th} bit γ_t from the given γ, since the oracle $R_{t+1}(\Psi,\gamma)$ must also satisfy γ taken modulo 2^{t+1}. If the input variables are 0-truncated, then the only one vector $\Psi|_0=(0,0,\ldots,0)$ of local carry values is possible, i.e., $R_0=(1\ 0\ \ldots\ 0)$. Therefore, we assign the oracle R_0 to be just a zero vector, but $R_0(0,\gamma)=1$.

In this way, $2n$ such matrices have to be constructed. However, in most cases this number is much smaller. The algorithm to construct matrices from (4.12) and then calculate (4.8) is given as follows.

Theorem 4.1: For a given PLFM $F(X_1,\ldots,X_k)$, and a fixed $\gamma\in\mathbb{Z}_{2^n}$, we have:

$$\Pr\{F(X_1,\ldots,X_k)=\gamma\}=\frac{1}{2^{k\cdot n}}(1\ 1\ \ldots\ 1)\times\left(\prod_{t=n-1}^{0}M_{\gamma_t|t}\right)\times(1\ 0\ \ldots\ 0)^{\mathrm{T}}, \tag{4.13}$$

where $M_{\gamma_t|t}$ are connection matrices of size $(\theta_{\max}\times\theta_{\max})$, precomputed with the algorithm below.

Algorithm: Construction of $2n$ matrices $M_{\gamma_t|t}$.

1. *Input:*

   $F(X_1, \ldots, X_k)$ – a PLFM with a arithmetical terms $\mathcal{A}_i$, each having k_i^+ operators '+', correspondingly.

2. *Data structures:*

   $\theta_{\max} = \prod_{i=1}^{a}(k_i^+ + 1)$.

   $M_{\{0,1\}|t=[0\ldots n-1]}[\theta_{\max}][\theta_{\max}]$ – $2n$ square matrices of size $(\theta_{\max} \times \theta_{\max})$, initialised with zeros.

3. *Precomputation algorithm:*

   `for` $t = 0 \ldots n-1$

   Temporarily set the constants from $\mathcal{C}$ to be just t^{th} bit of the original ones, i.e., set $(C_1, \ldots, C_l) = (C_1[t:t], \ldots, C_l[t:t])$

   `for` $(X_1, \ldots, X_k) \in \{0,1\}^k$ – *(all combinations for the t^{th} bits of Xs)*

   `for` $\theta = 0 \ldots \theta_{\max} - 1$ – *(all combinations for Ψ)*

   $(\sigma_1, \ldots, \sigma_a) = \texttt{Index}^{-1}(\theta)$

   [z] Evaluate all $\mu_i = \sigma_i + \mathcal{A}_i(X_1, \ldots, X_n)$, but in $\mathcal{A}_i$ substitute all sub-terms $\mathcal{A}_j$ with the values $(\mu_j \mod 2)$, correspondingly

   $\theta' = \texttt{Index}(\mu_1 \text{ div } 2, \ldots, \mu_a \text{ div } 2)$ – *(a new resulting Ψ')*

   Evaluate the function $f = F(\cdot) \mod 2$, but substitute all terms $\mathcal{A}_j$ with the values μ_j, correspondingly

   $M_{f|t}[\theta'][\theta] := M_{f|t}[\theta'][\theta] + 1$

- Time Complexity: $O(n \cdot \theta_{\max} \cdot 2^k)$
- Memory Complexity: $O(2n \cdot \theta_{\max}^2)$

[z] Variables μ_i, which correspond to the terms $\mathcal{A}_i$, should be calculated recursively. The deepest $\mathcal{A}$ term should be calculated first, and so on.

□

Below we give an example that demonstrates all the steps of the algorithm.

EXAMPLE *4.1 (Pseudo Linear Function):* Let $k = 3$, $n = 5$. Assume that our goal is to calculate the probability $\Pr\{F(X_1, X_2, X_3) = 10110_2\}$, where:

$$F(X_1, X_2, X_3) = (X_1 \boxplus (X_2 \oplus (X_1 \boxminus X_2 \boxplus 25)))) \oplus (X_1 \texttt{ AND } X_3). \quad (4.14)$$

The first step is to cancel the operator $\boxminus$ by (4.1), and by rewriting the

expression we get:

$$F(X_1,X_2,X_3) = \overbrace{(X_1 + \overbrace{(X_2 \oplus (X_1 + \underbrace{(\texttt{NOT}\ X_2)}_{\mathcal{B}_1}) + 26)}^{\mathcal{B}_2})}^{\mathcal{A}_2} \oplus \underbrace{(X_1\ \texttt{AND}\ X_3)}_{\mathcal{A}_1}\,. \qquad \overbrace{\qquad\qquad\qquad\qquad\qquad\qquad}^{\mathcal{B}_3} \tag{4.15}$$

The function $F(\cdot)$ is a PLFM, since it can be expressed in $\mathcal{A}$ and $\mathcal{B}$ terms marked above. The $\mathcal{S}$ terms are simply elements from the set $\{X_1, X_2, X_3, 26\}$, i.e.,

$$\begin{aligned}
\mathcal{B}_1(\mathcal{X},\mathcal{C}) &= \texttt{NOT}\ X_2 \\
\mathcal{A}_1(\mathcal{X},\mathcal{C}) &= \underbrace{X_1 + \mathcal{B}_1(\mathcal{X},\mathcal{C}) + 26}_{k_1^+=2} \\
\mathcal{B}_2(\mathcal{X},\mathcal{C}) &= X_2 \oplus \mathcal{A}_1(\mathcal{X},\mathcal{C}) \\
\mathcal{A}_2(\mathcal{X},\mathcal{C}) &= \underbrace{X_1 + \mathcal{B}_2(\mathcal{X},\mathcal{C})}_{k_2^+=1} \\
\mathcal{B}_3(\mathcal{X},\mathcal{C}) &= \mathcal{A}_2(\mathcal{X},\mathcal{C}) \oplus (X_1\ \texttt{AND}\ X_3) \\
F(X_1,X_2,X_3) &= \mathcal{B}_3(\mathcal{X},\mathcal{C})
\end{aligned} \tag{4.16}$$

The algorithm to compute the $2n$ matrices $M_{f|t}$ is as follows.

```
1.  θ_max = (k_1^+ + 1)(k_2^+ + 1) = 3 · 2 = 6;
2.  for t = 0...4
3.      C = 26[t : t]
4.      for (X_1, X_2, X_3) ∈ {0,1}^3
5.          for (σ_1, σ_2) = (0...2, 0...1)
6.              μ_1 = σ_1 + X_1 + ( NOT X_2) + C
7.              μ_2 = σ_2 + X_1 + (X_2 ⊕ μ_1  mod 2)
8.              f = (μ_2 ⊕ (X_1 AND X_3))  mod 2
9.              M_{f|t}[(μ_1  div  2) · 2 + (μ_2  div  2)][σ_1 · 2 + σ_2] + +
```

Applying Theorem 4.1 to construct $2n$ matrices.

After all computations we receive the following matrices

$$M_{\gamma_0=0|t=0} = \begin{pmatrix} 1 & 0 & 2 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 1 & 0 & 2 & 0 & 1 & 0 \\ 0 & 1 & 2 & 2 & 0 & 5 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \qquad M_{\gamma_0=1|t=0} = \begin{pmatrix} 5 & 0 & 0 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 & 5 & 0 \\ 0 & 1 & 2 & 2 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M_{\gamma_1=0|t=1} = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 2 & 0 \\ 2 & 2 & 0 & 5 & 0 & 0 \\ 0 & 0 & 1 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 & 2 & 2 \end{pmatrix} \qquad M_{\gamma_1=1|t=1} = \begin{pmatrix} 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 5 & 0 & 0 & 2 \\ 2 & 2 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 & 2 & 2 \end{pmatrix}.$$

No need to construct the matrices for $t = 2, 3, 4$, because they will repeat as $M_{*|t=2} = M_{*|t=0}$ and $M_{*|t=4} = M_{*|t=3} = M_{*|t=1}$. This happens since there are only two different combinations for any t^{th} "bit slice" of constants from the set $\mathcal{C} = \{26\}$. In particular, for every bit t we have $26[t : t] = 0$ or 1 in step 3 in the figure above. Finally, from (4.13) we calculate

$$\Pr\{F(X_1, X_2, X_3) = 10110_2\} = \frac{1}{2^{15}}(1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1) \times M_{1|4} \times M_{0|3} \times M_{1|2} \times$$
$$\times M_{1|1} \times M_{0|0} \times (1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0)^{\mathrm{T}} = \frac{1}{2^{15}} \cdot 404 \approx 0.0123291015625. \tag{4.17}$$

One can check this probability using the classical approach of trying all possible values for $(X_1, X_2, X_3) \in \mathbb{Z}_{2^5}^3$ and calculating the function $F(\cdot)$ directly from (4.14).

Preparing the matrices requires $2 \cdot 2^3 \cdot 6 = 96$ steps (2 values for t, 8 combinations for (X_1, X_2, X_3), and the number of different local carries is $\theta_{\max} = 6$); each step requires one function evaluation. To calculate one probability we need to make 5 multiplications of a matrix and a vector, which takes $5 \cdot 6^2$ operations, plus one scalar product of two vectors at the end, i.e., in total 186 operations. Calculating the complete distribution for all possible γs takes $2^5 \cdot 186 = 5952$ operations in total. Note that the classical approach requires $2^{3 \cdot 5} = 32768$ steps, including the function evaluation in each step.

□

The next example is taken from real cryptanalysis. In this example we, additionally, demonstrate a new trick and show how time complexity can be reduced even more than in Theorem 4.1. With a precomputation, which

usually takes a negligible time, the construction of the complete distribution can have a very small time complexity $O(\theta_{\max} \cdot 2^n)$. This also shows the advantage of using the proposed technique since the computation complexity of 2^{96} steps from the classical approach is reduced down to $2^{32.585}$ steps.

EXAMPLE *4.2 (Technique for Computation Complexity Reduction):* Let us have $k = 3$ uniformly distributed independent random variables $X_1, X_2, X_3 \in \mathbb{Z}_{2^{32}}$, i.e., $n = 32$. Let us assume that we want to perform a linear approximation '$X_1 \boxplus X_2 \boxplus X_3 \rightarrow X_1 \oplus X_2 \oplus X_3 \oplus N$', where N is a noise variable introduced due to the approximation. The task is to find the bias ϵ of the noise variable N. The expression for N is:

$$N = \underbrace{(\underbrace{X_1 + X_2 + X_3}_{\mathcal{A}_1}) \oplus X_1 \oplus X_2 \oplus X_3}_{\mathcal{B}_1} \mod 2^{32}, \tag{4.18}$$

which is a PLFM with only one $\mathcal{A}$ term. The maximum carry-bit index value is $\theta_{\max} = (k_1^+ + 1) = 3$. Since no constants are involved, all matrices $M_{*|t}$ for all ts are the same. Hence, only two matrices $M_{0|0}$ and $M_{1|0}$ have to be constructed, using Theorem 4.1.

$$M_{\gamma_0=0|t=0} = \begin{pmatrix} 4 & 0 & 0 \\ 4 & 0 & 4 \\ 0 & 0 & 4 \end{pmatrix}, \quad M_{\gamma_0=1|t=0} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 6 & 0 \\ 0 & 1 & 0 \end{pmatrix}. \tag{4.19}$$

The probability $\Pr\{N = \gamma\}$ can now be calculated efficiently. For example,

$$\begin{aligned} \Pr\{N = \gamma = \text{0x72A304F8}\} &= \frac{1}{2^{3 \cdot 32}} (1 \;\; 1 \;\; 1) \times \left(\prod_{t=n-1}^{0} M_{\gamma[t:t]|0} \right) \times (1 \;\; 0 \;\; 0)^{\mathrm{T}} \\ &= \frac{1}{2^{96}} \cdot 2187 \cdot 2^{51} \approx 0.266967773/2^{32}. \end{aligned} \tag{4.20}$$

Note that the probability for an odd γ is 0. To calculate one probability, $32 \cdot 3^2 + 3 = 291$ operations are required. Hence, to calculating the complete distribution, it would require $291 \cdot 2^{32}$ operations.

However, this time complexity can be reduced significantly by using specific data structures, which we call "*fast-tables*". Each table contains 2^{16} entries, each having 3-dimensional vectors. These tables are precomputed as shown in Figure below.

1. *Data structures:*

   FastT[2][$0 \dots 2^{16}-1$] – two 'fast-tables'

2. *Initialisation:*

   FastT[0][0] = (1 0 0), FastT[1][0] = (1 1 1)

3. *Precomputation of the tables:*
   for $t = 0 \dots 15$
   for $x = 1, 0$ (*note, the order is backward*)
   for $Y = 0 \dots 2^t - 1$
   [z] FastT$[0][x||Y_t] = M_{x|t} \times$FastT[0][$Y$]
   FastT$[1][x||Y_t] =$ FastT[1][Y]$\times M_{x|n-t-1}$

   Fast-tables precomputation algorithm.

[z] Y_t is a t bit value of Y. In, for example, C/C++, it would look like: $(x||Y_t) \Rightarrow$ (x<<t) | Y

This precomputation requires $2^{16} \cdot 2 \cdot 3^2 = 9 \cdot 2^{17}$ operations. The advantage is that any probability can now be derived as just one scalar product [2]

$$\Pr\{N = \gamma\} = \frac{1}{2^{3 \cdot 32}} \cdot <\text{FastT}[0][\overline{\gamma_{15} \dots \gamma_0}], \text{FastT}[1][\overline{\gamma_{16} \dots \gamma_{31}}] >, \qquad (4.21)$$

which takes only 3 operations (instead of 291). Finally, the bias ϵ can be derived as follows:

1. $\epsilon = 0.5$ *(the bias for odd values of γ)*
2. for $\gamma = 0 \dots 2^{31} - 1$ *(only even 2γ:s are considered)*
3. $\epsilon += |\Pr\{N = 2\gamma\} - 2^{-32}|$

The total time needed in this approach is the following sum: $2 \cdot 2^3 \cdot 3 = 48$ to compute matrices, $9 \cdot 2^{17}$ to precompute fast-tables, and $3 \cdot 2^{31}$ to calculate the bias ϵ. In total, $6443630640 \approx 2^{32.585}$ operations are required. To calculate the distribution of the noise variable N, the same number of operations is needed, whereas the classical approach requires 2^{96} operations. Thus, when the question is only to find the bias ϵ for some large distribution with memory limits conditions, the classical approach will fail because of the memory limits. □

[2] Note, the input for FastT[1][·] is bit-reversed.

4.2 Distributions of Functions With Arbitrarily Distributed Inputs

The previous section assumed $X_1, X_2, \ldots$ to be uniformly distributed, allowing a combinatorial approach. In this section we consider $X_1, X_2, \ldots$ *independent* but with *arbitrary* distributions. Despite the fact that the ideas described in this section were partly mentioned in [GM04], we include them for completeness.

Let us have a probability space Ω of size $q = |\Omega| = 2^n$ and two distributions P_X and P_Y over Ω for two random variables X and Y, respectively. Given the distributions P_X and P_Y we consider two major types of convolution, defined as

$$\begin{aligned} &P_Z = P_X * P_Y :\Rightarrow \\ &\Pr\{Z = Z_0\} = \sum_{\substack{\forall X_0, Y_0 \in \Omega : \\ X_0 * Y_0 = Z_0}} \Pr\{X = X_0\} \cdot \Pr\{Y = Y_0\}, \quad \forall Z_0 \in \mathbb{Z}_{2^n}, \end{aligned} \tag{4.22}$$

where $*$ is either $\boxplus$ or $\oplus$.

In both cases the time complexity to calculate the resulting distribution P_Z is $O(q^2)$, i.e., quadratic. Due to such a high complexity, many attacks in cryptanalysis deal with at most 16-18 bit distributions only. Nowadays, when design of ciphers is often 32 bit oriented, it would be a useful task to perform a convolution of two 32 bit distributions, i.e., calculating $\Pr\{X + Y = \gamma\}$ for all γ when X and Y have arbitrary distributions.

For notation purposes the distribution P_X will also be represented as a vector of size 2^n of probabilities as

$$[P_X] = \{p_X(0), p_X(1), \ldots, p_X(2^n - 1)\}, \tag{4.23}$$

where $p_X(X_0) = \Pr\{X = X_0\}$.

4.2.1 Convolution over $\boxplus$

If $[P_X]$ and $[P_Y]$ are represented as two polynomials with coefficients from these two vectors, then the resulting vector $[P_Z]$ has coefficients of the product of the polynomials $[P_X]$ and $[P_Y]$. Fast multiplication of two polynomials can be done via *Fast Fourier Transform (FFT)* [CLRS01], the complexity of which is $O(q \log q)$ [3]. The convolution over $\boxplus$ can now easily be calculated as

$$[P_Z] = [P_X \boxplus P_Y] = \text{FFT}_n^{-1}(\text{FFT}_n([P_X]) \cdot \text{FFT}_n([P_Y])). \tag{4.24}$$

[3] The resulting polynomial $[P_X] \cdot [P_Y]$ is of degree $2q$, but its powers have to be taken modulo q. It means that the second half just needs to be added to the first half of $2n$ coefficients, in order to receive $[P_Z]$. However, this is done automatically when FFT of size q is applied to $[P_X]$ and $[P_Y]$ directly.

4.2.2 Convolution over $\oplus$

A similar idea can be applied to this type of convolution. Instead, we use *Fast Hadamard Transform (FHT)* [CLRS01].

FHT is a linear transformation of a vector of size 2^n. This transformation can also be done by a matrix multiplication $H_n \times [V]$, where H_n is a well-known Hadamard matrix. FHT, however, performs this matrix multiplication for time $O(q \log q = n \cdot 2^n)$, the same as FFT. In practice, FHT is much faster than FFT, since it does not need to work with complex and float numbers. Therefore, approximations of kind $\boxplus \Rightarrow \oplus$ are more preferable, than otherwise. Additionally, the implementation of FHT is extremely simple and short in C/C++, and can be done as follows.

Fast Hadamard Transform (FHT) implementation in C/C++

```
// butterfly operation
template<class T> void inline bfly (T &a, T &b)
{ T tmp; tmp=a; a+=b; b=tmp-b; }

// FHT_n, size of the input distribution is 2^n
template<class T> void FHT(int n, T *Dist)
{ for (int i=0; i<n; ++i)
    for (int j=0; j<(1<<n); j+=1<<(i+1) )
      for (int k=0 ; k<(1<<i); ++k)
         bfly (Dist[j+k], Dist[j+k+(1<<i)]);
}
```

Since FHT_n^{-1} differs from FHT_n only in the coefficient 2^{-n}, then the convolution over $\oplus$ via FHT is computed as

$$[P_Z] = [P_X \oplus P_Y] = \frac{1}{2^n} \cdot \text{FHT}_n(\text{FHT}_n([P_X]) \cdot \text{FHT}_n([P_Y])). \tag{4.25}$$

Finally, we point out that the convolution of a linear composition of k independent terms is derived as

$$P_{(Z=C_1X_1 \oplus C_2X_2 \oplus \ldots \oplus C_kX_k)} = \frac{1}{2^n} \cdot \text{FHT}_n \left(\text{FHT}_n([P_{C_1X_1}]) \cdot \ldots \cdot \text{FHT}_n([P_{C_kX_k}]) \right), \tag{4.26}$$

where C_i are some constants. In practice, this also means that if these distribution tables for $X_1, \ldots, X_k$ are stored with precisions $\xi_1, \ldots, \xi_k$ bits after point, respectively, then for probabilities of Z the precision of only $\xi = n + \sum_{j=1}^{k} \xi_j$ bits after point should be considered (or reserved) before the FHT procedure.

4.3 Data Structures for Large Distributions and Operations

In the sections above several algorithms have been derived with good time complexities, which, in most cases, allow us to operate on large distributions. However, memory complexity problems can be a main concern in implementation. We have algorithms that operate with 32 bit distributions, but how do we manage the memory? In this section we present a possible solution, suggest our data structures for large distributions and show how typical operations can be performed.

4.3.1 Data Structure Proposal

Let us assume that we want to operate on a distribution of size 2^n, but the operation memory allows us to work only with a distribution of the maximum size 2^m, where $m < n$. In order to be able to work with large distributions of size 2^n, we propose to use *hard disk memory (HDD)*. If, for simplicity, we define

$$r = n - m. \tag{4.27}$$

Then one need to create 2^r files on HDD, which we denote as $\text{File}^r_{(0\ldots 2^r-1)}$, to store one distribution table. The upper parameter r denotes the number of files to be created (2^r), and the index on the bottom is the selector of a particular file. Sometimes we will also write

$$\text{File}^r_{X:(A)} \tag{4.28}$$

to show that this is the sub-distribution file A for the random variable X. Each file stores the corresponding sub-distribution of size 2^m. I.e., the probability $\Pr\{X = X_0\}$ can be accessed by

$$\Pr\{X = X_0\} = \text{File}^r_{X:(X_0[m:n-1])}[X_0 \mod 2^m]. \tag{4.29}$$

Note that the *upper* $r = (n - m)$ bits select the file, and the *lower* m bits are the cell index in the sub-distribution.

The operation memory is regarded as *a fast memory*, whereas the HDD memory is regarded as *a very slow memory*. Working with such data structure frequent access (loading and saving) to the files on HDD should be avoided, since these operations are much slower than access to the memory. I.e., the most operations have to be done in the operation memory domain, and the number of accesses to the files has to be reduced as much as possible. In the next parts of this chapter we present efficient solutions to apply common algorithms when operating on large distributions with the proposed data structures.

4.3.2 A PLFM Distribution Construction

For a given pseudo-linear function $F(\cdot)$ modulo 2^n, its distribution can be constructed as follows.

1. for $A = 0 \dots 2^r - 1$
2. load sub-distribution SubDist$[\cdot] \leftarrow \text{File}^r_{(A)}$
3. calculate the vector
   $v = (1 \quad 1 \quad \dots \quad 1) \times (\prod_{t=r-1}^{0} M_{A[t:t]\|t+m})$
4. for $B = 0 \dots 2^m$
5. SubDist$[B] = \Pr\{F = \overline{AB}\}$
   $= v \times (\prod_{t=m-1}^{0} M_{B[t:t]\|t}) \times (1 \quad 0 \dots \quad 0)^{\mathrm{T}}$
6. save sub-distribution $\text{File}^r_{(A)} \leftarrow$SubDist$[\cdot]$

This algorithm requires accessing each file only once. Additionally, the steps 3 and 5 could be done more efficiently with precomputed fast-tables (see Example 4.2).

4.3.3 A Function $Y = F(X)$ Evaluation Distribution

Let us have a distribution P_X of a random variable X, stored in the suggested data structures. Let us also have a function defined on one variable $F(X)$. We need to construct the distribution of $Y = F(X)$ in an efficient way. For example, this function could be a multiplication $\alpha \cdot X$ in a finite field, a permutation of X, a multiplication on a matrix, or some other function on X in general.

One could take the values of X consecutively, and then each time calculate Y. The problem appears when the consecutive values Y need to be stored in different files. It could happen that we need to access the Y's files $O(2^n)$ times, which is very time-consuming.

We suggest the following algorithm consisting of three stages. In the first stage the function is evaluated and the resulting Y's are separated into two files (bins), according to the upper bit value. In the second stage we perform *binary sorting* algorithm, each time dividing each bin into two new bins. In the third stage probabilities from the bins are accumulated and the resulting sub-distributions are transferred to the data structures of Y (files).

Stage I: Evaluate $Y = F(X)$ and separate into two files (narrowed distribution)

```
1.  create two files-bins f_0 = *File^1_{Y:(0)} and f_1 = *File^1_{Y:(1)}
2.  for all A = 0...2^{n-m} - 1
3.      load sub-distribution SubDist_X[·] ←File^r_{X:(A)}
4.      for all B = 0...2^m - 1
5.        Evaluate Y_0 = F(A||B)
6.        Save the pair f_{Y_0[n-1:n-1]} ← (SubDist_X[B], Y_0)
7.  close the files f_0 and f_1
```

Stage II: Expand the files $^*\text{File}^1_{Y:(A_1)} \to {}^*\text{File}^2_{Y:(A_2)} \to \ldots \to {}^*\text{File}^r_{Y:(A_r)}$

```
1.  for k = 1...r - 1
2.    for all A = 0...2^k - 1
3.      open two files f_0 = *File^{k+1}_{Y:(A||0)} and f_1 = *File^{k+1}_{Y:(A||1)}
4.      while( not the end of the file *File^k_{Y:(A)} )
5.         read the pair (p, Y_0) ← *File^k_{Y:(A)}
6.         save the pair f_{Y_0[n-k-1:n-k-1]} ← (p, Y_0)
7.      close the files f_0 and f_1
```

Stage III: Construct $\text{File}^r_{Y:(A)}$ *from* $^*\text{File}^r_{Y:(A)}$

```
1.  for all A = 0...2^r
2.      clear SubDist_Y[0...2^m - 1]
3.      while( not the end of the file *File^r_{Y:(A)} )
4.        read the pair (p, Y_0) ← *File^k_{Y:(A)}
5.        SubDist_Y[Y_0]=SubDist_Y[Y_0]+p
6.      save sub distribution File^r_{Y:(A)}←SubDist_Y[·]
```

The complexity of this algorithm is $O((1+r)\cdot 2^n)$. However, the coefficient r in the complexity can be reduced with a small programming trick. If at the step II.3 we, instead, open 2^d files (in Windows at most 2^9 files can be open at the same time), and perform a d-tuple bits (not a binary) sorting at once, then the complexity will be reduced to $O((1+r/d)\cdot 2^n)$. For example, if the number of files is 2^{16} (r=16), then with $d=8$ we can compute the distribution of any function $F(X)$ by reading and storing distributions of size 2^n from the files only 3 times (instead of 17).

Note that in the implementation of FFT the first operation is the construction of the distribution $P_{\text{Rev}(X)}$ for the *bit reverse* of the random variable X, which is just a subcase of the general problem of this subsection. We

simply define the function $Y = F(X)$ such that Y is the *bit-reverse* of X, and apply the algorithm above. There are other nice and more efficient solutions for this particular problem, but here we only mention them.

4.3.4 Convolution over $\oplus$

To perform a convolution over $\oplus$ we need to be able to perform FHT on the proposed data structures. We propose a modified FHT algorithm, where first local FHTs for sub distributions are separately performed, and then evaluate the "convolution" over the files as follows.

```
1.  for A = 0...2^r - 1
2.      load sub distribution SubDist[·] ←File^r_(A)
3.      FHT(m, SubDist)
4.      save sub distribution File^r_(A) ←SubDist[·]
5.  FHT*(r, NULL) -- the same FHT as before but
                     with another butterfly function
                     bfly*(j+k, j+k+(1<<i)).
```

The modified butterfly function `bfly*` is

```
1.  bfly*(A, B)
2.     load SubDist_1[·] ←File^r_(A) and SubDist_2[·] ←File^r_(B)
3.     for i = 0...2^m - 1
4.        bfly(SubDist_1[i], SubDist_2[i])
5.     save File^r_(A) ←SubDist_1[·] and File^r_(B) ←SubDist_2[·]
```

This algorithm requires to load/save each file $r = n - m$ times. The modified butterfly function `bfly*` can also be implemented memoryless. It can read one value from $\text{File}^r_{(A)}$ and one value from $\text{File}^r_{(B)}$, perform the usual butterfly operation and save the results back to the files immediately. There are two additional ideas to accelerate the FHT evaluation:

(a) In steps 3 and 4 of the algorithm above, only two files are processed. Instead, we could have a larger block of 2^d files opened and processed at the same time. The calculation of the butterfly function on two probabilities $\texttt{SubDist}_1[i]$ and $\texttt{SubDist}_2[i]$ can be substituted by a 'local' FHT on 2^d inputs, instead. Since the size of each file is 2^m, we need to repeat this procedure 2^m times for each group of 2^d files (inputs are taken in parallel from a group of 2^d files opened at the same time, but the number of such parallel inputs for each group is 2^m). As a result, each file is accessed approximately $(r+1)/d$ times.

(b) The computation can also be split into 2^c independent processes (2^c computers), and the results can be merged together afterwards.

4.3.5 Convolution over $\boxplus$

A convolution over $\boxplus$ on the suggested data structures can be done in a similar way as for $\oplus$. In the first step we perform the *bit reversing* operation on the input distribution, as described in Appendix B.3. Afterwards, we use the same idea as in the previous subsection, based on the *parallel FFT circuit*. The description of the parallel FFT circuit can be found in the book [CLRS01].

4.4 Application Example: 32 bit Cryptanalysis of SNOW 2.0

In this section we apply our results in linear cryptanalysis. We consider the stream cipher SNOW 2.0 and we operate with large distributions to achieve improved results.

A stream cipher is a cryptographic primitive used to ensure privacy on a communication channel. The SNOW family is a typical example of word-oriented KSGs based on a linear feedback shift register (LFSR). SNOW 2.0 is an improved version of SNOW 1.0 aimed to be more secure and still more efficient in performance. The most powerful attack on SNOW 2.0 was presented by Watanabe, Biryukov and De Cannie're [WBC03] in 2003. It is a linear distinguishing attack similar to the general framework presented in [CHJ02, Gol96] and it requires a received keystream sequence of 2^{225} bits in length and has a similar time complexity.

In this section we propose an improved attack on SNOW 2.0. Whereas the attack in [WBC03] uses a binary linear approximation approach, the new attack is based on approximations of words, i.e., 32 bit vectors. This technique is more powerful and we get a reduction of the required keystream length to 2^{202}. To make the calculation of 32 bit distributions possible we use algorithms and data structures described in the previous two sections.

4.4.1 A Short Description of SNOW 2.0

The structure of SNOW 2.0 is shown in Figure 4.1. It has 128 or 256 bit secret key and a 128 bit initial vector. It is based on LFSR over $\mathbb{F}_{2^{32}}[x]$ and the feedback polynomial is given by

$$\pi(x) = \alpha x^{16} + x^{14} + \alpha^{-1} x^5 + 1, \tag{4.30}$$

where α is a root of the polynomial

$$y^4 + \beta^{23}y^3 + \beta^{245}y^2 + \beta^{48}y + \beta^{239} \in \mathbb{F}_{2^8}[y], \tag{4.31}$$

and β is a root of

$$z^8 + z^7 + z^5 + z^3 + 1 \in \mathbb{F}_2[z]. \tag{4.32}$$

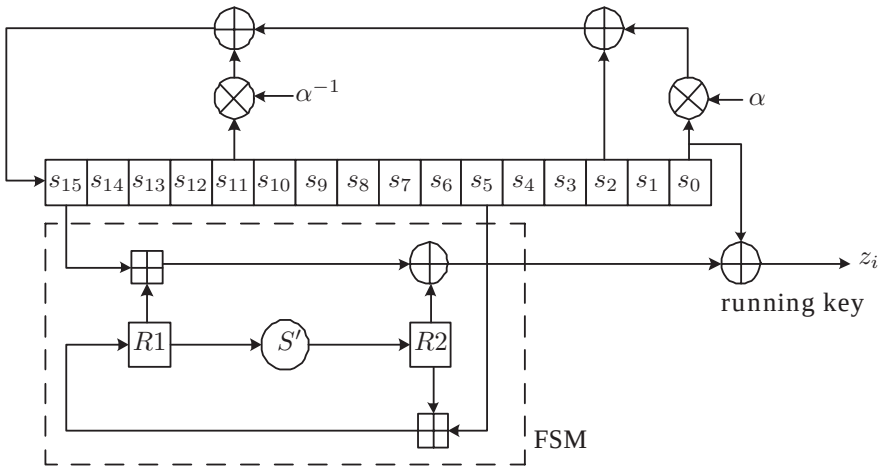


Figure 4.1: The structure of SNOW 2.0

The state of the LFSR is denoted by $(s_{t+15}, s_{t+14}, \ldots, s_t)$. Each s_{t+i} is an element of the field $\mathbb{F}_{2^{32}}$. The Finite State Machine (FSM) has two 32 bit registers, $R1$ and $R2$. The output of the FSM F_i is given by

$$F_i = (s_{t+15} \boxplus R1_t) \oplus R2_t, \quad t \geq 0, \tag{4.33}$$

and the keystream z_t is given by

$$z_t = F_t \oplus s_t, \quad t \geq 1. \tag{4.34}$$

Two registers $R1$ and $R2$ are updated as follows,

$$\begin{aligned} R1_{t+1} &= s_{t+5} \boxplus R2_t, \\ R2_{t+1} &= S'(R1_t). \end{aligned} \tag{4.35}$$

where $S'(W)$ is a one-to-one mapping transformation $S' : \mathbb{F}_{2^{32}} \rightarrow \mathbb{F}_{2^{32}}$. If a 32 bit integer W is represented as a vector of four 8 bit bytes $W =$

$(\; w_0 \quad w_1 \quad w_2 \quad w_3 \;)^{\mathrm{T}}$, then

$$S'(W) = \begin{pmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{pmatrix} \cdot \begin{pmatrix} S_R[w_0] \\ S_R[w_1] \\ S_R[w_2] \\ S_R[w_3] \end{pmatrix}, \tag{4.36}$$

where S_R is the Rijndael 8-to-8 bit S-box, and the linear transformation (matrix multiplication) is done in the field $\mathbb{F}_{2^8}$ with generating polynomial

$$g(x) = x^8 + x^4 + x^3 + x + 1 \in \mathbb{F}_2[x]. \tag{4.37}$$

4.4.2 Basic Idea Behind the New Attack

The basic idea behind the new attack is to find such a linear combination of the output words z_i that is equal to 0 if the system is linear, or that some biased noise is produced if the system is approximated by a linear function. On the other hand, the linear combination representing the noise should be unbiased if the given sequence z_i is truly random.

Consider the feedback polynomial of the LFSR given in equation (4.30), i.e., $\pi(x) = \alpha x^{16} + x^{14} + \alpha^{-1}x^5 + 1$. A similar relation holds for the LFSR's output s_t at any time t, i.e.,

$$s_{t+16} \oplus \alpha^{-1} s_{t+11} \oplus s_{t+2} \oplus \alpha s_t = 0, \quad t \geq 1. \tag{4.38}$$

Next we make an approximation of the FSM to make it look linear. For any time $t \geq 1$ two output words z_t and z_{t+1} can be expressed as

$$\begin{cases} z_t &= s_t \oplus (R1 \boxplus s_{t+15}) \oplus R2 \\ z_{t+1} &= s_{t+1} \oplus S'(R1) \oplus (R2 \boxplus s_{t+5} \boxplus s_{t+16}). \end{cases} \tag{4.39}$$

Let us substitute $\boxplus \rightarrow \oplus$ and change $S'(R) \rightarrow R$. Then the sum $z_t \oplus z_{t+1}$ is expressed as

$$\begin{aligned} z_t \oplus z_{t+1} &= s_t \oplus (R1 \oplus s_{t+15} \oplus N_{c2}(R1, s_{t+15})) \oplus R2 \\ &\quad \oplus s_{t+1} \oplus (R1 \oplus N_S(S'(R1), R1)) \\ &\quad \oplus (R2 \oplus s_{t+5} \oplus s_{t+16} \oplus N_{c3}(R2, s_{t+5}, s_{t+16})) \\ &= s_t \oplus s_{t+1} \oplus s_{t+5} \oplus s_{t+15} \oplus s_{t+16} \oplus N_0(t), \end{aligned} \tag{4.40}$$

where $N_0(t)$ is a variable representing the error introduced by the linear approximation in time t,

$$N_0(t) = N_{c2}(R1, s_{t+15}) \oplus N_S(S'(R1), R1) \oplus N_{c3}(R2, s_{t+5}, s_{t+16}). \tag{4.41}$$

Here $N_{c2}(R1, s_{t+15})$ is a noise random variable introduced by the approximation of the modulo sum of two variables of the kind "$R1 \boxplus s_{t+15} \rightarrow$

$R1 \oplus s_{t+15} \oplus N_{c2}$". The variable $N_{c3}(R2, s_{t+5}, s_{t+16})$ is a similar approximation noise, but for the modulo sum of three variables. Finally, $N_S(S'(R1), R1))$ is the noise variable from the approximation "$S'(R1) \rightarrow R1 \oplus N_S$". Let us derive a linear relation, based on (4.38).

$$
\begin{aligned}
0 \overset{\text{Eq(4.38)}}{=} & (s_{t+16} \oplus \alpha^{-1} s_{t+11} \oplus s_{t+2} \oplus \alpha s_t) \oplus (s_{t+17} \oplus \alpha^{-1} s_{t+12} \oplus_{t+3} \oplus \alpha s_{t+1}) \\
& \oplus (s_{t+21} \oplus \alpha^{-1} s_{t+16} \oplus s_{t+7} \oplus \alpha s_{t+5}) \oplus (s_{t+31} \oplus \alpha^{-1} s_{t+26} \oplus s_{t+17} \\
& \oplus \alpha s_{t+15}) \oplus (s_{t+32} \oplus \alpha^{-1} s_{t+27} \oplus s_{t+18} \oplus \alpha s_{t+16}) \\
= & (s_{t+16} \oplus s_{t+17} \oplus s_{t+21} \oplus s_{t+31} \oplus s_{t+32}) \oplus \alpha^{-1} \cdot (s_{t+11} \oplus s_{t+12} \\
& \oplus s_{t+16} \oplus s_{t+26} \oplus s_{t+27}) \oplus (s_{t+2} \oplus s_{t+3} \oplus s_{t+7} \oplus s_{t+17} \oplus s_{t+18}) \\
& \oplus \alpha \cdot (s_t \oplus s_{t+1} \oplus s_{t+5} \oplus s_{t+15} \oplus s_{t+16}) \\
\overset{\text{Eq(4.39)}}{=} & (z_{t+2} \oplus z_{t+3} \oplus z_{t+16} \oplus z_{t+17}) \oplus \alpha^{-1} \cdot (z_{t+11} \oplus z_{t+12}) \\
& \oplus \alpha \cdot (z_t \oplus z_{t+1}) \oplus (N_0(t+2) \oplus N_0(t+16)) \oplus \alpha^{-1} \cdot N_0(t+11) \\
& \oplus \alpha \cdot N_0(t) \;=\; \mathbf{Z}(t) \oplus \mathbf{N}(t),
\end{aligned}
\tag{4.42}
$$

where $\mathbf{N}(t)$ is the 32 bit total sum of noise variables introduced by several approximations, expressed as

$$\mathbf{N}(t) = (N_0(t+2) \oplus N_0(t+16)) \oplus \alpha^{-1} \cdot N_0(t+11) \oplus \alpha \cdot N_0(t), \tag{4.43}$$

and $\mathbf{Z}(t)$ is the "known" part calculated from the output sequence at any time t,

$$\mathbf{Z}(t) = (z_{t+2} \oplus z_{t+3} \oplus z_{t+16} \oplus z_{t+17}) \oplus \alpha^{-1}(z_{t+11} \oplus z_{t+12}) \oplus \alpha(z_t \oplus z_{t+1}). \tag{4.44}$$

Obviously, $\mathbf{N}(t) \oplus \mathbf{Z}(t) = 0$.

After all, a linear distinguishing attack can now be performed, if we know the distribution $P_{\mathbf{N}}$ of the 32 bit noise variable $\mathbf{N}$. For a sufficiently large number of received symbols from either the random distribution P_{Random}, or the distribution of the noise $P_{\mathbf{N}}$, one can construct the *type* (or *empirical distribution*) P_{Type}. We then make a decision whether the stream comes from a truly random generator or from the cipher, according to the distances from P_{Type} to $P_{\mathbf{N}}$ and P_{Random}. Note that the 32 bit noise distribution definitely contains the best binary approximation found in [WBC03], but it also contains some additional information, which makes the bias of the noise larger.

The procedure of a distinguishing attack when two distributions are known is described in detail in Section 3.4.5.

4.4.3 Computational Aspects

We adopted the data structures from Section 4.3 for our simulations as follows: we use 2^{10} files, each containing 2^{22} points of a sub distribution. Since the precision of the probabilities has to be at least $2^{-(192\cdot 4+32)}$ (four noises N_0, each containing N_S with precision 2^{-32}, N_{c2} with precision 2^{-64}, and N_{c3} with precision 2^{-96}; plus 32 bits must be reserved for FHT), each cell has to be of the minimum size of 100 bytes. I.e., each sub distribution in the memory takes at least 400Mb. However, this estimate is conservative, and in our simulations we used almost 2Gb of operation memory.

To calculate the bias of the 32 bit noise variable $\mathbf{N}$, its distribution table has to be constructed. It can be calculated via the distribution of N_0, expressed in (4.41).

To construct the distributions of N_{c2} and N_{c3} we use Theorem 4.1 (PLFM construction). The expression for N_S is a function on one variable, i.e., it takes no more than $O(2^{32})$ operations to build the distribution P_{N_S}. Next, the distribution of N_0 is calculated via FHT with the algorithm from Section 4.2 (convolution over $\oplus$) and Section 4.3.4 (FHT for large distributions). Afterwards, the distribution of $\alpha \cdot N_0$ and $\alpha^{-1} \cdot N_0$ is computed using algorithms described in Section 4.3.3 (function evaluation). Finally, we use FHT to calculate the distribution of the total noise variable $P_{\mathbf{N}}$, and then calculate the bias $\epsilon = |P_{\mathbf{N}} - P_{\text{Random}}|$.

All these operations took us less than 2 weeks on a usual Pentium IV 3.4GHz, 2Gb of memory and 256Gb of HDD.

4.4.4 Simulation Results and Discussion

At the end of our simulations we received the distance $\epsilon = |P_{\mathbf{N}} - P_{\text{Random}}| \approx 2^{-101}$, which means that SNOW 2.0 can be distinguished from random with the known keystream of size 2^{202}, and with a similar time complexity. The advantage of our attack is presented in the following table.

Attack on SNOW 2.0	bit(s) considered	bias (ϵ)	complexity
Watanabe et al. [WBC03]	1	$2^{-112.25}$	2^{225}
our attack	32	2^{-101}	2^{202}

For future research on this topic, it is important to note that the expression for the noise variable $\mathbf{N}(t)$ (4.42) contains two parts: $N_{c3}(R2_t, s_{t+5}, s_{t+16})$ and $N_{c3}(R2_{t+11}, s_{t+16}, s_{t+27})$, which were in our simulations, considered as independent. However, since they both use the same input s_{t+16}, they are not really independent and, theoretically, the result should be slightly improved if one considers them mutually dependent.

4.5 Summary

In this chapter we have proposed new algorithms for computation of distributions of certain functions where input variables are from a large alphabet. When the input variables were uniformly distributed, the distribution for a class of functions called PLFM was shown to be efficiently calculated. The second case considered the same problem but for arbitrary distribution of input variables. Efficient methods for calculating the distribution of sums of variables both in $\mathbb{Z}_{2^n}$ and $\mathbb{F}_{2^n}$ were proposed, based on Fast Fourier Transform and Fast Hadamard Transform, respectively.

The cryptologic applications of the results were demonstrated by extending the linear cryptanalysis of the stream cipher SNOW 2.0 to work over a larger alphabet. We believe that there are many instances of stream ciphers as well as block ciphers, where cryptanalytic results can be improved by considering analysis over a larger alphabet. In all these cases, the algorithms derived in this chapter will be useful for calculating the performance of such attacks.

We also believe that the technique considering "local carries" presented in algorithms for PLFMs can easily be transformed to find *one* or even *all solutions* for equations like

$$F(X_1, \ldots, X_k) = 0. \tag{4.45}$$

Finding solutions for other kinds of equations, including $F(X_1, \ldots, X_k) = \gamma$ and systems of equations, depends on finding solutions for an equation of the first kind. Consequently, many properties of PLFM functions can be derived, as it was done for smaller classes in, e.g., [LM02, LWD04, Lip02].

Recently, at the conference FSE 2006, an improved distinguishing attack on SNOW 2.0 was presented [NW06], which requires 2^{174} samples. The authors exploited a better binary approximation of the `MixColumn` transformation used in the cipher, and estimated the bias of the FSM approximation in an accurate way. We believe that the use of multiple approximations jointly would improve their results as well. In order to perform calculations over multidimentional distributions, the techniques from this chapter can be used.

A few open problems can be mentioned. For example, we would like to find other classes of functions for which their distributions can be computed efficiently. We would also like to recognize further instances of existing ciphers where linear attacks over larger alphabets are applicable.

5

Cryptanalysis of A5/1

Ericsson's telephone *Eiffel Tower*, 1892

"The secret of getting ahead is getting started. The secret of getting started is breaking your complex overwhelming tasks into small manageable tasks, and then starting on the first one"

Mark Twain

The security of GSM conversation is based on usage of the A5 family of stream ciphers. Many hundred million customers in Europe are protected from over-the-air piracy by the stronger version in this family, the A5/1 stream cipher. Other customers on other markets use the weaker version, A5/2. The approximate design of A5/1 leaked in 1994, and in 1999 the exact design of both A5/1 and A5/2 was discovered by Briceno [BGW99]. A lot of investigations of the A5 stream ciphers followed.

The first analysis of the A5/1 cipher resulted in "Guess-and-Determine" type of attacks [Gol97a]. Then a time-memory trade-off attack was proposed by Biryukov, Shamir, and Wagner [BSW00], which in some cases can break A5/1 in seconds. Unfortunately, it needs to use a huge precomputational time and about 4×73Gb of hard memory. The attack complexity

grows exponentially depending on the length of the LFSRs in the design of a cipher. Another attack was presented by Biham and Dunkelman [BD00]. Their attack breaks the cipher within $2^{39.91}$ A5/1 clocking assuming $2^{20.8}$ bits of keystream available. This attack has expensive asymptotic behaviour. In 2002, Krause, [Kra02] presented a general attack on LFSR-based stream ciphers, called BDD-based cryptanalysis. This attack requires computation complexity of $n^{O(1)}2^{an}, a < 1$ polynomial time operations, where a is a constant depending on the cipher and n is the combined shift registers length. For A5/1, the attack achieves $a = 0.6403$, so the complexity is again exponential with the shift registers length.

A completely different way to attack A5/1 was proposed by Ekdahl and Johansson in 2001 [EJ01]. The attack needs a few minutes for computations, and 2-5 minutes of conversation (plaintext). The idea behind the attack came from correlation attacks. This is the only attack for which the complexity does not grow exponentially with the shift register's length.

Finally, Barkan, Biham and Keller [BBK03] investigated the usage of the A5 ciphers in GSM. They demonstrated an active attack where a false base station can intercept a conversation and perform a man in the middle attack. By asking for usage of the weak A5/2 algorithm in the conversation with the base station and then breaking it, the false base station finds the session key which is also used in the A5/1 protected conversation with the mobile unit. In [BBK03] the authors also propose the passive memory-time trade-off ciphertext only attack. As one of the examples, if 5 minutes of conversation is available, then the attack needs one year of precomputations with 140 computers working together, 22×200GBs hard discs. Then the attack can be done in time 2^{28} by one PC. Obviously, the authors did not try to implement the attack and the complexity was just estimated.

In this chapter a new approach to attack the A5/1 stream cipher is proposed. We consider the Ekdahl-Johansson attack as the basis, and apply several new improvements. The new attack now needs only less than one minute of computations, and a few seconds of known conversation. It does not need any notable precomputation time, and needs reasonable space of operation memory.

In the case of a ciphertext-only attack on A5/1, we start from the fact that some redundancy is part of the plaintext. There are at least two kinds of redundancy that are explicit and may be used in an attack where only ciphertext is available. *The first* kind is the fact that coding is done before encryption, which results in linear relationships in the plaintext since the parity check symbols are also encrypted. This observation was used in [BBK03]. *The second* kind of redundancy is the fact that during silence, a special frame including a large number of zeros is sent [vS00]. Silence occurs very often, but unfortunately these frames used for silence are trans-

mitted less frequently, one to initialise a period of silence and then two each second. The attack that we propose can be considered in a ciphertext-only scenario, in which case we use this redundancy during silence to get some known outputs from the cipher.

Although several of the previous attacks are sufficient to break A5/1 in a known plaintext attack, we believe that further progress is very important. The A5/1 stream cipher is perhaps the most used cipher in the world, and from the wireless communication channel interception of the communication is very easy. Mobile base stations are not expensive to buy and they can be used to record GSM conversations.

Although A5/1 is generally regarded as "broken", one may look at the practical difficulty of actually getting access to a conversation, when the encrypted communication data is recorded. Previous attacks assume a known plaintext, which is not really what we have in practice. In practice we have a ciphertext-only attack, but the plaintext includes some known redundancy. The only previous attack considering this situation is the one by Barkan, Biham and Keller [BBK03]. This attack is a very nice idea, but as it is an active attack it requires the man in the middle attack to be performed in real time. A less complex case to perform in practice would be to assume that we record an encrypted conversation to disk and then later analyse it to recover the conversation. This is one case we have in mind when we present our new algorithm.

This chapter is organized as follows. In Section 5.1 a short description of the cipher A5/1 is given. The basic Ekdahl-Johansson attack on A5/1 is briefly described in Section 5.2. Then, in Section 5.3, we give new ideas to improve the attack in general. The details and particulars of the attack simulations are described in Section 5.3.2. In Section 5.4 the results of our simulations are presented. Finally, we summarize the status of the A5/1 stream cipher.

5.1 Description of A5/1

A GSM conversation between A and B is a sequence of frames, each sent in about 4.6 milliseconds. Each frame consists of 228 bits – 114 bits of which is the message from A to B, and the second half bits are representing communication from B to A. One session is encrypted with a secret *session key* K. For the jth frame the running key generator is initialised with mixture of K and the publicly known *frame counter*, denoted by F_j. It then generates 228 bits of running key for the current frame. The ciphertext is a binary xor of the running key and the plaintext.

A5/1 consists of 3 LFSRs of lengths 19, 22, and 23, which are denoted R_1, R_2, and R_3, respectively. The LFSRs are clocked in an irregular fashion.

values of			clocking		
C_1	C_2	C_3	R_1	R_2	R_3
$1 \oplus c$	c	c	$\times$	$\surd$	$\surd$
c	$1 \oplus c$	c	$\surd$	$\times$	$\surd$
c	c	$1 \oplus c$	$\surd$	$\surd$	$\times$
c	c	c	$\surd$	$\surd$	$\surd$

Table 5.1: Taps majority as the clocking control unit for A5/1.

Each of them has one tap-bit, C_1, C_2, and C_3, respectively. In each step, 2 or 3 LFSRs are clocked, depending on the current values of the bits C_1, C_2, and C_3. Thus, the clocking control device implements the majority rule, shown in Table 5.1. Note, for each step the probability that an individual LFSR is being clocked is $3/4$.

After the initialisation procedure for the LFSRs, 228 bits of running key are produced, using irregular clocking. In each step one bit of the running key is calculated as the binary xor of the current output bits from the LFSRs.

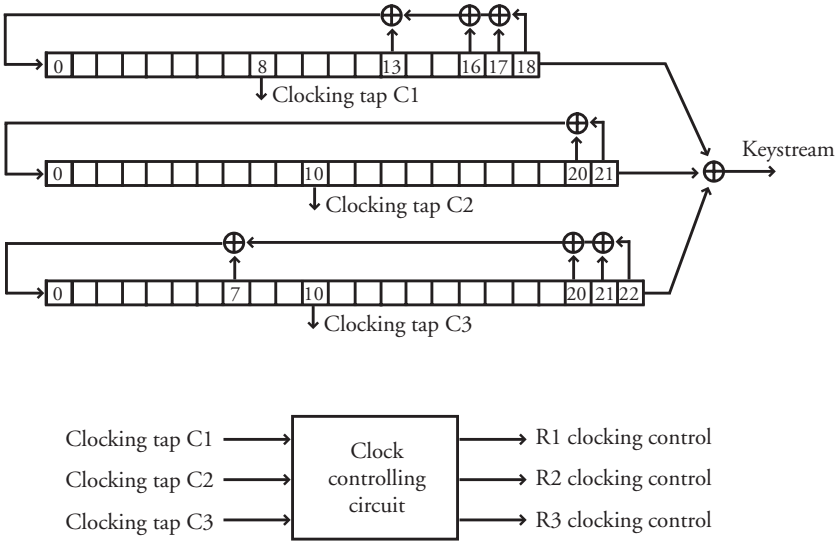


Figure 5.1: The structure of A5/1 cipher

The initialisation process uses the session key K and the known frame counter F_n. First the LFSRs are initialised to zero. They are then clocked 64 times, ignoring the irregular clocking, and the key bits of K are consecutively xored in parallel to the feedback of each of the registers. In the second step the LFSRs are clocked 22 times, ignoring the irregular clocking, and the

successive bits of F_n are again xored in parallel to the feedback of the LFSRs. Let us call the state of LFSRs at this time the *initial state* of the frame. In the third step the LFSRs are clocked 100 times *with* irregular clocking, but ignoring outputs. Then, the LFSRs are clocked 228 times with the irregular clocking, producing 228 bits of the running key. For a more detailed description of A5/1 we refer to [BGW99].

5.2 A Short Description of the Ekdahl-Johansson Attack on A5/1

This attack was proposed in 2002 by Ekdahl and Johansson. The idea behind the attack came from correlation attacks, and is based on the linearity of the initialisation procedure. The attack needs a set of m frames (about 20000-50000 in their attack), during one session, i.e., when the session key K is not changed.

For notation purposes, let the key $K = (k_1, \ldots, k_{64})$, and the frame counter $F_j = (f_1, \ldots, f_{22})$, where $k_i, f_j \in \mathbb{F}_2$, $i = 1..64, j = 1..22$. Denote by $u_1^j(l_1)$, $u_2^j(l_2)$, and $u_3^j(l_3)$ the output bits of LFSRs, if they are independently *clocked* l_1, l_2, and l_3 times, respectively, *after* the LFSRs being in the initial state, and when the current frame is number j. The 228 bits of the running key are then denoted as $v^j(101), \ldots, v^j(100+228)$, and every

$$v^j(t) = u_1^j(l_1) \oplus u_2^j(l_2) \oplus u_3^j(l_3), \tag{5.1}$$

for *some unknown* l_1, l_2, l_3.

Note, that $u_1^j(l_1)$ is a linear combination of K and F_j bits, since all operations before the initial state are linear. I.e., $u_1^j(l_1)$ can be represented as

$$u_1^j(l_1) = X_{1,l_1}(F_j) + Y_{1,l_1}(K), \tag{5.2}$$

where $X_{1,l_1}(F_j)$ is a known fixed value and $Y_{1,l_1}(K) = \sum_{i=1}^{64} y_{1,l_1,i} \cdot k_i$ is a linear function with known coefficients $y_{1,l_1,i} \in \mathbb{F}_2$.

With the same arguments we define

$$\begin{aligned} u_2^j(l_2) &= X_{2,l_2}(F_j) + Y_{2,l_2}(K), \\ u_3^j(l_3) &= X_{3,l_3}(F_j) + Y_{3,l_3}(K), \end{aligned} \tag{5.3}$$

where $X_{a,l_a}(F_j)$ and the coefficients $y_{a,l_a,i} \in \mathbb{F}_2$, for $a = 2, 3$, $l_a = 0, 1, \ldots, 100 + 228$, $i = 1, \ldots, 64$ are precomputed and fixed. Let us write

$$\begin{aligned} s_1(l_1) &= Y_{1,l_1}(K), \\ s_2(l_2) &= Y_{2,l_2}(K), \\ s_3(l_3) &= Y_{3,l_3}(K). \end{aligned} \tag{5.4}$$

Our target is to estimate 19 bits from the first LFSR $s_1(0), \ldots, s_1(18)$, 22 bits from the second LFSR $s_2(0), \ldots, s_2(21)$, and 23 bits from the third LFSR $s_3(0), \ldots, s_3(22)$. These 64 bits map one-to-one to 64 bits of the key K, if the frame counter F_j is given.

REMARK: For notation purposes we write $E \stackrel{p}{=} \hat{E}$, when $\hat{E}$ appears to be an estimator for E, such that $\Pr\{E = \hat{E}\} = p$, for some probability p. $\hat{E}$ can be derived from accessible data, or assumed (guessed).

One can think about the data we have access to as a binary table of m frames in the form

$$\begin{pmatrix} v^1(101) & v^1(102) & \ldots & v^1(100+228) \\ v^2(101) & v^2(102) & \ldots & v^2(100+228) \\ & \vdots & & \\ v^m(101) & v^m(102) & \ldots & v^m(100+228) \end{pmatrix}. \tag{5.5}$$

The idea behind the attack is the observation that

$$v^j(101) \stackrel{p}{=} s_1(l_1) + s_2(l_2) + s_3(l_3) + X_{1,l_1}(F_j) + X_{2,l_2}(F_j) + X_{3,l_3}(F_j) \tag{5.6}$$

for some $p \neq 1/2$, if l_1, l_2, l_3 are chosen properly. The probability p is

$$p = \frac{1}{2} + \frac{1}{2}\Pr\{(l_1, l_2, l_3) \text{ at time } t\}, \tag{5.7}$$

where $\Pr\{(l_1, l_2, l_3)$ at time $t\}$ is the probability that at time 101 the LFSRs were regularly clocked exactly l_1, l_2, l_3 times, respectively. The probability that at time $t \in \{101 \ldots 100 + 228\}$, the LFSRs have been clocked (l_1, l_2, l_3) times is

$$\Pr\{(l_1, l_2, l_3) \text{ at time } t\} = \frac{\binom{t}{t-l_1}\binom{t-(t-l_1)}{t-l_2}\binom{t-(t-l_1)-(t-l_2)}{t-l_3}}{4^t}. \tag{5.8}$$

Let us now define the known value

$$\hat{O}^j_{l_1,l_2,l_3}(t) = v^j(t) \oplus X_{1,l_1}(F_j) \oplus X_{2,l_2}(F_j) \oplus X_{3,l_3}(F_j). \tag{5.9}$$

Then we have

$$\hat{O}^j_{l_1,l_2,l_3}(t) \stackrel{p}{=} s_1(l_1) \oplus s_2(l_2) \oplus s_3(l_3). \tag{5.10}$$

The case when $\hat{O}^j_{l_1,l_2,l_3}(t)$ is equal to the value $s_1(l_1) \oplus s_2(l_2) \oplus s_3(l_3)$ can happen only in two ways.

a) The LFSRs are really clocked l_1, l_2, l_3 at time t, happening with probability $\Pr\{(l_1, l_2, l_3)$ at time $t\}$. If so, the expression will be true with probability 1.

b) If the condition in a) is not fulfilled, the expression will still be true with probability $1/2$.

This means that the relation (5.10) is biased ($p > 1/2$).

From the given frames we can estimate many of the linear combinations $s_1(l_1) \oplus s_2(l_2) \oplus s_3(l_3)$ for different triples (l_1, l_2, l_3). But we only need 64 correct estimates in order to recover the key K uniquely.

To minimise the amount of frames m and perform the estimation with low probability of error, Ekdahl and Johansson suggested to use the values of $v^j(101), \ldots, v^j(164)$ for all j for better estimation of $s_1(l_1) \oplus s_2(l_2) \oplus s_3(l_3)$. The following expression can be used.

$$\begin{aligned}\Pr\{s_1(l_1) \oplus s_2(l_2) \oplus s_3(l_3) = 1, \text{ for the frame } j\} &= p^j_{(l_1,l_2,l_3)} = \\ &= \sum_{t \in \{101 \ldots 164\}} \Pr\{(l_1, l_2, l_3) \text{ at time } t\} \cdot \left[\hat{O}^j_{l_1,l_2,l_3}(t) = 0\right] \\ &\quad + 1/2 \cdot \left(1 - \sum_{t \in \{101 \ldots 164\}} \Pr\{(l_1, l_2, l_3) \text{ at time } t\}\right). \qquad (5.11)\end{aligned}$$

This probability gives the estimation of the corresponding linear combination for one frame j. We will increase the possibility to estimate the value of $s_1(l_1)+s_2(l_2)+s_3(l_3)$ correctly, when m frames (samples) $v^1(101 \ldots 328), \ldots, v^m(101 \ldots 328)$ are given, as each of them provides some small contribution. By calculating the likelihood ratio

$$\Lambda_{l_1,l_2,l_3} = \sum_{j=1}^{m} \log_2 \left[\frac{p^j_{(l_1,l_2,l_3)}}{1 - p^j_{(l_1,l_2,l_3)}}\right] \qquad (5.12)$$

we achieve a likelihood value (estimate) which is taken over all m frames. This can be turned into a binary estimate by

$$s_1(l_1) \oplus s_2(l_2) \oplus s_3(l_3) \stackrel{p}{=} \begin{cases} 0 & \text{if } \Lambda_{l_1,l_2,l_3} \geq 0 \\ 1 & \text{if } \Lambda_{l_1,l_2,l_3} < 0 \end{cases}, \qquad (5.13)$$

where $p > 0.5$ depends mainly on m. In [EJ01] the authors finally examine different strategies for implementing the recovery of the key bits as efficient as possible.

5.3 Explaining the New Attack

In this section we describe our discovered improvements in general. Our main purpose is to reduce the number of frames m, which is needed for the attack.

5.3.1 Statistical Analysis of m Frames

We mentioned before that we have identified two general ideas for improving the previous results. The first is the fact that it is beneficial to study the derivative sequences instead of the sequences themselves. Assume that at time t the LFSRs are clocked l_1, l_2, and l_3 times, respectively. Then we also assume that at time $t+1$ the third LFSR is not clocked. In this case we have the equalities,

$$\begin{aligned}\hat{O}^j_{l_1,l_2,l_3}(t) &= s_1(l_1) \oplus s_2(l_2) \oplus s_3(l_3),\\ \hat{O}^j_{l_1+1,l_2+1,l_3}(t+1) &= s_1(l_1+1) \oplus s_2(l_2+1) \oplus s_3(l_3).\end{aligned} \tag{5.14}$$

Then the probability

$$\begin{aligned}\Pr\{\hat{O}^j_{l_1,l_2,l_3}(t) \oplus \hat{O}^j_{l_1+1,l_2+1,l_3}(t+1) = s_1(l_1) \oplus s_2(l_2) \oplus s_1(l_1+1) \oplus s_2(l_2+1)\}\\ = \frac{1}{4} \cdot \Pr\{(l_1,l_2) \text{ at time } t\},\end{aligned} \tag{5.15}$$

where

$$\Pr\{(l_1,l_2) \text{ at time } t\} = \frac{\binom{t}{t-l_1}\binom{l_1}{t-l_2}}{2^{3t-(l_1+l_2)}}. \tag{5.16}$$

Note, that $\frac{1}{4} \cdot \Pr\{(l_1,l_2) \text{ at time } t\} > \Pr\{(l_1,l_2,l_3) \text{ at time } t\}$ so it gives us a larger bias when estimating the value of linear combinations of $s_i(l_i)$'s. Below is a comparison of these probabilities.

$(l_1,l_2,l_3),t$	$\Pr\{(l_1,l_2,l_3) \text{ at } t\} \cdot 10^4$	$\frac{1}{4}\Pr\{(l_1,l_2) \text{ at } t\} \cdot 10^4$
$(76,76,76),101$	9.7434	22.1207
$(79,79,79),105$	9.2012	21.2840
$(80,80,80),105$	6.6388	19.3778
$(79,80,81),106$	8.3858	20.8899
$(82,82,82),109$	8.7076	20.5083

The first idea to improve the attack is then to consider two consecutive expressions (5.14). Their sum only depends on two LFSRs, and the probability of the event is higher than before. We also note that we can similarly assume that LFSR-1 and LFSR-2 are not clocked at some time t. This gives us 3 *cases*. We define

$$\begin{aligned}{}_1\hat{z}^j_{l_2,l_3}(t) &= \hat{O}^j_{l_1,l_2,l_3}(t) \oplus \hat{O}^j_{l_1,l_2+1,l_3+1}(t+1) \stackrel{p}{=} s_2(l_2) \oplus s_3(l_3) \oplus s_2(l_2+1) \oplus s_3(l_3+1),\\ {}_2\hat{z}^j_{l_1,l_3}(t) &= \hat{O}^j_{l_1,l_2,l_3}(t) \oplus \hat{O}^j_{l_1+1,l_2,l_3+1}(t+1) \stackrel{p}{=} s_1(l_1) \oplus s_3(l_3) \oplus s_1(l_1+1) \oplus s_3(l_3+1),\\ {}_3\hat{z}^j_{l_1,l_2}(t) &= \hat{O}^j_{l_1,l_2,l_3}(t) \oplus \hat{O}^j_{l_1+1,l_2+1,l_3}(t+1) \stackrel{p}{=} s_1(l_1) \oplus s_2(l_2) \oplus s_1(l_1+1) \oplus s_2(l_2+1).\end{aligned} \tag{5.17}$$

The case when ${}_3\hat{z}^j_{l_1,l_2}(t)$ is equal to the value $s_1(l_1) \oplus s_2(l_2) \oplus s_1(l_1+1) \oplus s_2^j(l_2+1)$ can happen in two ways,

a) The first and the second LFSRs are indeed clocked l_1, l_2 times at time t occuring with probability $\Pr\{(l_1, l_2) \text{ at time } t\}$, **AND** at time $t+1$ the third LFSR *is not clocked*, with probability $1/4$. The expression is always true in this case.

b) If the condition in a) is not fulfilled the expression will still be true with probability $1/2$.

The second idea is to consider d consecutive estimators jointly as one d-dimension estimator. If we look at the sequence of d estimators of the form ${}_3\hat{z}^j_{l_1,l_2}(t), \ldots, {}_3\hat{z}^j_{l_1+d-1,l_2+d-1}(t+d-1)$, then we note that they depend on each other. To use this fact we suggest to consider not binary expressions, but vectors of d bits. Introduce a new d bits vector, derived from the frame j,

$$ {}_3\hat{\mathcal{Z}}^j_{l_1,l_2}(t) = \begin{pmatrix} {}_3\hat{z}^j_{l_1,l_2}(t) \\ {}_3\hat{z}^j_{l_1+1,l_2+1}(t+1) \\ \vdots \\ {}_3\hat{z}^j_{l_1+d-1,l_2+d-1}(t+d-1) \end{pmatrix} \tag{5.18} $$

$$ = \begin{pmatrix} v^j(t) \oplus v^j(t+1) \oplus X_{1,l_1}(j) \oplus X_{2,l_2}(j) \oplus X_{1,l_1+1}(j) \oplus X_{2,l_2+1}(j) \\ v^j(t+1) \oplus v^j(t+2) \oplus X_{1,l_1+1}(j) \oplus X_{2,l_2+1}(j) \oplus X_{1,l_1+2}(j) \oplus X_{2,l_2+2}(j) \\ \vdots \\ v^j(t+d-1) \oplus v^j(t+d) \oplus X_{1,l_1+d-1}(j) \oplus X_{2,l_2+d-1}(j) \oplus X_{1,l_1+d}(j) \oplus X_{2,l_2+d}(j) \end{pmatrix}. $$

Define the d-dimension vector ${}_3\mathcal{S}_{l_1,l_2}$ (which is unknown for the attacker) as

$$ {}_3\mathcal{S}_{l_1,l_2} = \begin{pmatrix} s_1(l_1) + s_2(l_2) + s_1(l_1+1) + s_2(l_2+1) \\ s_1(l_1+1) + s_2(l_2+1) + s_1(l_1+2) + s_2(l_2+2) \\ \vdots \\ s_1(l_1+d-1) + s_2(l_2+d-1) + s_1(l_1+d) + s_2(l_2+d) \end{pmatrix}. \tag{5.19} $$

Then, from (5.17) it follows that

$$ {}_3\mathcal{S}_{l_1,l_2} \stackrel{p}{=} {}_3\hat{\mathcal{Z}}^j_{l_1,l_2}(t), \tag{5.20} $$

with some biased probability p. Note that the symbols are now of alphabet size 2^d.

Examining this in more detail, consider d consecutive irregular steps. The total number of possible scenarios is 4^d, since in each step one of four

types of irregular clockings can be chosen, according to the bits C_1, C_2, C_3. If we assume that at time t the first and the second LFSRs are clocked exactly l_1, l_2 times, then we can classify the bits of the vector ${}_3\hat{\mathcal{Z}}^j_{l_1,l_2}(t)$. They can be either **C***orrect* (i.e., the next clocking is the required one so the bit has the same value as the corresponding bit in the vector ${}_3\mathcal{S}_{l_1,l_2}$), or **R***andom* (i.e., the bit can be 0 or 1, with probability 1/2). For each possible pattern $\{\mathbf{C}orrect, \mathbf{R}andom\}^d$ we calculate the corresponding number of scenarios out of 4^d possible, by exhaustively trying all the scenarios. For example, when $d = 4$, we have the following distribution:

Condition	${}_3\hat{\mathcal{Z}}^j_{l_1,l_2}(t)$				Probability	Event
	${}_3\hat{z}^j_{l_1,l_2}(t)$	${}_3\hat{z}^j_{l_1+1,l_2+1}(t+1)$	${}_3\hat{z}^j_{l_1+2,l_2+2}(t+2)$	${}_3\hat{z}^j_{l_1+3,l_2+3}(t+3)$		
Assumption is NOT correct	**R***andom*	**R***andom*	**R***andom*	**R***andom*	$1 - P_0$	E_R
Assumption is correct	**C***orrect*	**C***orrect*	**C***orrect*	**C***orrect*	$P_0 \cdot 1/2^8$	E_0
	R*andom*	**C***orrect*	**C***orrect*	**C***orrect*	$P_0 \cdot 1/2^8$	E_1
	C*orrect*	**R***andom*	**C***orrect*	**C***orrect*	$P_0 \cdot 1/2^8$	E_2
	R*andom*	**R***andom*	**C***orrect*	**C***orrect*	$P_0 \cdot 1/2^8$	E_3
	C*orrect*	**C***orrect*	**R***andom*	**C***orrect*	$P_0 \cdot 1/2^8$	E_4
	R*andom*	**C***orrect*	**R***andom*	**C***orrect*	$P_0 \cdot 1/2^8$	E_5
	C*orrect*	**R***andom*	**R***andom*	**C***orrect*	$P_0 \cdot 1/2^8$	E_6
	R*andom*	**R***andom*	**R***andom*	**C***orrect*	$P_0 \cdot 1/2^8$	E_7
	C*orrect*	**C***orrect*	**C***orrect*	**R***andom*	$P_0 \cdot 3/2^8$	E_8
	R*andom*	**C***orrect*	**C***orrect*	**R***andom*	$P_0 \cdot 3/2^8$	E_9
	C*orrect*	**R***andom*	**C***orrect*	**R***andom*	$P_0 \cdot 3/2^8$	E_{10}
	R*andom*	**R***andom*	**C***orrect*	**R***andom*	$P_0 \cdot 3/2^8$	E_{11}
	C*orrect*	**C***orrect*	**R***andom*	**R***andom*	$P_0 \cdot 11/2^8$	E_{12}
	R*andom*	**C***orrect*	**R***andom*	**R***andom*	$P_0 \cdot 11/2^8$	E_{13}
	C*orrect*	**R***andom*	**R***andom*	**R***andom*	$P_0 \cdot 43/2^8$	E_{14}
	R*andom*	**R***andom*	**R***andom*	**R***andom*	$P_0 \cdot 171/2^8$	E_{15}

where $P_0 = \Pr\{(l_1, l_2) \text{ at time } t\}$ and the assumption is that the first two LFSRs have clocked (l_1, l_2) at time t.

Let us assume that we have received the vector ${}_3\hat{\mathcal{Z}}^j_{l_1,l_2}(t) = (0,1,1,0)^{\mathrm{T}}$ at time t from the frame j. If we consider the hypothesis that ${}_3\mathcal{S}_{l_1,l_2} = (0,0,1,1)$, then the error pattern is $\mathcal{E}_d = {}_3\mathcal{S}_{l_1,l_2} \oplus {}_3\hat{\mathcal{Z}}^j_{l_1,l_2}(t) = (0,1,0,1)$. This *error pattern* $\mathcal{E}_d$ can be the result of one of the following events: $E_R, E_{10}, E_{11}, E_{14}, E_{15}$. Thus, the conditional probability

$$\begin{aligned}\Pr\{{}_3\mathcal{S}_{l_1,l_2} = (0,0,1,1) | {}_3\hat{\mathcal{Z}}^j_{l_1,l_2}(t) = (0,1,1,0)\} &= \Pr\{\mathcal{E}_d = (0,1,0,1)\} \\ &= \frac{\Pr\{E_R\}}{2^4} + \frac{\Pr\{E_{10}\}}{2^2} + \frac{\Pr\{E_{11}\}}{2^3} + \frac{\Pr\{E_{14}\}}{2^3} + \frac{\Pr\{E_{15}\}}{2^4} \\ &= (1-P_0)/2^4 + P_0 \cdot 275/2^{12}. \end{aligned} \tag{5.21}$$

Continuing in this way, the complete table for $\Pr\{\mathcal{E}_d\}$ can be derived. The distribution for $d = 4$ is given as in the table on the right.

$\mathcal{E}_d = {}_3\mathcal{S}_{l_1,l_2} \oplus {}_3\hat{\mathcal{Z}}^j_{l_1,l_2}(t)$	
$\mathcal{E}_d$	$\Pr\{\mathcal{E}_d\}$
(0, 0, 0, 0)	$(1-P_0)/2^4 + P_0 \cdot 431/2^{12}$
(1, 0, 0, 0)	$(1-P_0)/2^4 + P_0 \cdot 229/2^{12}$
(0, 1, 0, 0)	$(1-P_0)/2^4 + P_0 \cdot 293/2^{12}$
(1, 1, 0, 0)	$(1-P_0)/2^4 + P_0 \cdot 183/2^{12}$
(0, 0, 1, 0)	$(1-P_0)/2^4 + P_0 \cdot 341/2^{12}$
(1, 0, 1, 0)	$(1-P_0)/2^4 + P_0 \cdot 199/2^{12}$
(0, 1, 1, 0)	$(1-P_0)/2^4 + P_0 \cdot 263/2^{12}$
(1, 1, 1, 0)	$(1-P_0)/2^4 + P_0 \cdot 173/2^{12}$
(0, 0, 0, 1)	$(1-P_0)/2^4 + P_0 \cdot 377/2^{12}$
(1, 0, 0, 1)	$(1-P_0)/2^4 + P_0 \cdot 211/2^{12}$
(0, 1, 0, 1)	$(1-P_0)/2^4 + P_0 \cdot 275/2^{12}$
(1, 1, 0, 1)	$(1-P_0)/2^4 + P_0 \cdot 177/2^{12}$
(0, 0, 1, 1)	$(1-P_0)/2^4 + P_0 \cdot 323/2^{12}$
(1, 0, 1, 1)	$(1-P_0)/2^4 + P_0 \cdot 193/2^{12}$
(0, 1, 1, 1)	$(1-P_0)/2^4 + P_0 \cdot 257/2^{12}$
(1, 1, 1, 1)	$(1-P_0)/2^4 + P_0 \cdot 171/2^{12}$

For each frame j and for each vector $(b_0, \ldots, b_{d-1})^{\mathrm{T}}$ we then calculate

$$\Pr\{{}_3\mathcal{S}_{l_1,l_2} = (b_0,\ldots,b_{d-1})^{\mathrm{T}} \text{ in } j\text{th frame}\} = {}_3p^j_{l_1,l_2}(b_0,\ldots,b_{d-1})$$
$$= \sum_{t\in\{101\ldots164\}} \Pr\{(l_1,l_2) \text{ at time } t\}\cdot\Pr\{\mathcal{E}_d = {}_3\hat{\mathcal{Z}}^j_{l_1,l_2}(t)\oplus(b_0,\ldots,b_{d-1})^{\mathrm{T}}\}$$
$$+\frac{1}{2}\left(1-\sum_{t\in\{101\ldots164\}}\Pr\{(l_1,l_2) \text{ at time } t\}\right). \tag{5.22}$$

All the m frames give us a more precise estimation:

$$\Pr\{{}_3\mathcal{S}_{l_1,l_2} = (b_0,\ldots,b_{d-1})^{\mathrm{T}}\} = {}_3p_{l_1,l_2}(b_0,\ldots,b_{d-1})$$
$$= \prod_{j=1}^{m} {}_3p^j_{l_1,l_2}(b_0,\ldots,b_{d-1}) = 2^{\sum_{j=1}^{m}\log_2({}_3p^j_{l_1,l_2}(b_0,\ldots,b_{d-1}))}. \tag{5.23}$$

In this formula the last two values should both be divided by a factor equal to their sum over all possible values of $(b_0,\ldots,b_{d-1})$. This factor has been left out because we are really interested in the relative values of the probabilities for different values of $(b_0,\ldots,b_{d-1})$. To simplify numerical calculations, ${}_3p_{l_1,l_2}(b_0,\ldots,b_{d-1})$ can be normalised through division by any constant.

We have just found the way how to calculate the probability $\Pr\{{}_3\mathcal{S}_{l_1,l_2} = (b_0,\ldots,b_{d-1})^{\mathrm{T}}\}$, for every d-dimension value $(b_0,\ldots,b_{d-1})^{\mathrm{T}}$. In a similar fashion, based on the equation (5.17), we can derive the d-dimension vectors ${}_1\hat{\mathcal{Z}}^j_{l_2,l_3}(t)$ and ${}_2\hat{\mathcal{Z}}^j_{l_1,l_3}(t)$, and then define the vectors ${}_1\mathcal{S}_{l_2,l_3}$ and ${}_2\mathcal{S}_{l_1,l_3}$. The formulas to calculate $\Pr\{{}_1\mathcal{S}_{l_2,l_3} = (b_0,\ldots,b_{d-1})^{\mathrm{T}}\}$ and $\Pr\{{}_2\mathcal{S}_{l_1,l_3} = (b_0,\ldots,b_{d-1})^{\mathrm{T}}\}$ are similar to equations (5.22) and (5.23).

Finally, we have a set of h tables like $\Pr\{{}_r\mathcal{S}_{l_i,l_j} = (b_0,\ldots,b_{d-1})\}$. If we "guess" the key $\hat{K}$, then in each such distribution table one row (probability) can be selected, corresponding to $\hat{K}$. *The measure of likelihood acceptance of $\hat{K}$ is the product of the selected probabilities through all the h tables.*

Our task is then to select a set of "guessed" keys $\hat{K}$ with maximum probabilities, and then perform a test whether the real key K can be one of the selected. More details depend on the exact structure of simulations, which we discuss in the next section.

5.3.2 Creating Candidate Tables of $s(l)$-Sequences

In the previous subsection we have found how to create a distribution table for d-dimension random variables ${}_r\mathcal{S}_{l_i,l_j}$. If we have h such distributions, then a "guessed" key $\hat{K}$ is measured by its probability, as described above.

We are now faced with the problem of how to select the most likely $\hat{K}$'s in an efficient way. For this purpose we partly use the idea that was introduced in the Ekdahl-Johansson attack, but in a modified way. In this section we show the technical details of searching for the best $\hat{K}$'s, and focus on computation aspects.

The idea is that first we choose some interval $\mathcal{I}_1 = [I_{1,a} \dots I_{1,b}]$ and then we construct h_1 distribution tables for ${}_3\mathcal{S}_{l_1,l_2}$, where $l_1, l_2 \in \mathcal{I}_1$. I.e., the number of distribution tables will be $h_1 = (I_{1,b} - I_{1,a} + 1)^2$, and the number of $s_1(l)$'s and $s_2(l)$'s that are involved in the linear expressions for ${}_3\mathcal{S}_{l_1,l_2}$ is $2 \cdot (I_{1,b} - I_{1,a} + 1 + d)$, see formula (5.19).

Let us consider some choice of values for $s_1(I_{1,a}), \dots, s_1(I_{1,b}+d), s_2(I_{1,a}), \dots, s_2(I_{1,b}+d)$ to be a pair of vectors $(\mathcal{S}_{1,\mathcal{I}_1}, \mathcal{S}_{2,\mathcal{I}_1})$ (note, the vector of interest ends with $I_{1,b}+d$, rather then $I_{1,b}+d-1$; the reason can be seen from (5.19), where $l_1, l_2 \in \mathcal{I}_1$), i.e.,

$$(s_1(I_{1,a}), \dots, s_1(I_{1,b}+d), s_2(I_{1,a}), \dots, s_2(I_{1,b}+d)) \stackrel{p}{=} (\mathcal{S}_{1,\mathcal{I}_1}, \mathcal{S}_{2,\mathcal{I}_1}). \quad (5.24)$$

The measure of the choice is the probability mass defined as

$$\prod_{l_1,l_2 \in \mathcal{I}_1} \Pr\{{}_3\mathcal{S}_{l_1,l_2} | (\mathcal{S}_{1,\mathcal{I}_1}, \mathcal{S}_{2,\mathcal{I}_1})\}. \quad (5.25)$$

Now, by exhaustive search the most likely r pairs $(\mathcal{S}_{1,\mathcal{I}_1}, \mathcal{S}_{2,\mathcal{I}_1})$ form a set ${}_3\mathfrak{T}_{\mathcal{I}_1} = \{(\mathcal{S}_{1,\mathcal{I}_1}, \mathcal{S}_{2,\mathcal{I}_1})\}$. The size of the exhaustive search is $2^{2\cdot(I_{1,b}-I_{1,a}+1+d)}$. In a similar way we can perform the same exhaustive search to create the sets ${}_1\mathfrak{T}_{\mathcal{I}_1} = \{(\mathcal{S}_{2,\mathcal{I}_1}, \mathcal{S}_{3,\mathcal{I}_1})\}$ and ${}_2\mathfrak{T}_{\mathcal{I}_1} = \{(\mathcal{S}_{1,\mathcal{I}_1}, \mathcal{S}_{3,\mathcal{I}_1})\}$, each containing the r most likely candidates.

To understand better how the exhaustive search for ${}_3\mathfrak{T}_{\mathcal{I}_1}$ is done, one can think of the matrix multiplication:

$$\left(\begin{array}{c|c}
\begin{array}{cccc} 110\dots00 & \dots & 000\dots00 \\ 011\quad 00 & & 000\quad 00 \\ \vdots\quad \ddots & & \vdots\quad \vdots \\ 000\dots11 & \dots & 000\dots00 \end{array} & \begin{array}{cccc} 110\,0\dots000\dots000\dots00 \\ 011\,0\quad 000\quad 000\quad 00 \\ \vdots\quad \ddots\quad \vdots\quad \vdots \\ 000\,0\dots110\dots000\dots00 \end{array} \\ \hline
\begin{array}{cccc} 110\dots00 & \dots & 000\dots00 \\ 011\quad 00 & & 000\quad 00 \\ \vdots\quad \ddots & & \vdots\quad \vdots \\ 000\dots11 & \dots & 000\dots00 \end{array} & \begin{array}{cccc} 011\,0\dots000\dots000\dots00 \\ 001\,1\quad 000\quad 000\quad 00 \\ \vdots\quad \ddots\quad \vdots\quad \vdots \\ 000\,0\dots011\dots000\dots00 \end{array} \\ \hline
\vdots & \vdots \\ \hline
\begin{array}{cccc} 000\dots00 & \dots & 110\dots00 \\ 000\quad 00 & & 011\quad 00 \\ \vdots\quad \vdots & & \vdots\quad \ddots \\ 000\dots00 & \dots & 000\dots11 \end{array} & \begin{array}{cccc} 000\,0\dots000\dots110\dots00 \\ 000\,0\quad 000\quad 011\quad 00 \\ \vdots\quad \vdots\quad \vdots\quad \ddots \\ 000\,0\dots000\dots000\dots11 \end{array}
\end{array}\right) \cdot \begin{pmatrix} s_1(I_a) \\ \vdots \\ s_1(I_b+d) \\ \hline s_2(I_a) \\ \vdots \\ s_2(I_b+d) \end{pmatrix} = \begin{pmatrix} {}_3\mathcal{Z}_{I_a,I_a} \\ \hline {}_3\mathcal{Z}_{I_a,I_a+1} \\ \hline \vdots \\ \hline {}_3\mathcal{Z}_{I_b,I_b} \end{pmatrix},$$

where for every "guessed" vector $(\mathcal{S}_{1,\mathcal{I}_1}, \mathcal{S}_{2,\mathcal{I}_1})$ (exhaustive search) the set of vectors ${}_3\mathcal{S}_{l_1,l_2} \stackrel{p}{=} {}_3\mathcal{Z}_{l_1 b,l_2}$ is determined uniquely by the matrix multiplication. We can then calculate the value of our choice by formula (5.25). After that, the most likely r pairs are selected and stored in the list (or table) ${}_3\mathfrak{T}_{\mathcal{I}_1}$.

Recall that to recover the key K uniquely, we need to have 64 bits: 19 bits of $s_1(l)$'s, 22 bits of $s_2(l)$'s, and 23 bits of $s_3(l)$'s. It means that for $d = 4$ it might be enough to have only one interval $\mathcal{I}_1$ of size 19. When we try to reduce the number of frames m needed for the attack, then there are two reasons for why this simple scenario is not working:

a) to create one likelihood table ${}_3\mathfrak{T}_{\mathcal{I}_1}$ the exhaustive search will be of size $2^{2\cdot(19+4)} = 2^{46}$ – this is practically impossible.

b) when the number of frames m is reduced, then the number of candidates r must be increased significantly, so that the correct pairs are present in the tables ${}_1\mathfrak{T}_{\mathcal{I}_1}, {}_2\mathfrak{T}_{\mathcal{I}_1}$, and ${}_3\mathfrak{T}_{\mathcal{I}_1}$. Otherwise, the joint intersection of these sets will not give us the correct triple $(\mathcal{S}_{1,\mathcal{I}_1}, \mathcal{S}_{2,\mathcal{I}_1}, \mathcal{S}_{3,\mathcal{I}_1})$.

To overcome these problems, we could take $\mathcal{I}_1$ of a short size, and introduce one more interval, $\mathcal{I}_2 = [I_{2,a} \dots I_{2,b}]$, and then we construct two kinds of tables ${}_*\mathfrak{T}_{\mathcal{I}_1}$ and ${}_*\mathfrak{T}_{\mathcal{I}_2}$ each of size r. We need to take $\mathcal{I}_2$ such that it intersects $\mathcal{I}_1$, otherwise the intersection would be r^2, and, hence, r cannot be large. Now in a similar way we can create the sets ${}_1\mathfrak{T}_{\mathcal{I}_2} = \{(\mathcal{S}_{2,\mathcal{I}_2}, \mathcal{S}_{3,\mathcal{I}_2})\}$, ${}_2\mathfrak{T}_{\mathcal{I}_2} = \{(\mathcal{S}_{1,\mathcal{I}_2}, \mathcal{S}_{3,\mathcal{I}_2})\}$, and ${}_3\mathfrak{T}_{\mathcal{I}_2} = \{(\mathcal{S}_{1,\mathcal{I}_2}, \mathcal{S}_{2,\mathcal{I}_2})\}$, each containing the r most likely pairs, the measure of which is calculated similar to the formula (5.25). Due to the intersection

$$\mathcal{S}_{i,\mathcal{I}_1} \times \mathcal{S}_{i,\mathcal{I}_2} = \begin{cases} \mathcal{S}_{i,\mathcal{I}_1 \cup \mathcal{I}_2}, & \text{if the end of } \mathcal{S}_{i,\mathcal{I}_1} \text{ corresponds to the} \\ & \text{beginning of } \mathcal{S}_{i,\mathcal{I}_2} \\ \emptyset, & \text{otherwise} \end{cases} \tag{5.26}$$

the intersection of these two sets is

$${}_3\mathfrak{T}_{\mathcal{I}_1 \cup \mathcal{I}_2} = {}_3\mathfrak{T}_{\mathcal{I}_1} \cap {}_3\mathfrak{T}_{\mathcal{I}_2} = \left\{ (\mathcal{S}_{1,\mathcal{I}_1 \cup \mathcal{I}_2}, \mathcal{S}_{2,\mathcal{I}_1 \cup \mathcal{I}_2}) : \begin{cases} (\mathcal{S}_{1,\mathcal{I}_1}, \mathcal{S}_{2,\mathcal{I}_1}) \in {}_3\mathfrak{T}_{\mathcal{I}_1} \\ (\mathcal{S}_{1,\mathcal{I}_2}, \mathcal{S}_{2,\mathcal{I}_2}) \in {}_3\mathfrak{T}_{\mathcal{I}_2} \\ \mathcal{S}_{1,\mathcal{I}_1} \times \mathcal{S}_{1,\mathcal{I}_2} \neq \emptyset \\ \mathcal{S}_{2,\mathcal{I}_1} \times \mathcal{S}_{2,\mathcal{I}_2} \neq \emptyset \end{cases} \right\}. \tag{5.27}$$

The larger the intersection of the intervals $\mathcal{I}_1$ and $\mathcal{I}_2$, the smaller the intersection set, i.e., $|{}_3\mathfrak{T}_{\mathcal{I}_1 \cup \mathcal{I}_2}| \ll |{}_3\mathfrak{T}_{\mathcal{I}_1}| \cdot |{}_3\mathfrak{T}_{\mathcal{I}_2}| = r^2$. Let us call this type of intersections as *horizontal* intersection. Similar horizontal intersections are ${}_1\mathfrak{T}_{\mathcal{I}_1 \cup \mathcal{I}_2}$ and ${}_2\mathfrak{T}_{\mathcal{I}_1 \cup \mathcal{I}_2}$.

By *vertical* intersection we call the intersections of the form:

$$ {}_{1,2}\mathfrak{T}_{\mathcal{I}_i} = {}_1\mathfrak{T}_{\mathcal{I}_i} \cap {}_2\mathfrak{T}_{\mathcal{I}_i} = \left\{ (\mathcal{S}_{1,\mathcal{I}_i}, \mathcal{S}_{2,\mathcal{I}_i}, \mathcal{S}_{3,\mathcal{I}_i}) : \begin{cases} (\mathcal{S}_{2,\mathcal{I}_i}, \mathcal{S}_{3,\mathcal{I}_i}) \in {}_1\mathfrak{T}_{\mathcal{I}_i} \\ (\mathcal{S}_{1,\mathcal{I}_i}, \mathcal{S}_{3,\mathcal{I}_i}) \in {}_2\mathfrak{T}_{\mathcal{I}_i} \end{cases} \right\}, \tag{5.28} $$

and ${}_{2,3}\mathfrak{T}_{\mathcal{I}_i}$, ${}_{1,3}\mathfrak{T}_{\mathcal{I}_i}$ are defined in a similar way. One more *triple vertical* intersection is defined as

$$ {}_{1,2,3}\mathfrak{T}_{\mathcal{I}_i} = {}_1\mathfrak{T}_{\mathcal{I}_i} \cap {}_2\mathfrak{T}_{\mathcal{I}_i} \cap {}_3\mathfrak{T}_{\mathcal{I}_i} = \left\{ (\mathcal{S}_{1,\mathcal{I}_i}, \mathcal{S}_{2,\mathcal{I}_i}, \mathcal{S}_{3,\mathcal{I}_i}) : \begin{cases} (\mathcal{S}_{2,\mathcal{I}_i}, \mathcal{S}_{3,\mathcal{I}_i}) \in {}_1\mathfrak{T}_{\mathcal{I}_i} \\ (\mathcal{S}_{1,\mathcal{I}_i}, \mathcal{S}_{3,\mathcal{I}_i}) \in {}_2\mathfrak{T}_{\mathcal{I}_i} \\ (\mathcal{S}_{1,\mathcal{I}_i}, \mathcal{S}_{2,\mathcal{I}_i}) \in {}_3\mathfrak{T}_{\mathcal{I}_i} \end{cases} \right\}. \tag{5.29} $$

5.3.3 Design of Intervals

Let us take one interval $\mathcal{I}'_1 = [87 \dots 97]$. Two extreme situations are when $(l_1, l_2) = (87, 87)$ and $(l_1, l_2) = (97, 97)$. In each frame j there are only 228 bits are accessible $v^j(101), \dots, v^j(100 + 228)$. In Figure 5.2 we see that the probability $\Pr\{(l_1, l_2) \text{ at time } t\}$ for this interval gets its maximum value on around $t \approx (116 \dots 129)$. Hence, the bits around $v(116) \dots v(129)$ give us the most information about the d-dimension vectors, when $l_1, l_2 \in \mathcal{I}_1$. We can also say that for this interval the informative bits are around $v(105) \dots v(145)$, because for any other v's the probability is almost 0.

Let us now consider three more intervals

$$ \begin{aligned} \mathcal{I}'_2 &= [63 \dots 73], \\ \mathcal{I}'_3 &= [165 \dots 175], \\ \mathcal{I}'_4 &= [231 \dots 241]. \end{aligned} \tag{5.30} $$

In Figure 5.3 the bounded densities for each interval are shown. The interval $\mathcal{I}'_2$ is moved to the left below $t < 101$, where the valuable v's are inaccessible for us. It means that this choice is not appropriate. On the other hand, the interval $\mathcal{I}'_4$ is moved to the right and very close to the right border of accessible v's. This interval can be considered as the last appropriate interval. Also note that as the interval is moved to the right the amplitude decreases, i.e., the error probability of the random variables estimation is higher.

In our simulations we decided to choose the size of each interval to be 11. Independently of the parameter $d \geq 1$ in each table ${}_3\mathfrak{T}_{\mathcal{I}_i}$ we store only the pairs $(\mathcal{S}_{1,\mathcal{I}_i}, \mathcal{S}_{2,\mathcal{I}_i})$ of vectors each of size 12 bits only. Schematically, the structure of intervals is depicted below in Figure 5.4.

Two neighbour intervals intersect in 6 positions, whereas the last $d - 1$ positions are assumed to be badly estimated (tail bits). I.e., any horizontal

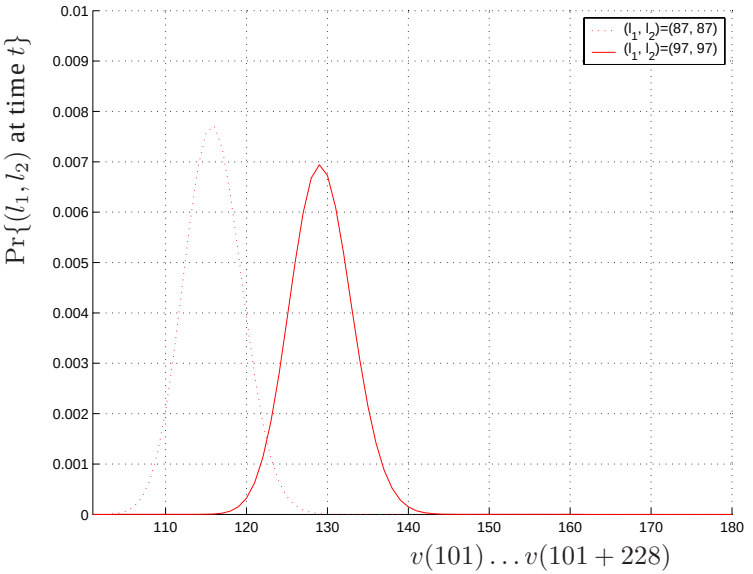


Figure 5.2: The density of $\Pr\{(l_1, l_2) \text{ at time } t\}$ when $(l_1, l_2) = (87, 87)$ and $(l_1, l_2) = (97, 97)$.

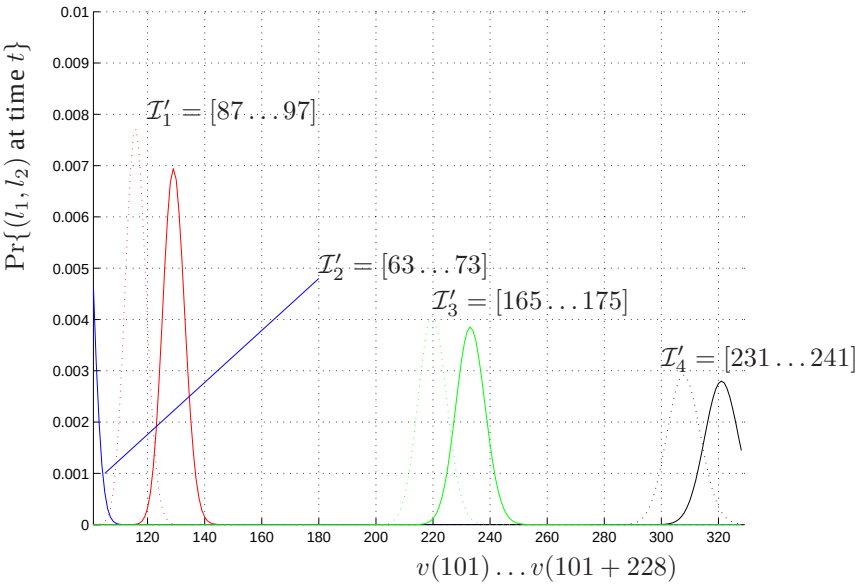


Figure 5.3: The bounded densities for $\mathcal{I}'_1 = [87 \ldots 97]$, $\mathcal{I}'_2 = [63 \ldots 73]$, $\mathcal{I}'_3 = [165 \ldots 175]$, and $\mathcal{I}'_4 = [231 \ldots 241]$.

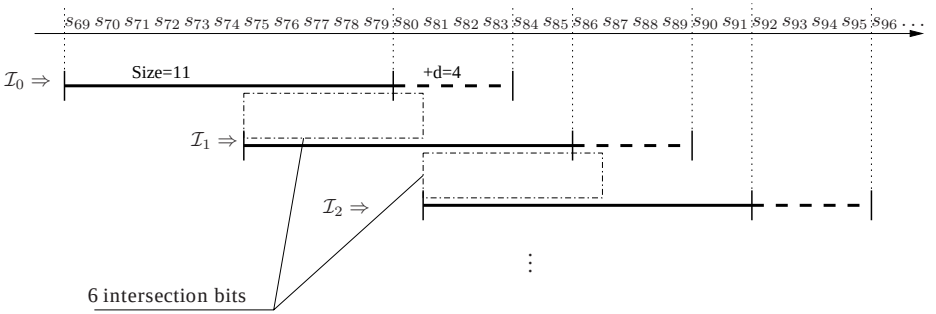


Figure 5.4: The structure of intervals used in simulations.

intersection of two tables ${}_3\mathfrak{T}_{\mathcal{I}_i}$ and ${}_3\mathfrak{T}_{\mathcal{I}_{i+1}}$ will be done by 12 bits (6 bits are $s_1(\mathcal{I}_{i+1})$, ..., $s_1(\mathcal{I}_{i+1}+5)$, and similar 6 bits are $s_2(\mathcal{I}_{i+1})$, ..., $s_2(\mathcal{I}_{i+1}+5)$). Also note that any vertical intersection will be done in 12 bits also. The choice of this structure of the intervals allowed us to introduce several efficient strategies to intersect the tables.

Since the size of each interval is 11, it means that the number of distribution tables of ${}_*\mathcal{S}_{l_i,l_j}$-random variables is $11^2 = 121$. When $d = 4$, the number of variables involved in ${}_*\mathcal{S}_{l_i,l_j}$'s is $2\cdot(11+4) = 30$. Hence, to create one ${}_*\mathfrak{T}_{\mathcal{I}_i}$-set of the r most likelihood pairs, we need to perform an exhaustive search of size 2^{30}. The number of such sets ${}_*\mathfrak{T}_{\mathcal{I}_i}$ is 9 (3 intervals times 3 cases for '*').

In our simulations we have considered 28 intervals:

$$\begin{cases} \mathcal{I}_0 = [69 \dots 79] \\ \mathcal{I}_k = 6 \cdot k + \mathcal{I}_0 \quad \text{for } k = 1, 2, \dots, 27. \end{cases} \tag{5.31}$$

So, the last interval is $\mathcal{I}_{27} = [231 \dots 241]$ (see also Figure 5.3). When for a chosen interval $\mathcal{I}_i$ we estimate the probability $\Pr\{{}_3\mathcal{S}_{l_1,l_2} = (b_0, \dots, b_{d-1})^{\mathrm{T}}\}$ with the formula (5.22), then we only need to look through the bits v^j that are valuable for $\mathcal{I}_i$. Let us set the "window" of valuable bits to be of size 64, then, for example, for the interval $\mathcal{I}_1$ on the Figure 5.3 the "window" is $t_1 = [101 \dots 164]$, for $\mathcal{I}_3 \Rightarrow t_3 = [203 \dots 266]$, and for $\mathcal{I}_4 \Rightarrow t_4 = [266 \dots 329]$. Actually, the "window" can be larger, but 64 bits completely cover the most valuable v's for any interval $\mathcal{I}_i$.

The likelihood sets ${}_*\mathfrak{T}_{\mathcal{I}_i}$, each containing r pairs, can be presented in Table 5.2.

The time complexity to form these data is

$$O(3 \cdot 28 \cdot (11^2 \cdot 2^d \cdot m \cdot 64 + 2^{22+2d})). \tag{5.32}$$

	$\mathcal{I}_0$	$\mathcal{I}_1$	...	$\mathcal{I}_{27}$
Case 1	${}_1\mathfrak{T}_{\mathcal{I}_0} = (\mathcal{S}_{2,\mathcal{I}_0}, \mathcal{S}_{3,\mathcal{I}_0})$	${}_1\mathfrak{T}_{\mathcal{I}_1} = (\mathcal{S}_{2,\mathcal{I}_1}, \mathcal{S}_{3,\mathcal{I}_1})$		${}_1\mathfrak{T}_{\mathcal{I}_{27}} = (\mathcal{S}_{2,\mathcal{I}_{27}}, \mathcal{S}_{3,\mathcal{I}_{27}})$
Case 2	${}_2\mathfrak{T}_{\mathcal{I}_0} = (\mathcal{S}_{1,\mathcal{I}_0}, \mathcal{S}_{2,\mathcal{I}_0})$	${}_2\mathfrak{T}_{\mathcal{I}_1} = (\mathcal{S}_{1,\mathcal{I}_1}, \mathcal{S}_{2,\mathcal{I}_1})$		${}_2\mathfrak{T}_{\mathcal{I}_{27}} = (\mathcal{S}_{1,\mathcal{I}_{27}}, \mathcal{S}_{2,\mathcal{I}_{27}})$
Case 3	${}_3\mathfrak{T}_{\mathcal{I}_0} = (\mathcal{S}_{1,\mathcal{I}_0}, \mathcal{S}_{2,\mathcal{I}_0})$	${}_3\mathfrak{T}_{\mathcal{I}_1} = (\mathcal{S}_{1,\mathcal{I}_1}, \mathcal{S}_{2,\mathcal{I}_1})$		${}_3\mathfrak{T}_{\mathcal{I}_{27}} = (\mathcal{S}_{1,\mathcal{I}_{27}}, \mathcal{S}_{2,\mathcal{I}_{27}})$

Table 5.2: Tabular representation of likelihood sets ${}_*\mathfrak{T}_{\mathcal{I}_i}$ for 28 intervals and 3 cases.

This is because there are 84 sets ${}_*\mathfrak{T}_i$; to create each set requires 11^2 distribution tables of size 2^d; to calculate each value in the table requires $m \cdot 64$ operations; and the exhaustive search complexity for each set is 2^{22+2d}.

5.3.4 Strategies for Intersection of the Tables ${}_*\mathfrak{T}_{\mathcal{I}_i}$

When the first part of the attack is done, the second part is just intersection of the sets until we get the set of triples ${}_{1,2,3}\mathfrak{T}_*$ of appropriate size. Here are several strategies that we can follow to achieve our goal:

I. *Intersection of 9 tables, large r.* Try all triples of intervals $(\mathcal{I}_k, \mathcal{I}_{k+1}, \mathcal{I}_{k+2})$, for $k = 0, 1, \ldots, 25$. The intersection of 9 tables gives us the table ${}_{1,2,3}\mathfrak{T}_{\mathcal{I}_k \cup \mathcal{I}_{k+1} \cup \mathcal{I}_{k+2}}$ of triples $(\mathcal{S}_{1,\mathcal{I}_k \cup \mathcal{I}_{k+1} \cup \mathcal{I}_{k+2}}, \mathcal{S}_{2,\mathcal{I}_k \cup \mathcal{I}_{k+1} \cup \mathcal{I}_{k+2}}, \mathcal{S}_{3,\mathcal{I}_k \cup \mathcal{I}_{k+1} \cup \mathcal{I}_{k+2}})$. Each $\mathcal{S}$ contains 24 bits, but we need only 19, 22, and 23 bits for LFSR-1, LFSR-2, and LFSR-3, respectively. We can do first vertical intersections and get ${}_{1,2,3}\mathfrak{T}_{\mathcal{I}_i}$, and then perform horizontal intersection. Since any of the intersections is done by 12 bits, the number of the most likely pairs in ${}_*\mathfrak{T}_{\mathcal{I}_i}$ can be quite large. For this strategy we can safely use $r \approx 50000$.

II. *Intersection of 6 tables, medium r.* The same as Strategy I, but for each interval one table is discarded. We just assume that the discarded tables do not contain the correct pairs. Then perform the intersection of the remaining 6 tables. The number of assumptions is 3^3. The parameter r is about $r \approx 30000$.

III. *Intersection of 4 tables, small r.* Try all pairs of intervals $(\mathcal{I}_k, \mathcal{I}_{k+2})$, for all $k = 0, 1, \ldots, 25$. We assume also that one of the tables ${}_*\mathfrak{T}_{\mathcal{I}_k}$ and one of ${}_*\mathfrak{T}_{\mathcal{I}_{k+2}}$ do not contain the correct pair. The number of assumptions is 3^2. For the remaining 4 tables we perform the intersection. Note, there is no horizontal intersection, but only 2 vertical intersections, one for

$\mathcal{I}_k$ and one for $\mathcal{I}_{k+2}$. Due to this the critical value for the parameter r is about $r \approx 10000$. The appropriate choice of the intersection scheme made this strategy work.

IV. *Intersection of 4 tables, small r, version 2.* The same as Strategy III, but the pairs of intervals $(\mathcal{I}_{k_1}, \mathcal{I}_{k_2})$ can be so that $k_1 = 0, 1, \ldots, 25$, and $k_2 = k_1 + 2, \ldots, 28$. Unfortunately, it can happen that *some* outputs from LFSR's in the second interval $\mathcal{I}_{k_2}$ will be a linear combination of $s(l)$'s from $\mathcal{I}_{k_1}$. For LFSR-1, the size of which is 19, it is not very critical because we achieve 24 bits of information. It means that even if 5 bits will depend on others, we still have a full rank in translation from $s(l)$'s to 19 bits of the key K. It is more critical for LFSR-3, which is of length 23. Anyway, if the system will not be of full rank, then some bits we can just guess. That makes this strategy work in general (implementation is then more complicated).

V. *Heuristic procedure, r is dynamic.* Can be introduced in the following way:
If in some step for some intersection $\mathfrak{T}' \cap \mathfrak{T}''$ we get $\emptyset$, or a very small set, then increase the value r for $\mathfrak{T}'$ and $\mathfrak{T}''$ selectively, until their intersection give us a set of size at least r_0, for some threshold value. Thus, we can start creating the sets ${}_*\mathfrak{T}_{\mathcal{I}_i}$ with a small value of r, and then increase it selectively, when necessary.

So, here is a wide choice to choose a strategy. In our simulations we have tried several of them.

5.4 Simulation Results

The attack can basically be divided into three steps,

1) Statistical analysis of m frames,

2) Decoding process and generating the tables ${}_*\mathfrak{T}_{\mathcal{I}_i}$,

3) Intersection of the tables and check estimated keys $\hat{K}$.

For the first two steps we present the actual time. The attack was implemented on Pentium-4, CPU 2.4GHz, 256Mb RAM, OS Windows XP Pro SP1.

1st step / 2nd step	m=2000	m=5000	m=10000
d=1	11 sec / 18 sec	26 sec / 18 sec	58 sec / 18 sec
d=2	14 sec / 8 min	32 sec / 8 min	72 sec / 8 min
d=4	40 sec / 7 hrs	94 sec / 7 hrs	190 sec / 7 hrs

The measure of "goodness" of the attack can be expressed in terms of the number of frames m needed and its success rate. The attack was successfully implemented on a usual PC-computer, and it performs the attack from several seconds to several minutes, depending on the choice of strategy, and parameters m, d, and r.

Success rate of the attack depends on the choice of the design parameters d and r, and the strategy that is used. For some values of m and d here we present in Figures 5.5-5.8 the plots for the probabilities:

$$\Pr\{ \text{the correct vector } \textit{is in } {}_*\mathfrak{T}_{\mathcal{I}_i}, \text{ for given parameter } r\}.$$

When the tables are constructed, in the intersection process it is very important that the correct pair is present in the corresponding table. Otherwise, the intersection will never give us the correct key.

In Figures 5.5-5.8 we show the real estimated success rates for different strategies, with different number of frames m and the attack design parameter d. In Figure 5.5 consider the curve corresponding to $d = 1$ and to Strategy I, when $m = 10000$ frames. For $r = 15000$ we have the success rate of the attack around 58%, whereas for Strategies II-IV the success rate is almost 100%.

From the plots below the Strategy IV looks the most attractive. In this strategy we need to intersect only 4 tables, but the disadvantage is that there is no horizontal intersection. And then after two vertical intersections we need to try all possible combinations of elements in two tables. One more disadvantage is that we could get some equation dependencies between two intervals, so then the actual time complexity will grow. On the contrary, strategy III looks the next the most attractive, and there are no problems with intervals. Since there are no horizontal intersections in these strategies, this forces us to reduce the parameter r significantly. The critical value of this parameter is $r_{cr} = 10000$, and the optimal is $r_{opt} = 2000$ from the computational and memory points of view. Strategy II avoids such problems mostly because of the presence of vertical intersections, which are intersecting on 12 bits.

A practical solution to overcome the time-memory problems related to intersections of the tables can be the use of the Heuristic Strategy V, combined with one of the previous strategies. The idea of heuristic is to control the size of the intersection. If the size is likely to be increased by some threshold criteria, then try to increase the initial parameter r until the limit is reached, or solution is found. Heuristic can also control the size of the tables independently, and this will give the best performance of the attack.

Dramatic advantage of use the proper design parameter d is seen in Figure 5.8. To make the advantage clearer that plot shows how much we gain

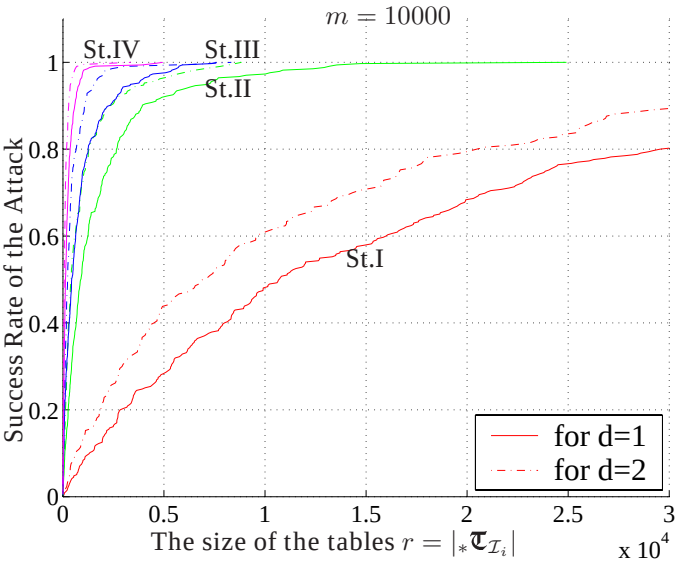


Figure 5.5: Strategies comparison for $m = 10000$.

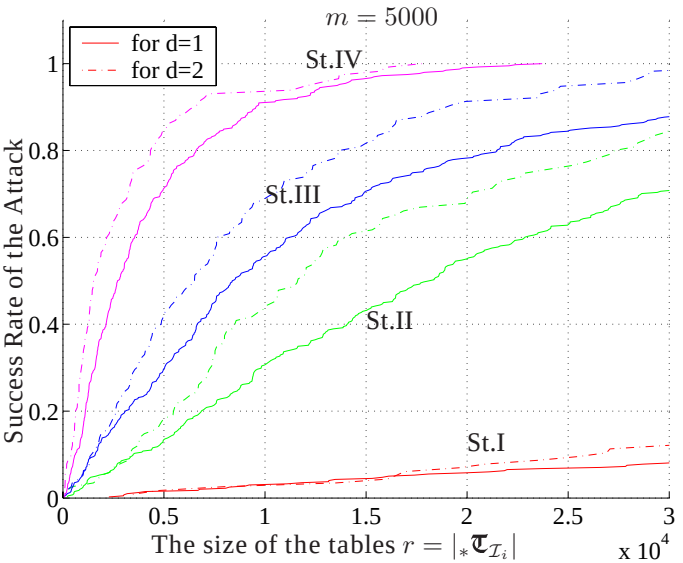


Figure 5.6: Strategies comparison for $m = 5000$.

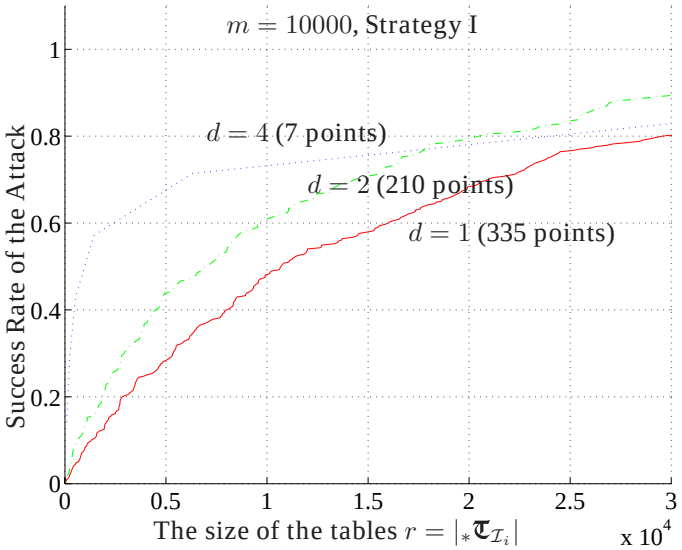


Figure 5.7: Strategies comparison for $m = 2000$.

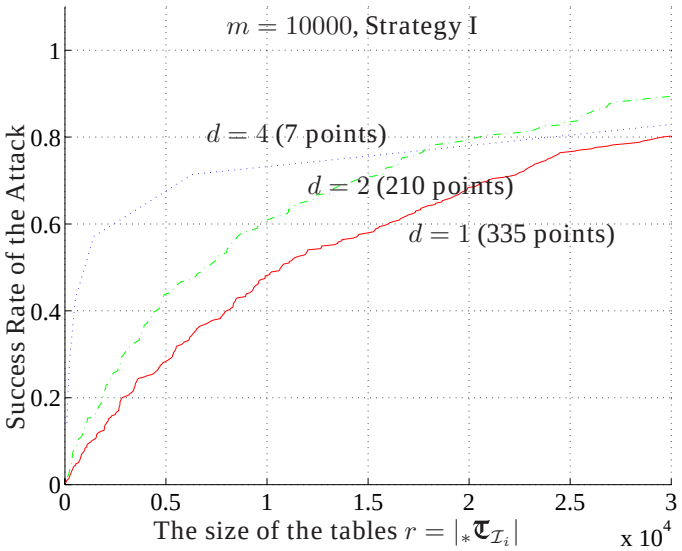


Figure 5.8: The effect of d on the success rate on the example when $m = 10000$ and Strategy I is applied.

when d is 1, 2, and 4. When $r = 15000$, the change of the parameter d from $d = 1$ to $d = 2$ significantly increases the success rate from 58% to 70%. These simulations were done for $m = 10000$ frames, and with the application of Strategy I.

Finally, we show the advantage of our attack in comparison with the previous Ekdahl-Johansson attack in the following two tables:

Success Rate / (Time of the Attack)	**Ekdahl-Johansson Attack (2002)** **Number of Frames/(time of GSM conversation in min/sec)**		
Configuration	**30000** (2m30s)	**50000** (3m45s)	**70000** (5m20s)
3 Intervals of size 7	0.02 / (1min)	0.13 / (2min)	0.49 / (3min)
3 Intervals of size 8	0.02 / (2min)	0.20 / (3min)	0.57 / (4min)
2 Intervals of size 9	0.03 / (3min)	0.33 / (4min)	0.76 / (5min)

Success Rate / (Time of the Attack)	**Our Proposed Attack** **Number of Frames/(time of GSM conversation in min/sec)**		
Configuration	**2000** (9sec)	**5000** (43sec)	**10000** (46sec)
St.I, d=2, r=10K	0.01 / (8min)	0.05 / (8min)	0.60 / (8min)
St.II, d=1, r=5K	0.01 / (29sec)	0.15 / (44sec)	0.93 / (76sec)
St.III, d=2, r=5K	0.02 / (8min)	0.40 / (8min)	0.99 / (8min)
St.IV, d=2, r=5K	0.05 / (10min)	0.85 / (10min)	0.9999(10min)

5.5 Summary

We have demonstrated two new ideas that provide improved performance for a correlation attack against A5/1. In simulation we get a high success rate for only 2000-5000 frames, using very little computation. But there is still deviation in performance depending on the strategies we choose, which means that there may very well be further improvements to come. Another further research topic would be to examine how small m can be made if we allow a substantial increase in attack complexity. If m can be decreased a bit further, ciphertext only attack may be practically possible.

Recently, an improvement of this attack was presented at SAC 2005 [BB05], which requires only 1500-2000 known frames for a key-recovering attack with success of 91%. Their attack is partly based on our cryptanalysis presented in this chapter, but the new idea is to use *conditional estimators* that gain a factor two in the correlation bias.

6

Cryptanalysis of VMPC and RC4A. Weakness of RC4-like Ciphers

"If I have a thousand ideas and only one turns out to be good, I am satisfied"

Alfred Bernhard Nobel

In 1987, Ron Rivest from RSA Data Security, Inc. made a design of a byte oriented stream cipher called RC4 [Sma03]. This cipher found its application in many Internet and security protocols. The design was kept secret up to 1994, when the alleged specification of RC4 was leaked for the first time [Sch96]. Since that time many cryptanalysis attempts have been done on RC4 [Gol97b, KMP+98, FM00, MS01, PP03].

At FSE 2004, a new stream cipher called VMPC [Zol04] was proposed by Bartosz Zoltak, which appeared to be a modification of the RC4 stream cipher. In cryptanalysis, a linear distinguishing attack is one of the most common attacks on stream ciphers. In the paper [Zol04] it was claimed that VMPC is designed especially to resist distinguishing attacks.

At the same conference, FSE 2004, another cipher, RC4A [PP04], was proposed by Souradyuti Paul and Bart Preneel. This cipher is another modification of RC4.

In this chapter we point out a general theoretical weakness of such ciphers, which, in some cases, can tell us without additional calculations

whether a new construction is weak against distinguishing attacks. We also investigate VMPC and RC4A in particular and find two linear distinguishing attacks on them. RC4A can be distinguished from random using around 2^{58} bytes of the keystream, whereas the attack on VMPC needs only $2^{39.97}$ bytes. These are the first existing attacks on VMPC and RC4A.

This chapter is organized as follows. In Section 6.2 we describe RC4, RC4A, and the VMPC ciphers. In Section 6.3 we study digraphs on an instance of VMPC, and then demonstrate a theoretical weakness of the RC4 family of stream ciphers in general. We propose our distinguishers for both VMPC and RC4A in Sections 6.4 and 6.5. Finally, we summarize the results and make conclusions in Section 6.6.

6.1 Introduction

6.1.1 Notation

The algorithms VMPC, RC4A and RC4 are byte oriented stream ciphers. For notation purposes *in this chapter* we consider VMPC-k, RC4A-k, and RC4-k to be k bit oriented ciphers, where the original designs are when $k = 8$. Therefore, in the design of these ciphers, $+$ means addition modulo 2^k. For simplicity in formulas, let q be the size of the permuters used in these ciphers, i.e.,

$$q = 2^k. \tag{6.1}$$

The ciphers have an internal state consisting of one or two permuters of length q, and a few iterators. The idea of these designs is derived from the RC4 stream cipher. Therefore, we call ciphers with a structure similar to RC4 as *the RC4 family of stream ciphers*. We denote by z_t a k bit output symbol at time t. When a permuter $R[\cdot]$ is applied r times, e.g., $R[R[\ldots R[x]\ldots]]$, then, for simplicity, we sometimes denote it as $R^r[x]$.

We will present *linear distinguishing attacks*, and use formulas and ideas described in Section 3.4. Let us denote the *random distribution* as P_{Random}, and the *cipher distribution* as P_{Cipher}. The number of samples required is n, and the *type* is denoted as P_{Type}. We denote the bias as $\epsilon = |P_{\text{Cipher}} - P_{\text{Random}}|$.

We use both formulas, (3.29) and (3.31), to estimate the required number of samples n.

6.1.2 Cryptanalysis Assumptions

We start our analysis of the RC4 family of stream ciphers by making the following reasonable assumptions.

(1) We assume that the initialisation procedure is perfect, i.e., all internal variables (except known iterators) are from the uniform distribution. In practice this is not true, but we make this assumption as long as we do not investigate the initialisation procedures.

(2) In our distinguishers we construct a type P_{Type} by collecting samples from the given keystream. Each derived sample at time t is from some *local distribution* of the keystream. We assume that at any time the internal state of a cipher is uniformly distributed and we don't have any knowledge about it. This assumption will be used to investigate different local distributions in the next sections. In our simulations we verified that the internal state of VMPC is roughly uniformly distributed. But for RC4A the internal state is not uniformly distributed.

(3) We assume that adjacent samples are independent. In practice it is not true, because between two consecutive samples the internal states of a cipher are dependent. It means that samples might have a strong dependency, which may influence on the resulting type P_{Type}. To reduce these dependencies we suggest to skip a few samples before accepting one, then the consecutive adjacent samples will be much less dependent on each other.

6.2 Descriptions of VMPC-k, RC4-k, and RC4A-k

The stream cipher RC4-k

RC4 [Sma03] was designed by Ron Rivest in 1987. It produces an infinite pseudo-random sequence of k bit symbols, which is used as the keystream. Encryption is then performed by bitwise adding the keystream to the plaintext. The structure of RC4-k is shown in Figure 6.1.

The stream cipher VMPC-k

VMPC [Zol04] was proposed at FSE 2004 by Bartosz Zoltak. This cipher is also byte oriented ($k = 8$), and is a generalised version of RC4-k. The structure of VMPC-k is shown in Figure 6.2.

The stream cipher RC4A-k

RC4A [PP04] was proposed at FSE 2004 by Souradyuti Paul and Bart Preneel. This cipher is an attempt to hide the correlation between the internal states and the keystream. The authors suggested to introduce a second permuter in the design. The structure of RC4A-k is shown in Figure 6.3.

Internal variables:
i, j – integers $\in [0 \dots q-1]$
$R[0 \dots q-1]$ – a permuter of integers $0 \dots q-1$

The RC4-k cipher
1. $R[\cdot]$ – are initialised with the secret key
   $i = j = 0$

2. Loop until enough k bit symbols
   $i++$
   $j += R[b]$
   `swap` $(R[i], R[j])$
   `output` $\leftarrow R[R[i] + R[j]]$

Figure 6.1: The structure of the RC4-k cipher.

Internal variables:
i, j – integers $\in [0 \dots q-1]$
$R[0 \dots q-1]$ – a permuter of integers $0 \dots q-1$

The VMPC-k cipher
1. $j, R[\cdot]$ – are initialised with the secret key
   $i = 0$

2. Loop until enough k bit symbols
   $j = R[j + R[i]]$
   `output` $\leftarrow R[R[R[j]] + 1]$
   `swap` $(R[i], R[j])$
   $i++$

Figure 6.2: The structure of the VMPC-k cipher.

6.3 Investigation of the RC4 Family of Stream Ciphers

In this section we approximate different *local distributions* of the accessible keystream in the RC4 family of stream ciphers, with the assumptions that were made in Section 6.1.2.

6.3.1 Digraphs Approach, on the Instance of VMPC-k

In this subsection we give the idea of how a distinguisher for VMPC can be built. In previous work [FM00], the cipher RC4-k was analysed. The authors suggested to observe two consecutive output symbols z_t, z_{t+1}, and the

Internal variables:
i, j_1, j_2 – integers $\in [0 \dots q-1]$
$R_1[0 \dots q-1], R_2[0 \dots q-1]$ – two permuters of integers $0 \dots q-1$

The RC4A-k cipher

1. $R_1[\cdot], R_2[\cdot]$ – are initialised with the secret key
   $i = j_1 = j_2 = 0$

2. Loop until enough k bit symbols
   $i++$
   $j_1 += R_1[i]$
   `swap` $(R_1[i],\ R_1[j_1])$
   `output` $\leftarrow R_2[R_1[i] + R_1[j_1]]$
   $j_2 += R_2[i]$
   `swap` $(R_2[i],\ R_2[j_2])$
   `output` $\leftarrow R_1[R_2[i] + R_2[j_2]]$

Figure 6.3: The structure of the RC4A-k cipher.

known variable i jointly. For RC4-5 they could calculate theoretical probabilities $\Pr\{(i, z_t = x, z_{t+1} = y)\}$, for all possible n^3 values of the triple (i, x, y) (let us denote such a distribution as $P_{(i,z_t,z_{t+1})}$). But for RC4-8 the authors could only approximate the bias for the distribution above due to high complexity of the calculations, and show that a distinguisher needs around $2^{30.6}$ samples (the required length of the plaintext to know).

We use a similar idea to create a distinguisher for VMPC-k. For this purpose we investigate the pair (z_t, z_{t+1}) in the following scheme, which is an extraction of steps in VMPC when generating two consecutive output symbols.

i – known value at time t
$j, R[\cdot]$ – are from a random source

1. $z_t = R[R^2[j] + 1]$
2. `swap` $(R[i], R[j])$
3. $j' = j + R[i+1]$
4. $z_{t+1} = R[R^3[j'] + 1]$

In Algorithm 1 we give an explicit algorithm to calculate the approximated distribution table $P_{(i,z_t,z_{t+1})}$. For each value i, in each cell of a table T

we want to store an integer number $T[i,x,y]$ of possible pairs $(i, R[\cdot])$, which cause the corresponding output pair $(z_t = x, z_{t+1} = y)$. It means, that the probability of any triple (i, z_t, z_{t+1}) can be calculated as:

$$\Pr\{(i, z_t = x, z_{t+1} = y)\} = \frac{T[i,x,y]}{q \cdot q!}. \tag{6.2}$$

Algorithm 1: *Recursive construction of the approximated distribution table* $P_{(i,z_t,z_{t+1})}$
Prepare the permuter: $R[i] = \infty$ at all positions, i.e., all cells of the permuter are undefined. In the algorithm the operation *define* $R[i]$ means that for the cell i in the permuter $R[\cdot]$ we need to try all possible values $0 \ldots (q-1)$. Note, we cannot select a value which has been already used in another cell of the permuter in a previous step. Before making a step back by the recursion, restore the value $R[i] = \infty$. In the case when the cell $R[i]$ was already defined (is not ∞) due to previous steps, then we just go to the next step directly.
Do the following steps recursively:

- for all $i = 0 \ldots q-1$.
- for all $j = 0 \ldots q-1$.
- define $R[j]$.
- define $R^2[j]$.
- define $R[R^2[j]+1] \Rightarrow$ remember $x = R[R^2[j]+1]$.
- define $R[i]$.
- swap$(R[i], R[j])$.
- define $R[i+1] \Rightarrow$ calculate $j' = j + R[i+1]$.
- define $R[j']$, then $R^2[j']$, then $R^3[j']$.
- define $R[R^3[j]+1] \Rightarrow$ remember $y = R[R^3[j]+1]$.
- $T[i,x,y] += (q-r)!$, where r is the actual number of currently defined cells in the permuter $R[\cdot]$.

The complexity of the algorithm is $O(2^{11k})$ [1]. In our simulations we

[1] The complexity to construct such a table with a similar algorithm for RC4-k is $O(2^{6k})$ [FM00].

could manage to calculate the approximation of $P_{(i,z_t,z_{t+1})}$ only for the reduced version VMPC-4. The bias of such a table appeared to be around $\epsilon \approx 2^{-8.7}$. It means that we can distinguish VMPC-4 from random having plaintext of length around 2^{18} 4-bit symbols. For notation purposes, let $P^{\mathrm{VMPC}-k}_{(i,z_t,z_{t+1})}$ be the distribution $P_{(i,z_t,z_{t+1})}$ for VMPC-k, and similar for $P^{\mathrm{RC4}-k}_{(i,z_t,z_{t+1})}$.

The calculation of a similar distribution table for VMPC-8 meets computational difficulties, as well as for RC4-8 in [FM00]. One of the ideas in [FM00] was to approximate the biases from small k's to a larger k, but we decided to avoid this way. Instead, in the next sections we will present only precise theoretical results on VMPC-8, and on the RC4 family of stream ciphers in general.

6.3.2 Theoretical Weakness of the RC4 Family of Stream Ciphers

The recursive Algorithm 1 is trivial and slow, but we use it to show further theoretical results. We prove that the approximated distribution table $P_{(i,z_t,z_{t+1})}$ cannot be the uniform distribution when k is larger than some threshold k_0. Moreover, we prove that *each* probability of the approximated distribution $P_{(i,z_t,z_{t+1})}$ differs from the corresponding probability in the case of a random source. In other words, the approximated distribution $P_{(i,z_t,z_{t+1})}$ is biased and we find the lower bound $\epsilon_{\min}$ for the bias.

Theorem 6.1: For VMPC-k, where $k \geq 8$, under the assumptions made in Section 6.1.2, the following holds.

1. Each probability

$$\Pr\{(i, z_t = x, z_{t+1} = y)\} \neq 1/q^3, \tag{6.3}$$

   although, in a random case it should be $1/q^3$.

2. The bias $\epsilon = |P_{\mathrm{Random}} - P^{\mathrm{VMPC}-k}_{(i,z_t,z_{t+1})}|$ is bounded by

$$\epsilon \geq \epsilon_{\min} = \frac{|\delta_{\min}| \cdot q \cdot (q-9)!}{q!} \geq q^{-8k}, \tag{6.4}$$

   where $|\delta_{\min}|$ is the minimum value, such that

$$(q-1)(q-2) \cdot \ldots \cdot (q-8) + \delta_{\min} \equiv 0 \pmod{q}.$$

3. For VMPC-8, we have $\epsilon_{\min} \approx 2^{-56.8}$.

Proof:
(1) Consider Algorithm 1. In the last step the value of r, i.e., the number of currently placed positions in the permuter, can be at most 9. It means that when the algorithm is finished, each cell in $P^{\mathrm{VMPC}-k}_{(i,z_t,z_{t+1})}$ can be written in the form $k \cdot (q-9)!$, for some integer k.

On the other hand, for a truly random sequence, the probability must be $\Pr\{(i, z_t, z_{t+1})\} = 1/q^3$. From (6.2) it follows that $\frac{k\cdot(q-9)!}{q\cdot q!}$ must be equal to $\frac{1}{q^3}$, i.e., k must be equal to

$$\frac{q \cdot (q-1) \cdot \ldots \cdot (q-8)}{q^2}. \tag{6.5}$$

Since k is an integer, then q must divide $(q-1) \cdot \ldots \cdot (q-8)$. It is easy to show that starting from $k \geq 8$, this is not true.

(2) We now try to choose k such that $\Pr\{(i, z_t, z_{t+1})\}$ is as close to $1/q^3$ as possible. Let $|\delta_{\min}|$ be the smallest value such that $(q-1) \cdot \ldots \cdot (q-8) + \delta_{\min}$ is divisible by q. Then $\Pr\{(i, z_t, z_{t+1})\} = \frac{1}{q^3} \pm \frac{q\cdot|\delta_{\min}|\cdot(q-9)!}{q^3\cdot q!}$. The minimum value of $|P_{\mathrm{Random}} - P^{\mathrm{VMPC}-k}_{(i,z_t,z_{t+1})}|$ is then derived as

$$\epsilon_{\min} = q^3 \cdot \frac{q \cdot |\delta_{\min}| \cdot (q-9)!}{q^3 \cdot q!} = \frac{|\delta_{\min}| \cdot q \cdot (q-9)!}{q!}. \tag{6.6}$$

(3) For VMPC-8, the minimum $\delta_{\min}$ is 128. Hence, the lower bound for the bias is $\epsilon_{\min} \approx 2^{-56.8}$. □

For RC4-k a maximum of 6 positions can be fixed, if we use a similar algorithm. Hence, all cells of the distribution table $P^{\mathrm{RC4}-k}_{(i,z_t,z_{t+1})}$ can be written in the form $k \cdot (q-6)!$. By similar arguments as above, we conclude.

Corollary 6.2: For RC4-k, $k \geq 4$, under the assumptions made in Section 6.1.2, the following holds.

1. Each probability in $P^{\mathrm{RC4}-k}_{(i,z_t,z_{t+1})}$ is different from $1/q^3$.
2. The minimum value $\epsilon = |P_{\mathrm{Random}} - P^{\mathrm{RC4}-k}_{(i,z_t,z_{t+1})}|$ is bounded by

$$\epsilon \geq \epsilon_{\min} = \frac{|\delta_{\min}| \cdot q \cdot (q-6)!}{q!} \geq q^{-5k}, \tag{6.7}$$

   where $|\delta_{\min}|$ is the minimum value, such that

$$(q-1)(q-2) \cdot \ldots \cdot (q-5) + \delta_{\min} \equiv 0 \pmod{q}.$$

3. For RC4-k, $k = 4, \ldots, 8$, we have the following lower bounds.

	k=4	k=5	k=6	k=7	k=8
$\delta_{\min}$	$+8$	-8	-8	-8	-120
$\epsilon_{\min}$	$2^{-15.46}$	$2^{-21.28}$	$2^{-26.65}$	$2^{-31.83}$	$2^{-33.01}$

□

Theorem 6.1 above shows the way how one can think when designing a new cipher from the RC4 family of stream ciphers to avoid these weaknesses. For the case of VMPC-8, for instance, we can say that the structure seems to be weak, without any deeper additional investigations of the cipher.

On the other hand, for RC4A-8 our theorem gave us a very small lower bound, so that a hypothetical distinguisher would be slower than an exhaustive key search. It means that this cipher would resist distinguishing attacks better than, for example, VMPC-8 or RC4-8. Note, these conclusions were made with the assumptions from Section 6.1.2. However, in the next sections we investigate digraphs for both ciphers VMPC-k and RC4A-k in detail.

6.4 Our Distinguisher for VMPC-k

6.4.1 What Should the Probability of $z_t = z_{t+1} = 0$, When $i = 0$ and $j = 1$, Be?

If VMPC-k would be a truly random generator, then the answer to the question of this section would be $1/q^2$, because when i and j are fixed, then $\Pr\{z_t = 0, z_{t+1} = 0 | i = 0, j = 1, \text{Random source}\} = 1/q^2$. In the case of VMPC-$k$ this is not true. The only case when the desired outputs can be produced is depicted in Figure 6.4. All the other permuters will lead to other pairs of outputs $(z_t, z_{t+1}) \neq (0, 0)$. As an example, in Figure 6.5 we show one of the cases, which contradicts the desired conditions.

By this small investigation we have shown that

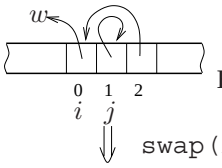
$$\Pr\{z_t = z_{t+1} = 0 | i = 0, j = 1, \text{VMPC-}k\} = \frac{(q-4)(q-4)!}{q!} = \frac{q-4}{q(q-1)(q-2)(q-3)} \approx 1/q^3. \quad (6.8)$$

This is significantly smaller when compared to

$$\Pr\{z_t = z_{t+1} = 0 | i = 0, j = 1, \text{Random source}\} = 1/q^2. \quad (6.9)$$

If we now assume that for the other values of j the probability

$$\Pr\{z_t = z_{t+1} = 0 | i = 0, j \neq 1, \text{VMPC-}k\} \approx 1/q^2, \quad (6.10)$$

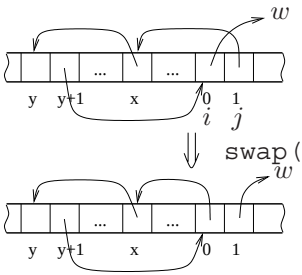


Case 1: $R[j] = j$
$0 = z_t = R[R^2[j] + 1]$
Effect: $w \neq 0, 1$

$j' = j + R[i+1] = 1 + w$
$0 = z_{t+1} = R[R^3[j'] + 1]$
$\Rightarrow R^3[j'] + 1$ m.b. $= 2$
$\Rightarrow R[1 + w]$ m.b. $= 2$
$\Rightarrow$ Effect: $w \neq -1, 0, 1, 2$

$$\text{\# Permuters} = \underbrace{(q-4)}_{\text{choose } w} \cdot \underbrace{1}_{w+1} \cdot (q-4)!$$

Figure 6.4: Condition: $z_t = z_{t+1} = 0$, $i = 0$, $j = 1$. The only case when the condition is satisfied.



Case 2: $R[j] \neq j, i, i-1$
$0 = z_t = R[R^2[j] + 1]$

$j' = j + R[i+1] = 1 + w$
$0 = z_{t+1} = R[R^3[j'] + 1]$
$\Rightarrow R^3[j'] + 1$ m.b. $= y + 1$
$\Rightarrow j'$ m.b. $= y + 1 = w + 1$
$\Rightarrow y = w$ – a contradiction!

Permuters = 0 (cannot exist)

Figure 6.5: Condition: $z_t = z_{t+1} = 0$, $i = 0$, $j = 1$. One of the cases when it is not satisfied.

like in a random case, then we can derive

$$\Pr\{z_t = z_{t+1} = 0 | i = 0\} = \left(\frac{1}{q} \cdot \frac{1}{q^3} + \frac{q-1}{q} \cdot \frac{1}{q^2}\right), \tag{6.11}$$

whereas in a random case it would be $1/q^2$. In the case of a binary distribution of two events, we have a bias $\epsilon \approx 2^{-k}$, and our hypothetical distinguisher needs to observe the event $z_t = z_{t+1} = i = 0$ from around 2^{2k} sam-

ples (i.e., 2^{5k} bytes of the keystream). It means that VMPC-8 can be distinguished from random having around 2^{40} bytes of keystream. In the next section we show how to compute the exact probability $\Pr\{z_t = z_{t+1} = 0 | i = 0\}$ for VMPC-8.

6.4.2 Calculating $\Pr\{z_t = z_{t+1} = 0 | i = 0\}$, When j and $R[\cdot]$ are Random.

We could calculate the complete distribution table $P_{(i,z_t=x,z_{t+1}=y)}$ for VMPC-4, and the bias is $\epsilon \approx 2^{-8.7}$. Unfortunately, we could not apply Algorithm 1 for VMPC-8, because the complexity is 2^{88} – infeasible for a common PC. Instead, we propose to consider only two events $\{z_t = z_{t+1} = 0\}$ and its complement, when $i = 0$. We distinguish between the following two binary distributions:

$$P_{\text{VMPC}-k} = \left(\begin{array}{c} \Pr\{z_t = z_{t+1} = 0\} \\ 1 - \Pr\{z_t = z_{t+1} = 0\} \end{array} \right)\Bigg|_{i=0},$$

and

$$P_{\text{Random}} = \left(\begin{array}{c} 1/q^2 \\ 1 - 1/q^2 \end{array} \right)\Bigg|_{i=0}. \tag{6.12}$$

In Algorithm 2 we give the algorithm for calculating the probability $\Pr\{z_t = z_{t+1} = 0 | i = 0\}$. It has complexity $O(2^{5k})$, i.e., in order to calculate this probability for VMPC-8 we need to make only 2^{40} operations.

After simulation we have received the following result.

Theorem 6.3: For VMPC-8, under the assumptions made in Section 6.1.2,

$$\begin{aligned} \Pr\{z_t = z_{t+1} = 0 | i = 0\} &= \frac{15938227062862998000}{256 \cdot 4096374767995023500000} \\ &\approx 2^{-16}(1 - 2^{-7.98322}), \end{aligned} \tag{6.13}$$

and the bias is $\epsilon \approx 2^{-7.98322}$. Thus, we can distinguish VMPC-8 from random having around $1/(p\epsilon^2) \approx 2^{31.97}$ samples (according to (3.31)), or $2^8 \cdot 2^{31.97} = 2^{39.97}$ bytes of the keystream, when the two events from the equation (6.12) are considered. The cipher and random distributions are the following,

$$P_{\text{Random}} = \left(\begin{array}{c} 2^{-16} \\ 1 - 2^{-16} \end{array} \right)\Bigg|_{i=0}$$

and

$$P_{\text{VMPC}-8} = \left(\begin{array}{c} 2^{-16}(1 - 2^{-7.98}) \\ 1 - 2^{-16}(1 - 2^{-7.98}) \end{array} \right)\Bigg|_{i=0}. \tag{6.14}$$

Algorithm 2: *Recursive computation of* $\Pr\{z_t = z_{t+1} = 0 | i = 0\}$
We use the same operation *define* $R[i]$ as in Algorithm 1.
Do the following steps recursively:

- for all $j = 0 \dots q-1$.
- define $R[j]$, then $R^2[j]$.
- Since $z_t = 0$, then *fix* the position $R[R^2[j]+1] = 0$. If this position is already defined ($\neq \infty$), and the value is not 0, or pointer to 0 is already used, then track back by the recursion.
- define $R[i=0]$.
- swap($R[i]$, $R[j]$).
- set $R[i+1] = R[1]$, if possible, otherwise return by recursion.
- calculate $j' = j + R[i+1]$ which is the same as $j + R[1]$.
- Since $z_{t+1} = 0$, and 0 is already placed in the permuter $R[\cdot]$, then we know the value $R^3[j'] + 1$, hence, we also know the value $R^3[j'] = c$. We can calculate the number of permuters of size q, where $R^3[j'] = c$, and r positions are fixed from the previous steps, by the subalgorithm of complexity $O(q)$, given in Subsection 6.4.4.

□

6.4.3 Simulations of the Attack on VMPC-k

Our theoretical distinguisher from the previous subsection is based on a few assumptions from Section 6.1.2. First of all, by simulations we have checked the distribution of the internal state of VMPC-k for different values of k, and we did not find any noticeable anomalies. From this we conclude that the internal state is indeed distributed close to the uniform distribution, and our theoretical distinguisher should work. Secondly, we can argue that the samples are quite independent, because each sample is connected to the known variable i, and the distance between two samples (for a fixed i) is q rounds of the internal loop.

Theorem 6.3 states that the complexity of the attack on VMPC-8 is $O(2^{39.97})$. However, we have performed simulations on the reduced version VMPC-4, and showed the attack in practice.

VMPC-4 has one permuter of size 16, and the internal indices i and j are taken modulo 16. In our simulations we made $n = 2^{34}$ iterations and from 2^{34} received samples we have constructed the type (empirical distribution) with probabilities $\Pr\{z_t = x, z_{t+1} = y | i\}$. Below we show this table (type) partly.

$n = 2^{34}$	i=0				i=1				...
$x \Rightarrow$	0	1	2	...	0	1	2	...	
	To get the probability of the event $(z_t = x, z_{t+1} = y)\|i$ the corresponding cell should be divided by 16^2. In the case of a random source each such event has the probability $1/16^2$.								
$y \Rightarrow 0$	**0.9247**	0.9987	1.0043		0.9929	0.9909	0.9989		
1	1.0008	0.9881	1.0120		0.9931	0.9966	0.9907		
2	1.0052	1.0057	1.0034	...	0.9950	1.0688	1.0652	...	...
3	1.0063	0.9999	0.9956		1.0008	0.9926	0.9977		
⋮	⋮	⋮	⋮		⋮	⋮	⋮		
15	0.9974	0.9893	1.0085		1.0005	0.9912	0.9950		

This table represents the type P_{Type} and we can see that many probabilities are far away from $1/16^2$, and the most biased probability is in the cell $(0, 0)$, which corresponds to

$$\Pr\{z_t = z_{t+1} = 0 | i = 0\} = \frac{0.924744}{16^2}. \tag{6.15}$$

When the type (the table with probabilities) is derived, one can analyse two possible distinguishers for VMPC-4.

(1) In the first scenario we consider the whole distribution table, i.e., all events of the form $(i, z_t = x, z_{t+1} = y)$. The probability of each event in this case is $1/16^3$. Thus, each cell of the table (type) should be divided by $1/16^3$.

The bias of the received multi-dimensional type is $\epsilon_0 = 2^{-8.679648}$, which is close to the theoretical value calculated in the previous section $\epsilon = 2^{-8.7}$. However, we could not calculate a theoretical bias for VMPC-8, therefore, we consider the second scenario.

(2) In the second scenario we observe only two events $\{z_t = z_{t+1} = 0 | i = 0,$ *the others*$\}$ – as in (6.12). As we have mentioned, the probability of the event $(z_t = z_{t+1} = 0)|i = 0$ is much lower than the corresponding probability in the case of a random source. In this example, the received bias is $\epsilon_0 = 1.0 - 0.924744 \approx 2^{-3.73205}$, which, again, is close to the theoretical value $\epsilon = 2^{-3.755716}$ (calculated in a similar way as for VMPC-8 in Theorem 6.3). Thus, the attack complexity is $16^2/\epsilon^2 \approx 2^{11.7}$ For other values of k the simulation results are presented in the table below.

	k=3	k=4	k=5	k=6	k=7	k=8
Theoretical bias, ϵ	$2^{-2.55}$	$2^{-3.76}$	$2^{-4.87}$	$2^{-5.93}$	$2^{-6.97}$	$2^{-7.98}$
	Simulations of the Attack on VMPC-k					
Number of rounds, n	2^{30}	2^{30}	2^{30}	2^{35}	—	—
The real bias, ϵ_0	$2^{-2.56}$	$2^{-3.73}$	$2^{-4.93}$	$2^{-5.91}$	—	—

Our simulations show that the attack on VMPC-k works in practice. We have also shown that the dependency of the adjacent samples does not have much influence on the type.

6.4.4 Subalgorithm for Algorithm 2

Problem statement: We are given a permuter template of size q, where r positions are already placed, whereas the rest are undefined. We want to calculate the number of permuters satisfying the given template, such that $R^3[j'] = c$, where j' and c are some known positions in the permuter.

The algorithm that solves this problem is given in Figures 6.7 and 6.6.

Sub-Algorithm:[a]

1. Go forward by the path $j' \to R[j'] \to R^2[j'] \to R^3[j']$, as much as possible, but not more then 3 steps. Let g be the point in this path where we have stopped, and l_g be the number of steps we made (from 0 to 3).

2. Go backward by the path $c \to R^{-1}[c] \to R^{-2}[c] \to R^{-3}[c]$, as much as possible, but not more then 3 steps. Let h be the point in the path where we have stoped, and l_h be the number of steps we made (from 0 to 3).

3. if ($l_g = 3$ and $g \neq c$) or ($l_h = 3$ and $h \neq j'$) then return 0;
   if ($l_g = 3$ and $g = c$) or ($l_h = 3$ and $h = j'$) then return $(q - r)!$;
   if ($l_g + l_h \geq 3$) return 0.

4. Count the number t_1 of positions $x \neq g, h$ in the permuter $R[\cdot]$ for which $R[x] = R^{-1}[x] = \infty$ (see Fig. 6.7(1)).
   Count the number t_2 of positions $x \neq g, h$, for which $R[x] \neq \infty, g, h$, and $R^{-1}[x] = R^2[x] = \infty$ (see Fig. 6.7(2)).

5. Now there are 7 possibilities to connect positions g and h, and they are depicted in Figure 6.7(a–g):

a)	$g = h, l_g + l_h = 0$	$\Rightarrow$ add $(q - r - 1)!$	cmb.[b]
b)	$g = h, l_g + l_h = 0, t_1 \geq 2$	$\Rightarrow$ add $t_1(t_1 - 1)(q - r - 3)!$	cmb.
c)	$g = h, l_g + l_h = 0$	$\Rightarrow$ add $t_2(q - r - 2)!$	cmb.
d)	$g \neq h, l_g + l_h = 2$	$\Rightarrow$ add $(q - r - 1)!$	cmb.
e)	$g \neq h, l_g + l_h = 1$	$\Rightarrow$ add $t_1(q - r - 2)!$	cmb.
f)	$g \neq h, l_g + l_h = 0, t_1 \geq 2$	$\Rightarrow$ add $t_1(t_1 - 1)(q - r - 3)!$	cmb.
g)	$g \neq h, l_g + l_h = 0$	$\Rightarrow$ add $t_2(q - r - 2)!$	cmb.

[a] The complexity of the subalgorithm is $O(q)$

[b] Here "cmb." is a short form for "combinations"

Figure 6.6: Subalgorithm for Algorithm 2.

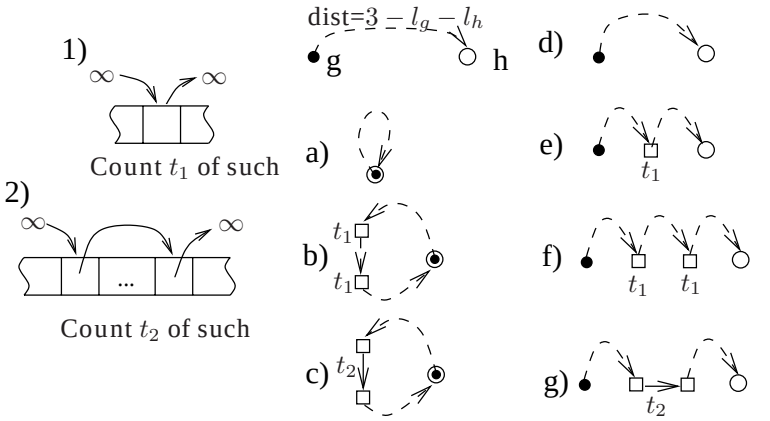


Figure 6.7: Possibilities to connect g and h, used in subalgorithm.

6.5 Our Distinguisher for RC4A-k

6.5.1 Building a Distinguisher

In this section we investigate the cipher RC4A-k (see Figure 6.3), and propose our distinguisher for RC4A-8. We again idealize the situation by the preliminary assumptions from Section 6.1.2, i.e., at any time t the values $j_1, j_2, R_1[\cdot]$, and $R_2[\cdot]$ are considered from the uniform distribution, and unknown for us. We would like to investigate the following scheme extracted from RC4A-k.

i – known value at even time t
$j_1, j_2, R_1[\cdot], R_2[\cdot]$ – are from a random source

1. $z_t = R_2[R_1[i] + R_1[j_1]]$
2. swap $(R_2[i], R_2[j_2])$
3. $z_{t+1} = \ldots$
4. $z_{t+2} = R_2[R_1[i+1] + R_1[j_1 + R_1[i+1]]]$

For cryptanalysis of RC4A-k, we use similar ideas as before. Our methodology of finding anomalies for both VMPC-k and RC4A-k was just to consider the distribution tables like $P_{(i,z_t,z_{t+2})}$ for small values of k, using an Al-

gorithm 1-like procedure. If some anomaly is found then we concentrate on them in particular for larger values of k, and try to understand why anomalies exist.

For RC4A-k we have noticed that $\Pr\{z_t = z_{t+2} |\ i \text{ is even}\} \neq 1/q$, i.e., these probabilities do not correspond to the random distribution. The complementary probabilities $\Pr\{z_t \neq z_{t+2} |\ i \text{ is even}\}$ are equal to each other, but not equal to $1/q$. On the other hand, all probabilities $\Pr\{z_t = z_{t+2} |\ i \text{ is odd}\} = 1/q$ – correspond to the random distribution. So, our target is to calculate the probabilities $\Pr\{z_t = z_{t+2} |\ i \text{ is even}\}$ for RC4A-8. We have used a similar idea as in the Algorithm 2, but much simpler. Our optimised search algorithm to find all such probabilities has complexity $O(2^{6k})$. The result of this work is the following.

Theorem 6.4: For RC4A-k, under the assumptions made in Section 6.1.2, consider the following vector of events, and its random distribution,

$$\text{Events} = \begin{pmatrix} z_t = z_{t+2}, i = 0 \\ z_t = z_{t+2}, i = 2 \\ \vdots \\ z_t = z_{t+2}, i = q-2 \\ \text{other cases} \end{pmatrix}, \quad P_{\text{Random}} = \begin{pmatrix} 1/q^2 \\ 1/q^2 \\ \vdots \\ 1/q^2 \\ 1 - 1/(2q) \end{pmatrix}. \quad (6.16)$$

For RC4A-8, the bias $P_{\text{RC4A}-8}$ is $\epsilon \approx 2 \cdot 2^{-30.05}$. Hence, our distinguisher needs around 2^{58} bytes of the keystream. □

6.5.2 Checking the Assumptions

By simulations we found that the internal state of RC4A-k is not close to the uniform distribution. We could clearly see these anomalies when running simulations many times for different k, each time sampling from at least $n = 2^{30}$ rounds of the loop. To begin counting anomalies, we would like to note that the internal variables $j_1, R_1[\cdot]$ are updated independently from $j_2, R_2[\cdot]$ as follows.

One-Round-Update for $j_*, R_*[\cdot]$, where $*$ is 1 or 2

1. $i{+}{+}$
2. $j_* += R_*[i]$
3. swap $(R_*[i], R_*[j_*])$

It means that all anomalies found for $j_1, R_1[\cdot]$ are true for $j_2, R_2[\cdot]$ as well. We found an event for which the probability is far from the probability of this event in the case of a random source. In particular, $\Pr\{j_1 =$

$i+1\} \approx \frac{q-1}{q^2}$, when in the random case it should be $1/q$. Other probabilities are $\Pr\{j_1|i, j_1 \neq i+1\} \approx \frac{q^2-q+1}{q^2(q-1)}$. For example, for RC4A-4, it appeared that $\Pr\{j_1 = i+1\} \approx 0.05859375$, and the others are $\Pr\{j_1|i, j_1 \neq i+1\} \approx 0.06276042$ – the difference is noticeable. Some other less notable non-uniformities in the internal state were also found.

6.5.3 Simulations of the Attack on RC4A-k

Despite finding the non-uniformity of the internal state of RC4A-k we make a set of simulations to see how our distinguisher behaves. We consider the attack scenario as in Theorem 6.4.

	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$
Theoretical bias, ϵ	$2^{-10.01}$	$2^{-14.01}$	$2^{-18.00}$	$2^{-22.00}$	$2^{-26.00}$	$2^{-29.05}$
	Simulations of the Attack on RC4A-k					
Number of rounds, n	2^{30}	2^{30}	2^{34}	2^{40}	2^{40}	—
The real bias, ϵ_0	$2^{-8.92}$	$2^{-12.27}$	$2^{-15.07}$	$2^{-18.04}$	$2^{-20.03}$	—

Note that the number of actual samples n in our simulations is larger than $1/\epsilon_0^2$. From (3.29) it follows that we have distinguished the cipher with a very small probability of error, and the real theoretical bias without presumptions should be close to what we get in our simulations. From the table above we see that the bias in practice (when the internal state is not from the uniform distribution) is larger than the approximated value of the bias (the uniformly distributed internal state), for $k = 3, \ldots, 7$. The same behaviour of the distinguisher we expect for $k = 8$ as well. Since we could not perform simulations for $k = 8$, we decided to leave theoretical bias as the lower bound of the attack, i.e., $\epsilon = 2^{-29.05}$ for $k = 8$, the complexity is $O(2^{58})$. However, we expect this bias to be even larger, and the complexity of the attack lower.

6.6 Summary

In this chapter we have shown some theoretical weaknesses of the RC4 family of stream ciphers. We have also investigated the recently suggested stream ciphers VMPC-k and RC4A-k, and found linear distinguishing attacks on them. They are regarded as academic attacks which show weak places in these ciphers. The table below summarizes our results in this chapter:

Cipher	Theoretical Lower Bound for ϵ, $k=8$	Our Distinguishers Complexity (# of symbols)					
		$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$
RC4-k'87	2^{-33} (Cor.6.2)	—	—	—	—	—	$2^{30.6}$, in [FM00]
VMPC-k'04	$2^{-56.8}$ (Th.6.1)	2^{16}	* 2^{20}	2^{25}	2^{30}	2^{35}	$\mathbf{2^{40}}$
RC4A-k'04	—	2^{18}	2^{28}	2^{36}	2^{44}	2^{52}	$\mathbf{2^{58}}$

The distinguisher for VMPC-8 that we propose is the following [2]:

Distinguisher for VMPC-8:

(*i*) Observe $n = 2^{40}$ output bytes. Calculate the number L of occurences such that $a = z_t = z_{t+1} = 0$.

(*ii*) Calculate two distances:
$\lambda_{\text{Random}} = |2^{-16} - 2^8 \cdot L/n|$
$\lambda_{\text{VMPC}} = |(2^{-16} - 2^{-23.98322}) - 2^8 \cdot L/n|$

(*iii*) If $\lambda_{\text{Random}} > \lambda_{\text{VMPC}}$ then **keystream of VMPC-8**, else **a random sequence**.

If the internal state of a cipher from the RC4 family is uniformly distributed, then, based on our discussions in Section 6.3, we conclude that such ciphers are not very secure. When the internal state is non-uniformly distributed then the real bias would more likely be larger rather than smaller, and the complexity of the attack would be lower, in most cases. We could observe that effect on the example of RC4A-k. It seems that the security level of such constructions depends more on the degree of the recursive relations between output symbols and internal states, rather than on the length of the permuter(s).

One of the solutions to protect against of such distinguishing attacks is to increase the number of accesses to the permuter(s) in the loop. This solution will increase the relation complexity between adjacent outputs. Another solution is to discard some output symbols before accepting one. Unfortunately, both the suggestions significantly decrease the speed of these ciphers and the main purpose of such designs (speed) is then destroyed.

[2] The distinguisher for RC4A-8 is in a similar fashion as for VMPC-8.

* In the first scenario from Subsection 6.4.3 the attack complexity for VMPC-4 is $O(2^{18})$.

7

Cryptanalysis of “Scream”

“Scream” by Edvard Munch

“Anyone attempting to generate random numbers by deterministic means is, of course, living in a state of sin”

John Von Neumann

Recently, we have seen many proposals of stream ciphers that have not been thoroughly analyzed yet. To mention a few, there are SNOW 2.0 [EJ02b], MUGI [WFY+02], Scream [HCJ02], Turing [RH03], Rabbit [BVP+03] and Helix [FWS+03]. Most of these ciphers are significantly faster than for example AES, and if we could gain trust in their security they would be very interesting alternatives for encryption.

Scream was developed by the IBM researchers Coppersmith, Halevi, and Jutla in 2002 [HCJ02]. It is a purely software-oriented stream cipher. The design is based on the ideas behind the SEAL stream cipher [RC94], but considered to be more secure (SEAL is not very secure, see e.g. [Flu01]). In the proposal, several versions of Scream are given. The so-called “toy cipher” denoted Scream_0 uses the AES S-box whereas the Scream stream

cipher uses secret S-boxes, generated by the key. A third version Scream$_F$ is also given, but it is not considered in this chapter.

In the security analysis of the Scream family of ciphers [HCJ02], two distinguishing attacks on Scream$_0$ (the “toy version”) were proposed. The best one has complexity around 2^{80}. However, for Scream, no attacks have yet been published, but the authors conjecture that there could exist a linear distinguishing attack if a sequence of 2^{80} output bytes is available.

For Scream, the situation with cryptanalysis is a bit complicated. In our distinguishing attack on Scream, one of the distributions is unknown due the fact that the S-boxes are secret. But as it was evident in Section 3.4.7, this does not affect the possibility of applying a linear distinguishing attack. The detailed description of statistical analysis on Scream is given in Section 7.2.

In this chapter we propose a distinguishing attack on Scream based on a derived linear approximation of the (nonlinear) S-boxes used in the cipher. It also demonstrates how to overcome the problem related to the situation when the S-boxes are unknown. Our distinguisher has a detectable advantage when only 2^{95} words of the keystream are given. The advantage is very close to 1 when the keystream length is 2^{115} words. This means that Scream does not offer full security. By full security we mean that there is no type of attack faster than exhaustive key search. On the other hand, the complexity of the attack is larger than what the Scream inventors conjectured in their paper.

The rest of the chapter is organized as follows. First in Section 7.1 we describe the stream cipher Scream. Before analysing Scream, in Section 7.2 we give our ideas how a distinguisher for Scream can be built, provide statistical tools for further analysis, and also derive and define our theoretical distinguisher on Scream. Detailed analysis of the cipher, its algorithms and internal states provided is in Section 7.3. In Subsection 7.3.4 we introduce noise variables due approximations, and refer to standard assumptions in linear cryptanalysis given in Subsection 7.2.2. The results of our simulations are presented in Section 7.4. Section 7.5 contains a short discussion on computational aspects. In Section 7.6 we propose possible improvements of our attack. Finally, in Section 7.7 we shortly discuss our ideas and conclude.

Definitions and Notation

In this chapter variables denoted by a capital letter X will usually mean a 16-byte block, and its individual bytes we denote by $X = (x_0, x_1, \ldots, x_{15})$. We introduce the most common variables and their notation used along the chapter:

$\mathbf{P} = (P_0, P_1, \ldots)$	plaintext
$\mathbf{C} = (C_0, C_1, \ldots)$	ciphertext
$\mathbf{O} = (O_0, O_1, \ldots)$	keystream (output stream)
$K = (key_0, \ldots, key_{15})$	the secret key
$S_R[\cdot], S_1[\cdot], S_2[\cdot]$	byte oriented S-boxes
P_X	distribution for some random variable X
$P_X(x) = \Pr\{X = x\} = \Pr\{x\}$	probability of the event $X = x$
$P_{\mathcal{U}}$	uniform distribution
$P_{\mathcal{N}}$	noise distribution
ϵ	distance between distributions
$P_{\mathbf{x}}$	type, or an empirical distribution
n	the number of samples for $P_{\mathbf{x}}$
m	the number of subdistinguishers
$\mathtt{SD}_i$	i-th subdistinguisher
p_α, p_β	error probabilities of two kinds in hypothesis testing
p_{err}	probability of error for one subdistinguisher

Definition 7.1: For any 16-byte vector $X = (x_0, \ldots, x_7, x_8, \ldots, x_{15}) \in \mathbb{F}_{2^8}^{16}$ and any integer number $i \in \{0, 1, \ldots 15\}$ we define $\hat{X}$ and $\hat{i}$ as

$$\hat{X} = (x_8, \ldots, x_{15}, x_0, \ldots, x_7) \tag{7.1}$$

and

$$\hat{i} = i + 8 \mod 16. \tag{7.2}$$

□

In formulas, symbol $\boxplus$ means a sum modulo 2^8, whereas $\oplus$ or just $+$ means the XOR of the two arguments.

7.1 A Short Description of Scream

The stream cipher Scream functionality is presented in Figure 7.1. Scream takes as an input a 128 bit key and a nonce value. The nonce value can be viewed as a part of the key that is allowed to be public. For any such pair, Scream produces an arbitrary long pseudo-random sequence (keystream sequence), denoted by $\mathbf{O} = O_0, O_1, O_2, \ldots$, where $O_i \in \mathbb{F}_{2^8}^{16}$.

In the initialisation stage, Scream initialises a number of state variables (X, Y, Z) and tables (W and S-box tables), see [HCJ02] for the details. Then Scream enters the "main loop", where in each iteration ($i = 1, 2, \ldots$) in the main loop the state variables and tables are updated and one output word O_i is generated.

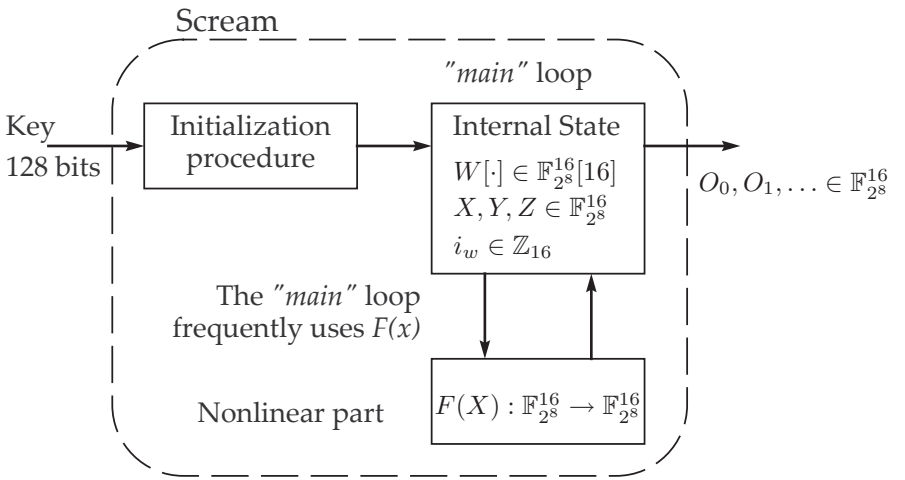


Figure 7.1: Scream functionality.

The calculations in the "main" loop of Scream are based on a certain function $F(X) : \mathbb{F}_{2^8}^{16} \to \mathbb{F}_{2^8}^{16}$ called the "round function". This function is quite similar to a round function in block ciphers and uses S-boxes with mixing operations.

The structure of $F(X)$ is illustrated in Figure 7.2. This function uses two different instances of a "half-round" function, denoted G_{S_1,M_1} and G_{S_2,M_2}, where S_1, S_2 are two S-boxes, and M_1, M_2 are two matrices.

An important aspect from the initilisation is the fact that Scream uses secret (keyed) S-boxed, derived from the key (but independent of the nonce) using the AES S-box, as follows,

$$\text{set } S_1[x] := S_R[\ldots S_R[S_R[x \boxplus key_0] \boxplus key_1] \ldots \boxplus key_{15}], \quad \text{for all } x \in [0 \ldots 255], \tag{7.3}$$

where S_R is the AES S-box, and $key_0, \ldots, key_{15}$ are 16 bytes of the secret key K (Note that in (7.3) $\boxplus$ denotes integer addition modulo 256). Furthermore, the second S-box is defined as $S_2[x] = S_1[x \oplus 00010101]$.

All byte operations in Scream are evaluated in the field $\mathbb{F}_{2^8}$ with generating polynomial $g(x) = x^8 + x^7 + x^6 + x + 1$. Let α denote the primitive element such that $g(\alpha) = 0$. We sometimes write "+" instead of "$\oplus$" when operating with bytes in Scream.

Finally, we give the description of the "main" loop of Scream. A pseudocode of the "main" loop of Scream is as follows.

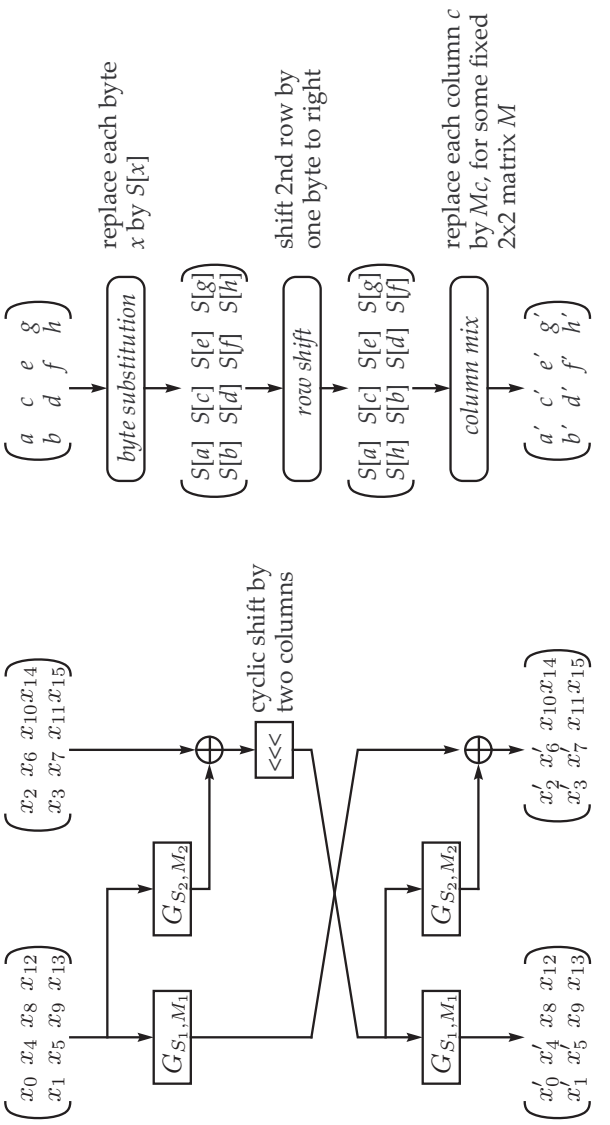


Figure 7.2: The schemes of the round function $F(X)$ (left) and the "half round" function $G_{S,M}$ (right), respectively.

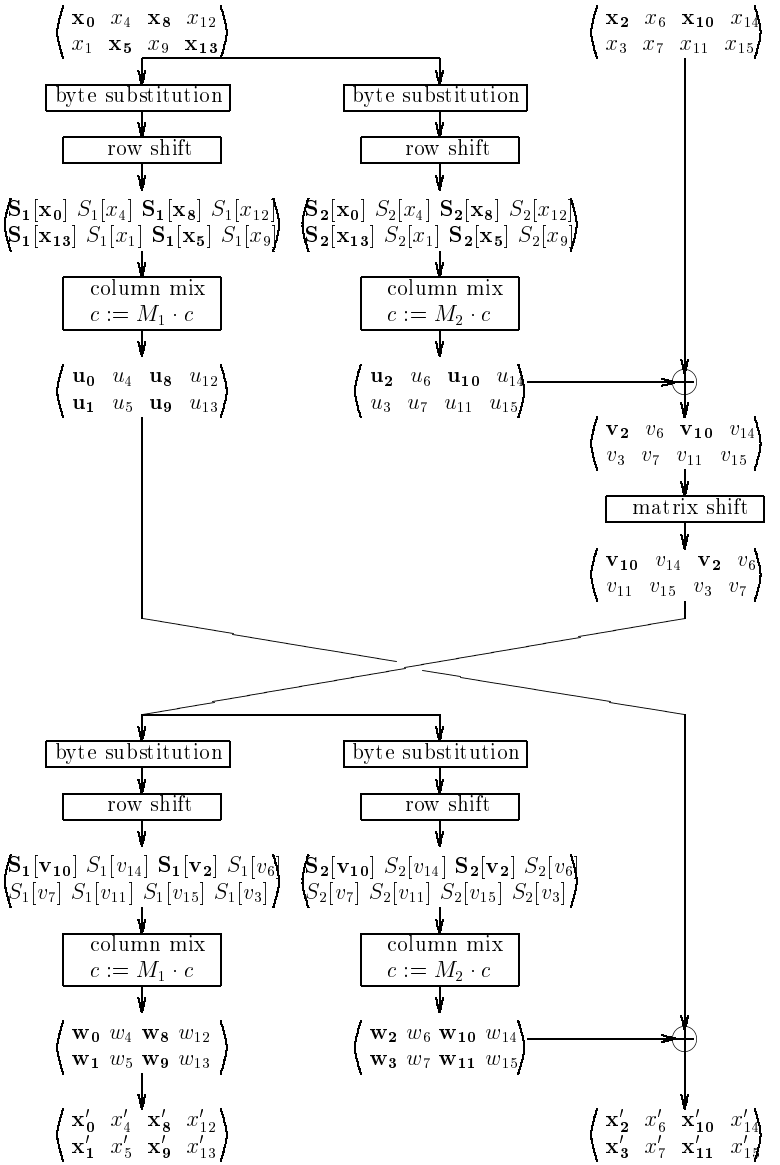


Figure 7.3: Scream round function in details. Variables in bold are used to form linear relations for the attack.

State: X, Y, Z – three 16-byte blocks
W – a table of sixteen 16-byte blocks
i_w – an index into W (initially $i_w = 0$)

1. repeat (until you get enough output bytes)
2. for $i = 0$ to 15
3. $X := F(X \oplus Y)$
4. $X := X \oplus Z$
5. output $X \oplus W[i \bmod 16]$
6. if $i = 0 \bmod 2$ then
7. rotate Y by 8 bytes, $Y := Y_{8..15,0..7}$
8. else if $i = 1 \bmod 4$ then
9. rotate each half of Y by 4 bytes, $Y := Y_{4..7,0..3,12..15,8..11}$
10. else if $i < 15$
11. rotate each half of Y by three bytes to the right, $Y := Y_{5..7,0..4,13..15,8..12}$
12. end–if
13. end–for
14. $Y := F(Y \oplus Z)$
15. $Z := F(Z \oplus Y)$
16. $W[i_w] := F(W[i_w])$
17. $i_w := i_w + 1 \bmod 16$
18. end–repeat

In line 5. above the output sequence $\mathbf{O} = O_0, O_1, O_2, \ldots$ is generated, one word each time.

In summary, the X and Y variables are frequently updated in the inner loop using the $F()$ function, whereas the Z variable is slowly updated once in each round. One 16-byte vector of the table W is updated during one round, whereas the remaining entries are unchanged.

For analysis purposes of the round function $F(X)$ in further sections, we also present its detailed structure in Figure 7.3. Highlighted variables are used to derive the first approximated expression in equation (7.16), described later in the chapter.

We have given a very brief overview of the design of Scream, without explaining the initialisation using the key and the nonce. For this and a more detailed description of Scream, we refer to [HCJ02].

7.2 Preparing a Distinguisher for Scream

7.2.1 Ideas for the Distinguisher

The main ideas behind our distinguisher for Scream can be highlighted here as follows.

(a) The internal state of Scream is large. However, the table $W[\cdot]$ is changed slowly along the rounds. This fact can be used to eliminate the contribution from $W[\cdot]$, if one considers rounds where the table is unchanged. This is possible since the index i, which selects the row from the table $W[i]$, is known.

(b) Due to approximations of nonlinear blocks in the cipher, a linear combination of noise variables is introduced. Since the S-boxes are keyed, then the distribution of the sum of noise variables is unknown. A type (an empirical distribution) is constructed from the samples, which are collected from the keystream. Usually, we need to test whether the constructed type is from the noise distribution or from the uniform distribution. However, since the noise distribution is unknown, we, instead, test the distance from the type to the uniform distribution only.

(c) The distance between the noise and the uniform distribution depends on the secret key and a chosen approximation function. In this case we perform a hypothesis testing for many randomly chosen approximations. We hope that for at least one of the approximations, the noise distribution has a large distance from the uniform distribution. With a larger distance the number of required samples becomes less. If one of the tests shows that the distance between the type and the uniform distributions is far away, then we conclude that the given stream (keystream) is from the cipher.

7.2.2 Assumptions

As we mentioned before, in our attack on Scream we approximate nonlinear parts by some linear functions. During these substitutions we introduce noise variables. To continue, we need to make a few assumptions, which are usually standard for linear cryptanalysis.

1. We assume that after each approximation of a nonlinear part of Scream we introduce a new *independent noise random variable*. However, if two parts of the cipher are approximated then in the real life these two new noise variables will be dependent, since they both come from the same source. Usually, these kinds of dependencies are rather small, and in the case of Scream we can note a similar situation. In the attack on Scream the inputs to nonlinear parts will be *almost independent* (contain different cells from the table W), when approximate S-boxes. Therefore, we ommit this small dependency and make such an assumption. We use this assumption when, for example, we derive the distribution of linear combinations of noise variables in equations (7.17) and (7.19).

2. From the output sequence we collect samples, which form a type (a pseudo empirical distribution). We assume that *all samples are independent*. However, this is not true in the real life. We think that we sample from a local distribution $P_{\mathcal{N}}$, but the samples are dependent. If, for example, at time t we approximate some parts A_0, A_1, and A_2, and in time $t+1$ approximated parts are A_1, A_2, and A_3 – obviously, two consecutive samples will be dependent.

   However, we can make that assumption as well. Between two consecutive samples their dependency is negotiated by the inherent operations of the cipher Scream, which is supposed to scramble the information[1].

3. The distinguisher for Scream consists of a set of m subdistinguisher. When deriving formulas for success and error probabilities, we also assume that subdistinguishers are independent.

In the following of this chapter we refer to these assumptions during derivation of formulas and results.

7.2.3 A Distinguisher for Scream

As it was mentioned in the introduction part, the main idea in a linear distinguishing attack is to find suitable linear approximations. Nonlinear parts of a cipher are substituted by some linear functions, and the introduced errors are compensated by introducing noise variables. This substitution is modelled through the notation

$$S(x) = R(x) + N(x),$$

where $S(x)$ is the nonlinear part that we try to approximate, $R(x)$ is a linear function, and $N(x)$ denotes a new unknown random variable with a (usually) biased distribution. For increased efficiency of the attack, the introduced noise variables should have a significant bias. This is completely determined by the choice of the linear approximation of the corresponding nonlinear operation.

After approximation, all operations in the "linearised" part of the cipher are linear. In this case we find some linear expression L_1 including only symbols from the output stream of the "linearised" cipher (the linear cipher without noise variables), which is always equal to a constant (usually equal to 0).

If we then apply the expression L_1 to the real cipher, where now the noise variables are included, then, obviously, we get the second linear expression L_2 consisting of the noise variables only, that sum to some linear

[1] In the case when two consecutive samples are very much dependent one can skip a few samples before accepting one, making the dependency as small as necessary

function of output symbols L_1. If we know the distribution of each noise variable, then, usually, it does not take much effort to get the distribution of the linear combination of all of them, L_2, which we denote by $P_{\mathcal{N}}$. These noise variables are dependent, but we assume them beeing independent, as stated in Subsection 7.2.2. In this way, we collect samples from the approximated distribution $P_{\mathcal{N}}$. A collection of n samples then constructs a type $P_{\mathbf{x}}$.

If the output is a truly random sequence, then the collected samples (the type $P_{\mathbf{x}}$) are drawn from the uniform distribution $P_{\mathcal{U}}$, since any nonzero linear combination of uniformly distributed random variables produces samples from $P_{\mathcal{U}}$ as well. Finally, the decision rule from the hypothesis testing algorithm gives the answer, testing $P_{\mathbf{x}} \leftarrow P_{\mathcal{N}}$ against $P_{\mathbf{x}} \leftarrow P_{\mathcal{U}}$ [2].

When examining the above procedure regarding the Scream cipher, we have some interesting observations. In particular, the only nonlinear part in the cipher is the S-box [3], which is a one-to-one function $S : \mathbb{F}_{2^8} \rightarrow \mathbb{F}_{2^8}$. As all operations in Scream are byte oriented, it is appropriate to *consider linear approximations over the field* $\mathbb{F}_{2^8}$. This is in opposite to the general approach in linear cryptanalysis, which usually considers binary approximations.

Furthermore, the S-box is secret since it is initialized with the secret 128 bit key K. The situation when one of the distributions is unknown makes the direct use of a hypothesis testing algorithm useless. But the problem can be resolved by the following approach.

Definition 7.2 (Distinguisher for Scream): The distinguisher for Scream is an X-Distinguisher, for which *known* random distribution is $P_0 = P_{\mathcal{U}}$, and *unknown* noise distribution is $P_1 = P_{\mathcal{N}}$.

Each subdistinguisher $\mathtt{SD}_i$ uses a randomly chosen linear function $R_i : \mathbb{F}_{2^8} \rightarrow \mathbb{F}_{2^8}$ which is its approximation of the S-box, when the key K is the same for all subdistinguishers. Let $P_{\mathcal{N}(K,R_i)}$ denote the corresponding unknown noise distribution, and let $\epsilon_i = |P_{\mathcal{N}(K,R_i)} - P_{\mathcal{U}}|$, which is also unknown. From the keystream $\mathbf{O}$, each $\mathtt{SD}_i$ constructs its own type $P_{\mathbf{x}i}$, according to the linear combination L_1 described above.

The overall distinguisher for Scream is then as follows,

$$\delta(\mathbf{O}) = \begin{cases} (\mathtt{CIPHER}) \text{ if } \mathtt{SD}_i(\mathbf{O}) = P_{\mathcal{U}} \text{ for At Least One } i = 1, 2, \ldots, m\,, \\ (\mathtt{RANDOM}) \text{ if } \mathtt{SD}_i(\mathbf{O}) = P_{\mathcal{N}} \text{ for All } i = 1, 2, \ldots, m. \end{cases} \tag{7.4}$$

□

[2] In the notation of Subsection 3.4.7, here the distribution $P_{\mathcal{U}}$ is $P_{\mathcal{R}}$, and $P_{\mathcal{N}}$ is $P_{\mathcal{C}}$.

[3] Scream uses two secret S-boxes S_1 and S_2. First S_1 is initialised by the secret key K, and the second box $S_2(x)$ is defined as $S_1(x \oplus c)$, where c is the constant of design. Therefore, here and further on we talk about one secret S-box in Scream, in particular S_2.

Error probabilities for this distinguisher are given in Sections 3.4.7 and 3.4.8. More detailed investigation of Scream and findings of the linear expressions L_1 and L_2, estimation of ϵ_0, m and n values are described in the next sections.

7.3 Scream Structure Analysis

Having described the general ideas on how to mount a distinguishing attack and having discussed some particular issues relating to Scream, we are now ready to do the detailed analysis. In particular, we want to find a good "path" through the cipher which gives rise to the L_1 and L_2 linear expressions discussed before. For this purpose we analyze the details of the components of Scream. The detaled structure of the round function is shown in Figure 7.3.

7.3.1 The F–Function Analysis

In this subsection we analyze the round function $F(\cdot)$ and introduce some notation. For each output byte from the function, we derive analytical expressions and obtain some useful properties.

Definition 7.3: Let us define the function $\psi : \mathbb{F}_{2^8}^6 \rightarrow \mathbb{F}_{2^8}$ as:

$$\begin{aligned}\psi(p_1, \ldots, p_6) =& S_1[p_1 + S_2[p_2] \\ &+ (1+\alpha) \cdot S_2[p_3]] + \alpha \cdot S_1[p_4 + S_2[p_5] + (1+\alpha) \cdot S_2[p_6]],\end{aligned} \tag{7.5}$$

and the function $\gamma : \mathbb{F}_{2^8}^8 \rightarrow \mathbb{F}_{2^8}$ as:

$$\begin{aligned}\gamma(p_1, \ldots, p_8) =& S_1[p_1] + \alpha \cdot S_1[p_2] + S_2[p_3 + S_2[p_4] + (1+\alpha) \cdot S_2[p_5]] \\ &+ (1+\alpha) \cdot S_2[p_6 + S_2[p_7] + (1+\alpha) \cdot S_2[p_8]],\end{aligned} \tag{7.6}$$

where $S_1[\cdot]$ and $S_2[\cdot]$ are two secret S-boxes used in Scream, and $p_i \in \mathbb{F}_{2^8}$. □

Let us use the following notation for the function F:

$$\begin{aligned}F(X) &= F(x_0, x_1, \ldots, x_{15}) = (f_0(X), f_1(X), \ldots, f_{15}(X)) \\ &= (x'_0, x'_1, \ldots, x'_{15}) = X',\end{aligned} \tag{7.7}$$

where $X, X' \in \mathbb{F}_{2^8}^{16}$ and $x'_0, x'_1, \ldots, x'_{15}, x_0, x_1, \ldots, x_{15} \in \mathbb{F}_{2^8}$. The function $F(X)$ satisfies the following two properties.

Proposition 7.1: For $k = 0, 1, 2, 3$ and all indices taken modulo 16,

$$\begin{aligned}
x'_{4k+0} = f_{4k+0}(X) =& \psi(x_{4k+10}, x_{4k+8}, x_{4k+5}, x_{4k+7}, x_{4k+1}, x_{4k+4}),\\
x'_{4k+1} = f_{4k+1}(X) =& \psi(x_{4k+7}, x_{4k+1}, x_{4k+4}, x_{4k+10}, x_{4k+8}, x_{4k+5}),\\
x'_{4k+2} = f_{4k+2}(X) =& \gamma(x_{4k+0}, x_{4k+13}, x_{4k+10}, x_{4k+8}, x_{4k+5}, x_{4k+7},\\
& x_{4k+1}, x_{4k+4}),\\
x'_{4k+3} = f_{4k+3}(X) =& \gamma(x_{4k+13}, x_{4k+0}, x_{4k+7}, x_{4k+1}, x_{4k+4}, x_{4k+10},\\
& x_{4k+8}, x_{4k+5}).
\end{aligned} \tag{7.8}$$

Proof: The equations can be verified against the algorithm description. □

Proposition 7.2: For any $X \in \mathbb{F}_{2^8}^{16}$, $j = 0, 1, \ldots, 15$,

$$f_j(X) = f_{\hat{j}}(\hat{X}). \tag{7.9}$$

Proof: The proof is a case by case verification for $j = 0, 1, 2, 3$ using Proposition 7.1 and Definition 7.3. □

7.3.2 The S-box Approximation

Let us represent the S-box function $S_2[x]$ as a sum of some arbitrarily selected linear function

$$R(x) = \sum_{i=0}^{7} a_i x^{2^i}, \quad \text{where } x, a_i \in \mathbb{F}_{2^8}, \tag{7.10}$$

and an unknown noise term $N(x)$. Then,

$$\begin{cases} S_2[x] = R(x) + N(x), \\ S_1[x] = S_2[x + c], \end{cases} \tag{7.11}$$

where c is a known constant used in Scream. The function $R(x)$ is *a linearised polynomial*[4] in the field $\mathbb{F}_{2^8}$ of characteristic 2, since the property $R(x + y) = R(x) + R(y)$ holds, but $R(c \cdot x) \neq c \cdot R(x)$ in general. In the attack we can use this type of approximation and we call it *linear*. We would like to select a linear function $R(x)$ as close to the S-box mapping as possible. This means that the distribution of the noise $N(x)$ (when x is selected at random) is as far as possible from the uniform distribution. This can be measured by the statistical distance introduced in Section 7.2.

Since the relation between two S-boxes S_1 and S_2 is obvious, in this chapter we consider the approximations only for one S-box, in particular for S_2.

[4] Another name for these polynomials is *generic vectorial linear functions*.

7.3.3 The "Main" Loop Analysis

In this section we investigate the expressions for the output blocks $O_i \in \mathbb{F}_{2^8}^{16}$, $i = 0, 1, \ldots$, each consists of 16 bytes according to the "main" loop of Scream. Let us first introduce some notation.

Definition 7.4: We introduce the notation $X(i), Y(i)$ and $Z(i)$ for the value of the internal state variables X, Y, Z **after** the i^{th} time they are assigned new values in the main loop, with $X(0), Y(0)$ and $Z(0)$ being the values of X, Y, Z when we output the first block O_0. □

Then we start to derive expressions for the output blocks as follows.

$$
\begin{aligned}
O_0 \;&=\; X(0) + W[0], \text{ hence } \Rightarrow X(0) = O_0 + W[0].\\
O_1 \;&=\; X(1) + W[1], \text{ hence } \Rightarrow X(1) = O_1 + W[1],\\
&\quad \text{where } \begin{cases} Y(1) = Y(0) \text{ rotated by 8 bytes} = \hat{Y}(0),\\ X(1) = F(X(0) + Y(1)) + Z(0) =\\ \quad = F(O_0 + W[0] + \hat{Y}(0)) + Z(0), \end{cases}\\
&\quad \Rightarrow O_1 = F(O_0 + W[0] + \hat{Y}(0)) + Z(0) + W[1].\\
O_2 \;&=\; X(2) + W[2], \text{ hence } \Rightarrow X(2) = O_2 + W[2],\\
&\quad \text{where } \begin{cases} Y(2) = Y(1) \text{ rotated each half by 4 bytes },\\ X(2) = F(X(1) + Y(2)) + Z(0) =\\ \quad = F(O_1 + W[1] + Y(2)) + Z(0), \end{cases}\\
&\quad \Rightarrow O_2 = F(O_1 + W[1] + Y(2)) + Z(0) + W[2].\\
O_3 \;&=\; X(3) + W[3], \text{ hence } \Rightarrow X(3) = O_3 + W[3],\\
&\quad \text{where } \begin{cases} Y(3) = Y(2) \text{ rotated by 8 bytes} = \hat{Y}(2),\\ X(3) = F(X(2) + Y(3)) + Z(0) =\\ \quad = F(O_2 + W[2] + \hat{Y}(2)) + Z(0), \end{cases}\\
&\quad \Rightarrow O_3 = F(O_2 + W[2] + \hat{Y}(2)) + Z(0) + W[3].\\
&\vdots\\
O_{16+2} \;&=\; X(16+2) + W[2], \text{ hence } \Rightarrow X(16+2) = O_{16+2} + W[2],\\
&\quad \text{where } Z(1) = F(Z(0) + Y(16)),\\
&\quad \Rightarrow O_{16+2} = F(O_{16+1} + W[1] + Y(16+2)) + Z(1) + W[2].\\
O_{16+3} \;&=\; X(16+3) + W[3], \text{ hence } \Rightarrow X(16+3) = O_{16+3} + W[3]\\
&\quad \Rightarrow O_{16+3} = F(O_{16+2} + W[2] + \hat{Y}(16+2)) + Z(1) + W[3]\\
&\vdots
\end{aligned}
$$

It follows from the "main" loop description that after the first 16 output blocks only the entry $W[0]$ is changed in the table W. After the next 16 output blocks only $W[1]$ is changed, and so on. This means that $W[2]$ and $W[3]$ are included unchanged in both the expressions for O_2, O_3, and the expressions for O_{16+2}, O_{16+3}, respectively.

Further useful observations are that the variable Z changes its value only after every 16 output blocks. Also, the variable Y is rotated by 8-bytes every second step, otherwise it uses another permutation of its bytes. This leads us to consider the following selection of output blocks.

Let us choose four output blocks $O_{16k+r}, O_{16k+r+1}, O_{16l+r}, O_{16l+r+1}$ such that the triple (k, l, r) has the following properties,

$$\begin{cases} 0 \le k < l \\ r \in \{2, 4, 6, 8, 10, 12, 14\} \\ W[r] \text{ and } W[r \pm 1] \text{ have not been changed between } k^{\text{th}} \text{ and } l^{\text{th}} \text{ rounds.} \end{cases} \tag{7.12}$$

This basically means that we look for two pairs of output blocks (from the rounds k and l) such that the state variable Y is rotated by 8-bytes between the consecutive blocks making up a pair (the value of r is even), and such that $W[r]$ and $W[r \pm 1]$ have not been changed between and during the k^{th} and l^{th} rounds of the "main" loop.

For these four output blocks we obtain the following four equations,

$$\begin{cases} O_{16k+r} &= F(O_{16k+r-1} + W[r-1] + Y(16k+r)) + Z(k) + W[r], \\ O_{16k+r+1} &= F(O_{16k+r} \quad + W[r] \quad + \hat{Y}(16k+r)) + Z(k) + W[r+1], \\ O_{16l+r} &= F(O_{16l+r-1} + W[r-1] \; + Y(16l+r)) + Z(l) + W[r], \\ O_{16l+r+1} &= F(O_{16l+r} \quad + W[r] \quad + \hat{Y}(16l+r)) + Z(l) + W[r+1]. \end{cases} \tag{7.13}$$

7.3.4 Introduction of Noise Variables

Recall, that the function $F(\cdot)$ consists of several operations which are all linear except one, the S-box mapping. Thus, $F(\cdot)$ is linear if and only if the S-box is linear. We represent the S-box as it was proposed in equation (7.11) ($S_2[x] = R(x) + N(x)$). To get the most efficient attack, we generally require a minimum number of approximated S-boxes (active S-boxes). Studying the "linear paths" of these expressions, we find that there are two different relations that contain the minimum number of 24 linearly approximated S-boxes. We continue to focus only on these two relations.

Definition 7.5: Let us introduce two polynomials over the field $\mathbb{F}_{2^8}$

$$R'(x) = R((1+\alpha)R(x)) = \sum_{i=0}^{7} r_i^{2^i+1}(1+\alpha)^{2^i} x^{2^{2i}}, \tag{7.14}$$

and

$$R''(x) = R(R(x)) = \sum_{i=0}^{7} r_i^{2^i+1} x^{2^{2i}}, \tag{7.15}$$

where the coefficients r_i are those used in the $R(x)$ polynomial, i.e., $R(x) = \sum_{i=0}^{7} r_i x^{2^i}$. □

Based on the structure of the $F(\cdot)$ function (see Figure 7.3), we can derive an expression for a linear combination of the bytes x'_0, x'_1, x'_8, and x'_9 as follows,

$$\frac{\alpha^2}{1+\alpha} \cdot (x'_0 + \alpha x'_1 + x'_8 + \alpha x'_9) = \alpha^2 \cdot [R''(x_0 + x_8) + R'(x_5 + x_{13}) + R(x_2 + x_{10})] + N_{0,8}, \quad (7.16)$$

where $N_{0,8}$ is a linear combination of noise variables $N(x)$ and defined as

$$N_{0,8} = \alpha^2 \cdot [N(v_2 + c) + N(v_{10} + c) + R(N(x_0)) + R(N(x_8)) + R((1+\alpha)N(x_5)) + R((1+\alpha)N(x_{13}))]. \quad (7.17)$$

Furthermore, we derive the second expression for a linear combination of the bytes x'_2, x'_3, x'_{10}, and x'_{11} as follows,

$$(x'_2 + (1+\alpha) \cdot x'_3 + x'_{10} + (1+\alpha) \cdot x'_{11}) = \alpha^2 \cdot (R''(x_0 + x_8) + R'(x_5 + x_{13}) + R(x_2 + x_{10})) + (\alpha^2 + \alpha + 1) \cdot R(x_0 + x_8) + R(x_5 + x_{13}) + N_{2,10}, \quad (7.18)$$

where $N_{2,10}$ is defined as

$$N_{2,10} = \alpha^2 \cdot (N(v_2) + N(v_{10})) + (\alpha^2 \cdot R(N(x_0)) + (\alpha^2 + \alpha + 1) \cdot N(x_0 + c)) + (\alpha^2 \cdot R(N(x_8)) + (\alpha^2 + \alpha + 1) \cdot N(x_8 + c)) + (\alpha^2 \cdot R((1+\alpha)N(x_5)) + N(x_5 + c)) + (\alpha^2 \cdot R((1+\alpha)N(x_{13})) + N(x_{13} + c)). \quad (7.19)$$

Here v_2 and v_{10} are two intermediate values obtained in the calculation of the function $F(X)$ (see Figure 7.3 and [HCJ02]). Let us explain in detail the interpretation of the expression for $N_{0,8}$. Above, $N_{0,8}$ represents the noise introduced by the required linear approximations. It denotes a random variable which is a sum of 6 random variables, each one of them corresponding to the noise variable added by the application of the S-box $S_2[\cdot]$ on a particular input variable. We regard these noise variables as independent, according to our assumptions from Subsection 7.2.2. Recall that $N(x)$ is defined as $S_2[x] + R(x)$ and has in general a nonuniform distribution. Each such noise variable $N(x)$ has a certain (but unknown) bias, and the sum of such noise variables will also have a certain (but much lower) bias.

In the next step we combine the expressions (7.16) and (7.18) to get a linear input-output relation for the $F(\cdot)$ function as given in Figure 7.3.

Recall the notation $X, X' \in \mathbb{F}_{2^8}^{16}$ and $F(X) = X'$. We introduce the following notation,

$$\mathcal{L}_{OUT}(F(X)) = \frac{\alpha^2}{1+\alpha}(x'_0+x'_8)+\frac{\alpha^3}{1+\alpha}(x'_1+x'_9)+(x'_2+x'_{10})+(1+\alpha)(x'_3+x'_{11}); \tag{7.20}$$

$$\begin{aligned}\mathcal{L}_{IN}(X) =&\alpha^2 \cdot [R''(x_0+x_8)+R'(x_5+x_{13})+R(x_2+x_{10})]\\ &+\alpha^2 \cdot (R''(x_0+x_8)+R'(x_5+x_{13})+R(x_2+x_{10}))\\ &+(\alpha^2+\alpha+1)\cdot R(x_0+x_8)+R(x_5+x_{13})\\ =&(\alpha^2+\alpha+1)\cdot R(x_0+x_8)+R(x_5+x_{13});\end{aligned} \tag{7.21}$$

$$\begin{aligned}N_\Sigma(X) =&\alpha^2 \cdot [N(v_2+c)+N(v_{10}+c)+R(N(x_0))+R(N(x_8))\\ &+R((1+\alpha)N(x_5))+R((1+\alpha)N(x_{13}))]\\ &+\alpha^2 \cdot (N(v_2)+N(v_{10}))+(\alpha^2 \cdot R(N(x_0))\\ &+(\alpha^2+\alpha+1)\cdot N(x_0+c))+(\alpha^2 \cdot R(N(x_8))\\ &+(\alpha^2+\alpha+1)\cdot N(x_8+c))+(\alpha^2 \cdot R((1+\alpha)N(x_5))\\ &+N(x_5+c))+(\alpha^2 \cdot R((1+\alpha)N(x_{13}))+N(x_{13}+c))\\ =&\alpha^2 \cdot (N(v_2)+N(v_2+c)+N(v_{10})+N(v_{10}+c))\\ &+(\alpha^2+\alpha+1)\cdot (N(x_0+c)+N(x_8+c))\\ &+(N(x_5+c)+N(x_{13}+c)),\end{aligned} \tag{7.22}$$

where c is a known constant, and $N_\Sigma(X)$ is a linear combination of 6 different noise variables each distributed as $N(x)$. Note, if the input for the expression $\mathcal{L}_{OUT}(\cdot)$ is X, then the x_i's are without prime signs.

Proposition 7.3: For any 16-byte vector $X \in \mathbb{F}_{2^8}^{16}$ we have

$$\mathcal{L}_{OUT}(F(X)) = \mathcal{L}_{IN}(X) + N_\Sigma(X). \tag{7.23}$$

□

We have now established a useful linear input-output relation for the $F(\cdot)$ function. The final step is to extend this relation to the whole cipher and remove the $F(\cdot)$-function application. It should result in an expression including only output symbols and biased noise.

Definition 7.6: Let us define $O_{\alpha(k,l,r)}$ and $O_{\beta(k,l,r)}$ as

$$\begin{aligned}O_{\alpha(k,l,r)} &= O_{16k+r}+O_{16k+r+1}+O_{16l+r}+O_{16l+r+1},\\ O_{\beta(k,l,r)} &= O_{16k+r}+O_{16k+r-1}+O_{16l+r}+O_{16l+r-1}.\end{aligned}$$

□

Let us take any triple (k,l,r) satisfying (7.12). Then we can derive the following expression.

$$\begin{aligned}
\mathcal{L}_{OUT}(O_{\alpha(k,l,r)}) &\overset{(\mathrm{Def.7.6})}{=} \mathcal{L}_{OUT}(O_{16k+r} + O_{16k+r+1} + O_{16l+r} + O_{16l+r+1}) \\
&\overset{(7.13)}{=} \mathcal{L}_{OUT}[\ F(O_{16k+r-1} + W[r-1] + Y(16k+r)) \\
&\qquad + F(O_{16k+r} \ + W[r] \ + \hat{Y}(16k+r)) \\
&\qquad + F(O_{16l+r-1} + W[r-1] + Y(16l+r)) \\
&\qquad + F(O_{16l+r} \ + W[r] \ + \hat{Y}(16l+r))] \\
&\overset{(\mathrm{Prop.7.3})}{=} \mathcal{L}_{IN}(O_{16k+r-1} + W[r-1] + Y(16k+r)) + N_\Sigma(\cdot) \\
&\qquad + \mathcal{L}_{IN}(O_{16k+r} \ + W[r] \ + \hat{Y}(16k+r)) + N_\Sigma(\cdot) \\
&\qquad + \mathcal{L}_{IN}(O_{16l+r-1} + W[r-1] + Y(16l+r)) + N_\Sigma(\cdot) \\
&\qquad + \mathcal{L}_{IN}(O_{16l+r} \ + W[r] \ + \hat{Y}(16l+r)) + N_\Sigma(\cdot) \\
&\overset{(\mathrm{Def.7.6})}{=} \mathcal{L}_{IN}(O_{\beta(k,l,r)}) + N^4_\Sigma((k,l,r)) \\
&\qquad + \mathcal{L}_{IN}(W[r] + W[r-1] + W[r] + W[r-1]) \\
&\qquad + \mathcal{L}_{IN}(Y(16k+r) + \hat{Y}(16k+r) + Y(16l+r) \\
&\qquad + \hat{Y}(16l+r)) \\
&\overset{(\mathcal{L}_{IN}(X+\hat{X})=0)}{=} \mathcal{L}_{IN}(O_{\beta(k,l,r)}) + N^4_\Sigma((k,l,r)),
\end{aligned} \tag{7.24}$$

where

$$\begin{aligned}
N^4_\Sigma((k,l,r)) =& N_\Sigma(O_{16k+r-1} + W[r-1] + Y(16k+r)) \\
&+ N_\Sigma(O_{16k+r} \ + W[r] \ + \hat{Y}(16k+r)) \\
&+ N_\Sigma(O_{16l+r-1} + W[r-1] + Y(16l+r)) \\
&+ N_\Sigma(O_{16l+r} \ + W[r] \ + \hat{Y}(16l+r)).
\end{aligned} \tag{7.25}$$

Corollary 7.4: For (k,l,r) satisfying (7.12) we have

$$\mathcal{L}_{OUT}(O_{\alpha(k,l,r)}) + \mathcal{L}_{IN}(O_{\beta(k,l,r)}) = N^4_\Sigma((k,l,r)). \tag{7.26}$$

□

The left hand side is a linear combination of output bytes and the right side is a linear combination of 24 noise variables (4 items by 6 biased noise variables each). These noise variables from the sum are dependent, since they are from the same source (Scream keystream generator). However,

their dependence is rather small, and we safely can treat them as independent random variables. Recall that the distribution of $N_\Sigma{}^4((k,l,r))$ is independent of (k,l,r). Let us denote this distribution by $P_{\mathcal{N}}$.

To conclude what we have done, define $L_1(\mathbf{O})$ to be the multiset of all samples possible to construct as above, i.e.,

$$L_1(\mathbf{O}) = \{\mathcal{L}_{OUT}(O_{\alpha(k,l,r)}) + \mathcal{L}_{IN}(O_{\beta(k,l,r)}) : (k,l,r) \text{ satisfy (7.12)}\}. \quad (7.27)$$

We summarize as follows.

Theorem 7.5: Under the assumptions made in Subsection 7.2.2, the samples in $L_1(\mathbf{O})$ are taken from the distribution $P_{\mathcal{N}}$. □

We have found how to calculate a set of samples $L_1(\mathbf{O})$ from the observed output stream $\mathbf{O} = O_0, O_1, \ldots$, and that these samples are drawn from the distribution $P_{\mathcal{N}}$.

Note that the distribution $P_{\mathcal{N}}$ depends on which linear approximation $R(x)$ we select for the S-box. Furthermore, for a known S-box mapping the $P_{\mathcal{N}}$ distribution can be easily calculated for any values $r_0, r_1, \ldots, r_7 \in \mathbb{F}_{2^8}$ chosen in the linear approximation of the S-box. However, the problem we face involves a secret S-box, and then we can not derive the distribution $P_{\mathcal{N}}$ explicitly, although, we can simulate it.

7.4 Simulations to Construct the Distinguisher

The general idea and the structure of the distinguisher for Scream is given in Definition 7.2. For a chosen $R(x)$, a linear approximation of the S-boxes in Scream, a type $P_{\mathbf{x}i}$ is constructed from the samples $L_1(\mathbf{O})$ according to the expression (7.27). We also need to establish how many samples we can get from t output words.

Proposition 7.6: The number of available triples (k,l,r) satisfying (7.12) in $O_0, O_1, \cdots, O_t$ is around $\frac{t \cdot 637}{256}$, when t is large.

Proof: According to the description of the "main" loop of Scream only one entry of the table $W[i_w]$ is updated after 16 output symbols are produced, the value of i_w is increased and taken modulo 16. It means that the distance between k^{th} and l^{th} rounds cannot be more than 16, otherwise sensitive entries of the table $W[\cdot]$ will be changed in between. Consider the situation when $r = 2$. To satisfy the equation (7.12) the first round index k cannot be equal to 1, 2, or 3, modulo 16, otherwise $W[r, r \pm 1]$ will be changed between the rounds. From the other hand, the second round index l can be equal to 1 modulo 16 (the value W[1] will be changed *after* the output symbols O_{16l+i} from this round are used in our formulas), however, it still cannot be

ϵ_0/λ	$P_{\max\,\vert K}(\epsilon_0/\lambda)$	ϵ_0/λ	$P_{\max\,\vert K}(\epsilon_0/\lambda)$
2^{-46}	0.001	2^{-53}	0.64
2^{-47}	0.009	2^{-54}	0.84
2^{-48}	0.017	2^{-55}	0.94
2^{-49}	0.038	2^{-56}	0.99
2^{-50}	0.097	2^{-57}	0.999
2^{-51}	0.19	2^{-58}	1.000
2^{-52}	0.39		

Table 7.1: Conditional probabilities $\Pr\{\max\limits_{R_1,\ldots,R_m} |P_{\mathcal{N}} - P_{\mathcal{U}}| \geq \epsilon_0/\lambda \mid K\}$ for different ϵ_0/λ, in average.

equal to 2 or 3 modulo 16. By counting, if $k \equiv 4 \mod 16$, then the possible values for l are $\{k+1, k+2, \ldots, k+13\}$ – 13 in total. If $k \equiv 5 \mod 16$ then l has 12 ways to be chosen. Finally, in one round of the index i_w we have $13 + 12 + 11 + \ldots + 1 = 91$ ways to choose the pair (k, l), and then the situation will be repeated. The same case happens for other values of r. Since r can be one out of 7 values, the number of possible triples (r, k, l) within one round of the index i_w is $7 \cdot 91 = 637$. In one round of the index i_w 256 output symbols O_i are produced, hence, the overall rate of the number of triples is $637 \cdot t/256$. Since we have to cut a few valid triples in the tails of accessible keystream, the real number of available triples is slightly less. □

The next step is to estimate the PDF for $\Pr\{\lambda \cdot \max\limits_{R_1,\ldots,R_m} |P_{\mathcal{N}} - P_{\mathcal{U}}| \geq \epsilon_0 \mid K\}$, i.e., the probability that the maximum distance between noise and uniform distributions among m different approximations R_i will not be less than $\epsilon_{\max}$, for some chosen threshold ϵ_0 and fixed λ. It is a conditional probability (conditioned on the secret key K), where ϵ_0 will be used later as the decision threshold. Let us denote this probability as

$$P_{\max\,\vert K}(\epsilon_0/\lambda) = \Pr\{\max_{R_1,\ldots,R_m} |P_{\mathcal{N}} - P_{\mathcal{U}}| \geq \epsilon_0/\lambda \mid K\}. \tag{7.28}$$

This distribution is constructed via simulations, that we performed in our work.

In one round of these simulations, for a fixed key K and many linear approximations R we calculate the distribution of $N(x)$ from (7.11). We then calculate the distribution $P_{\mathcal{N}}$ from (7.22) and (7.25), which is a linear combination of 24 different biased random variables distributed like $N(x)$.

We have tested around 2^{10} different randomly chosen keys, and for each of them we tried around $m = 2^{20}$ different randomly chosen linear approximations. The distribution of $N(x)$ is calculated according to (7.11),

Simulation scheme for estimating $P_{\max|K}(\epsilon_0/\lambda)$

I. **Infinite-loop**

II. Choose the secret key K randomly.

III. **Loop** $i = 1, \ldots, m$ **– for a sufficiently large** m

1) Choose an approximation $R(\cdot)$ randomly: $\{r_0, r_1, \ldots, r_7\}$, where $r_i \in \mathbb{F}_{2^8}$.

2) Make sure that $\forall x \in P_{\mathcal{U}} : R(x) \in P_{\mathcal{U}}$, otherwise, choose another approximation.

3) Construct the tables $S(x)$, $R(x)$, and $N(x)$, for $\forall x \in \mathbb{F}_{2^8}$.

4) Construct three 8 bit noise distributions (see (7.22) and (7.25))

$$\begin{aligned} P_{\mathcal{N}_1}(t) &= \Pr\{t = \alpha^2 \cdot (N(x) + N(x+c)), x \in P_{\mathcal{U}}\}, \\ P_{\mathcal{N}_2}(t) &= \Pr\{t = (\alpha^2 + \alpha + 1) \cdot N(x), x \in P_{\mathcal{U}}\}, \\ P_{\mathcal{N}_3}(t) &= \Pr\{t = N(x), x \in P_{\mathcal{U}}\}. \end{aligned} \tag{7.29}$$

5) Evaluate the 8 bit distribution for the sum of 24 noise variables from equation (7.25), which is exactly the same as $P_{\mathcal{N}} = (P_{\mathcal{N}_1} + P_{\mathcal{N}_2} + P_{\mathcal{N}_3})^{\times 8}$. The evaluation can be done logarithmically, and $P_{\mathcal{N}}$ is now the distribution of 24 introduced noise variables.

6) Calculate the distance $\epsilon_i = |P_{\mathcal{N}} - P_{\mathcal{U}}|$.

end-loop

IV. Calculate $\epsilon_{\max} = \max\{\epsilon_1, \epsilon_2, \ldots, \epsilon_m\}$ and attune the probability mass function $P_{\max|K}(\epsilon_0/\lambda)$.

V. **end-loop**

□

and then we calculate the distribution $P_{\mathcal{N}}$ from (7.22) and (7.25), which is a linear combination of 24 different biased random variables distributed like $N(x)$. In Table 7.1 we illustrate the following probability mass function for $P_{\max|K}(\epsilon_0/\lambda)$.

From Table 7.1 one can note a trade-off. From one hand we would like to choose the threshold ϵ_0 such that the probability rate is high. From another hand, according to Theorem 3.6, a small ϵ_0 means a large number of samples n required.

The remaining part is to estimate the necessary number of subdistinguishers m. During the simulations from Algorithm 7.4 we also calculated

the smallest value k such that $\max\{\epsilon_1, \ldots, \epsilon_k\} = \max\{\epsilon_1, \ldots, \epsilon_m\}$, which is the minimum number of subdistinguishers required to receive the maximum distance $|P_{\mathcal{N}} - P_{\mathcal{U}}|$, conditioned on K. This allowed us to estimate the probability $\Pr\{\text{maximum value is received} \mid m \text{ is fixed}\}$. Table 7.2 illustrates that if we take $m > 2^{20}$, then it will be useless, because the best linear approximation $R(\cdot)$ with the maximum possible distance $\epsilon = |P_{\mathcal{N}} - P_{\mathcal{U}}|$ will already be reached [5] . Therefore, we can safely set the number of subdistinguishers to be $m = 2^{20}$.

Number of subdistinguishers m	$\Pr\{$maximum distance value $\epsilon = \lvert P_{\mathcal{N}} - P_{\mathcal{U}}\rvert$ is reached$\}$
2^{10}	0.017
2^{11}	0.037
2^{12}	0.076
2^{13}	0.150
2^{14}	0.270
2^{15}	0.490
2^{16}	0.720
2^{17}	0.910
2^{18}	0.980
2^{19}	1.000
2^{20}	1.000

Table 7.2: Probabilities to reach the maximum distance $P_{\mathcal{N}} - P_{\mathcal{U}}$ for different values of m.

Formulas connecting parameters of the distinguisher for Scream depend on the choice of $\epsilon_{\max}$. If we assume that with probability $1 - P_{\max|K}(\epsilon_0/\lambda)$ we have a random distinguisher, then the actual advantage is then calculated as

$$\mathtt{Adv}_D = |1 - p_\alpha - p_\beta| \cdot P_{\max|K}(\epsilon_0/\lambda). \tag{7.30}$$

The typical values and the relation between the number of accessible samples and the advantage of the distinguisher for Scream are presented in Table 7.3. We can see that when around 2^{100} samples are available, the distinguisher has an advantage around 2^{-10}. Otherwise, when $n \approx 2^{120}$, the advantage is almost 1.

[5] The probability that the maximum distance will not appear before 2^{20} is not zero. However, we did not meet such a case during our tests.

When $m = 2^{20}$ subdistinguishers are used, and the desired values of the desired error probability p_α is fixed, whereas p_β remains negligible.							
$\epsilon_{\max}$	$P_{\max\|K}(\epsilon_0/\lambda)$	$p_\alpha < 2^{-10}$			$p_\alpha < 2^{-20}$		
		λ_{opt}	n	$\texttt{Adv}_D$	λ_{opt}	n	$\texttt{Adv}_D$
2^{-46}	0.001	0.8587	$2^{99.45}$	$2^{-9.9672}$	0.8112	$2^{99.6}$	$2^{-9.9658}$
2^{-48}	0.017		$2^{103.45}$	$2^{-5.8797}$		$2^{103.6}$	$2^{-5.8783}$
2^{-50}	0.097		$2^{107.45}$	$2^{-3.3673}$		$2^{107.6}$	$2^{-3.3659}$
2^{-52}	0.390		$2^{111.45}$	0.3896		$2^{111.6}$	0.3900
2^{-54}	0.840		$2^{115.45}$	0.8392		$2^{115.6}$	0.8400
2^{-56}	0.990		$2^{119.45}$	0.9890		$2^{119.6}$	0.9900
2^{-57}	0.999		$2^{121.45}$	0.9980		$2^{121.6}$	0.9990
2^{-58}	1.000		$2^{123.45}$	0.9990		$2^{123.6}$	1.0000

Table 7.3: Typical values for the advantage and the number of samples.

7.5 Computational Aspects

One may think that the complexity for one $\texttt{SD}_i$ is $O(n)$, and since we have m subdistinguishers, then the overall complexity for the distinguisher for Scream is $O(m \cdot n) = O(2^{20} \cdot n)$, which is quite large and the complexity is almost an exhaustive search. By the following computational approach we reduce $O(m \cdot n)$ to $O(n + m \cdot \text{const})$.

We observe that the expression (7.27) can be written in the form

$$\begin{aligned} |_{(k,l,r)} : \mathcal{L}_{OUT}(O_{\alpha(k,l,r)}) + \mathcal{L}_{IN}(O_{\beta(k,l,r)}) \\ = \theta_{k,l,r} + R(\eta_{k,l,r}) + (\alpha^2 + \alpha + 1) \cdot R(\vartheta_{k,l,r}), \end{aligned} \tag{7.31}$$

where $\eta_{k,l,r}, \theta_{k,l,r}, \vartheta_{k,l,r} \in \mathbb{F}_{2^8}$ are three bytes which can be calculated from the known output stream $\mathbf{O}$, for any time t.

We want to construct m different types from n samples, which are calculated from the same set of n triples $(\eta, \theta, \vartheta)$, but each time by applying different approximations $R(x)$. For this purpose, we first construct a table of the number of occurences for each triple $(\eta, \theta, \vartheta)$ in the sample set $L_1(\mathbf{O})$, denoted $T[\eta, \theta, \vartheta] = \{\textit{\#of occurences in the output stream}\}$. The table has the size $2^{3 \cdot 8} = 2^{24}$, all possible combinations of 3 bytes. Afterwards, for each approximation function $R(x)$ the corresponding type $P_{\mathbf{x}i}$ can be constructed directly from the table.

The complexity of calculating all types is $O(m \cdot 2^{24})$. The complexity to calculate the table $T[\eta, \theta, \vartheta]$ is $O(n)$. Therefore, the overall complexity is $O(n + m \cdot 2^{24})$.

Theorem 7.7: For the proposed Scream distinguisher (see Definition 7.2 and Theorem 7.5) under assumptions made in Subsection 7.2.2, in the case of using m subdistinguishers with different linear approximations of the S-box we get the overal time complexity around $O(n + m \cdot 2^{24})$. □

7.6 Improvements

In this section we give a few techniques for improving the attack.

7.6.1 Using a 16 bit Noise Construction

Consider the equations (7.16) and (7.18) – they describe byte relations "input-output" where the byte noise variables are introduced. By the previous technique we summed up these equations together and derived the equations (7.20), (7.21), (7.22), and (7.23). We just note that for the same triple (k, l, r), the input to these equations is from the same source, i.e., instead of summing up, we consider these two byte relations jointly, which potentially gives us more information. In this case, the noise variable is now a 16 bit random variable, constructed from $\binom{N_{0,8}}{N_{2,10}}$. We use again the expressions $O_{\alpha(k,l,r)}$ and $O_{\beta(k,l,r)}$ in order to eliminate the $F(\cdot)$-function application.

The 16 bit model can now be constructed directly

Proposition 7.8: From the expressions (7.16) and (7.18), for any 16-byte vector $X \in \mathbb{F}_{2^8}^{16}$ the following equation holds

$$ {}_{16}\mathcal{L}_{OUT}(F(X)) = {}_{16}\mathcal{L}_{IN}(X) + {}_{16}N_{\Sigma}(X), \tag{7.32} $$

where

$$ {}_{16}\mathcal{L}_{OUT}(F(X)) = \begin{bmatrix} \frac{1}{1+\alpha} \cdot (x'_0 + \alpha \cdot x'_1 + x'_8 + \alpha \cdot x'_9) \\ x'_2 + (1+\alpha) \cdot x'_3 + x'_{10} + (1+\alpha) \cdot x'_{11} \end{bmatrix}; \tag{7.33} $$

$$ {}_{16}\mathcal{L}_{IN}(X) = \begin{bmatrix} R(x_2 + x_{10}) + R''(x_0 + x_8) + R'(x_5 + x_{13}) \\ (1 + \alpha + \alpha^2) \cdot R(x_0 + x_8) + \alpha^2 R''(x_0 + x_8) + R(x_5 + x_{13}) \\ +\alpha^2 \cdot R((1+\alpha)R(x_5 + x_{13})) + \alpha^2 \cdot R(x_2 + x_{10}) \end{bmatrix}; \tag{7.34} $$

$$ \begin{aligned} {}_{16}N_{\Sigma}(X) = &\begin{bmatrix} N(v_2 + c) \\ \alpha^2 \cdot N(v_2) \end{bmatrix} + \begin{bmatrix} R(N(x_0)) \\ (1 + \alpha + \alpha^2) \cdot N(x_0 + c) + \alpha^2 \cdot R(N(x_0)) \end{bmatrix} \\ &+ \begin{bmatrix} R((1+\alpha) \cdot N(x_5)) \\ N(x_5 + c) + \alpha^2 \cdot R((1+\alpha) \cdot N(x_5)) \end{bmatrix} \\ &+ \begin{bmatrix} N(v_{10} + c) \\ \alpha^2 \cdot N(v_{10}) \end{bmatrix} + \begin{bmatrix} R(N(x_8)) \\ (1 + \alpha + \alpha^2) \cdot N(x_8 + c) + \alpha^2 \cdot R(N(x_8)) \end{bmatrix} \\ &+ \begin{bmatrix} R((1+\alpha) \cdot N(x_{13})) \\ N(x_{13} + c) + \alpha^2 \cdot R((1+\alpha) \cdot N(x_{13})) \end{bmatrix}, \end{aligned} \tag{7.35} $$

where $c = 00010101_2$ is the constant, and ${}_{16}N_{\Sigma}(X)$ is a linear combination of 6 16 bit noise variables, represented through the 8 bit noise $N(x)$ introduced in (7.11). □

Similar to Corollary 7.4 we derive a 16 bit relation betwen the noise and output symbols, and the expression (7.32) can be applied for the output stream **O** without calling the unknown $F(\cdot)$-function.

Corollary 7.9: For any triple (k, l, r) satisfying (7.12) we have

$$ {}_{16}\mathcal{L}_{OUT}(O_{\alpha(k,l,r)}) + {}_{16}\mathcal{L}_{IN}(O_{\beta(k,l,r)}) = {}_{16}N^4_{\Sigma}((k,l,r)), \tag{7.36}$$

where

$$\begin{aligned} {}_{16}N^4_{\Sigma}((k,l,r)) =& {}_{16}N_{\Sigma}(O_{16k+r-1} + W[r-1] + Y(16k+r)) \\ &+ {}_{16}N_{\Sigma}(O_{16k+r} \quad + W[r] \quad + \hat{Y}(16k+r)) \\ &+ {}_{16}N_{\Sigma}(O_{16l+r-1} + W[r-1] + Y(16l+r)) \\ &+ {}_{16}N_{\Sigma}(O_{16l+r} \quad + W[r] \quad + \hat{Y}(16l+r)). \end{aligned} \tag{7.37}$$

□

The left part is a linear combination of the output bytes, whereas the right side is the sum of 24 16-bit noise variables. Let us define the multiset of 16 bit samples as:

$$ {}_{16}L_1(\mathbf{O}) = \{ {}_{16}\mathcal{L}_{OUT}(O_{\alpha(k,l,r)}) + {}_{16}\mathcal{L}_{IN}(O_{\beta(k,l,r)}) : \ (k,l,r) \text{ satisfy (7.12)} \}. \tag{7.38}$$

The distribution of ${}_{16}N^4_{\Sigma}((k,l,r))$ is independent on (k,l,r), and we denote this distribution as ${}_{16}P_{\mathcal{N}}$.

Theorem 7.10: The 16 bit samples ${}_{16}L_1(\mathbf{O})$ are taken from the distribution ${}_{16}P_{\mathcal{N}}$. □

We have just found the way to sample 16 bit random variables from the given output stream $\mathbf{O} = O_1, O_2, \ldots$, and these samples are from the biased (unknown) distribution ${}_{16}P_{\mathcal{N}}$. Then we can perform the attack on Scream in a similar way, as described in Section 7.4.

7.6.2 Using Several Linear Approximations R_j in Parallel

Consider again the byte relation found in the previous sections and given by the formula (7.26). When we calculate a byte-sample, we can use the formula (7.31). Assume we choose three linear approximations $R_1(x), R_2(x),$

$R_3(x)$ of the $S(\cdot)$-boxes, and we now wish to consider three byte-samples jointly, according to these three approximations, i.e.:

$$
\begin{aligned}
{}_{3\times 8}L_1(\mathbf{O}) &= \begin{pmatrix} \mathcal{L}_{OUT}(O_{\alpha(k,l,r)}) + \mathcal{L}_{IN}(O_{\beta(k,l,r)})|_{R=R_1} \\ \mathcal{L}_{OUT}(O_{\alpha(k,l,r)}) + \mathcal{L}_{IN}(O_{\beta(k,l,r)})|_{R=R_2} \\ \mathcal{L}_{OUT}(O_{\alpha(k,l,r)}) + \mathcal{L}_{IN}(O_{\beta(k,l,r)})|_{R=R_3} \end{pmatrix} \\
&= \begin{pmatrix} {N_\Sigma}^4((k,l,r))|_{R=R_1} \\ {N_\Sigma}^4((k,l,r))|_{R=R_2} \\ {N_\Sigma}^4((k,l,r))|_{R=R_3} \end{pmatrix} = {}_{3\times 8}N_\Sigma^4((k,l,r)). \qquad (7.39)
\end{aligned}
$$

All the three samples are taken from the same source at the same time, but only the approximations are different. For a random source, the distribution of this triple vector (a 24 bit sample) would look like

$$
\begin{pmatrix} x + R_1(y) + (\alpha^2+\alpha+1)R_1(z) \\ x + R_2(y) + (\alpha^2+\alpha+1)R_2(z) \\ x + R_3(y) + (\alpha^2+\alpha+1)R_3(z) \end{pmatrix}, \quad \text{for } x,y,z \in P_{\mathcal{U}}. \qquad (7.40)
$$

For an appropriate choice of the approximations, this distribution is again the uniform distribution. So, further we consider only such triples $(R_1(\cdot), R_2(\cdot), R_3(\cdot))$, for which the random distribution of this triple vector is uniform (for simplicity purposes).

The advantage of using these different approximation functions in parallel is that we can extract more information from the stream produced by Scream, and our simulations gave us improved results.

One more improvement can be to consider multiple approximations in parallel, but for the case of 16 bit sampling, as described in the previous subsection. We only mention this idea, but we did not implement it due to very high computational complexity.

7.6.3 Simulation Results for the Improved Versions

Due to a very high computational complexity we could not perform many simulations for the attack with improvements described in the previous subsections (16 bit and 24 bit cases) and create the distribution for $\Pr\{\lambda \cdot \max_{R_1,\ldots,R_m} |P_{\mathcal{N}} - P_{\mathcal{U}}| \geq \epsilon_0 \mid K\}$ as we did before for the 8 bit case (see Table 7.1). However, we could perform just a few simulations to be able to see the advantage of the improvements, and compare all the described techniques. Here we present one comparison result produced by our simulation program.

Key = D2 A6 C5 C1 30 6A 1A 22 2C 5B D7 7B B8 31 26 58			
Approximations	8 bit	16 bit	Triple×8
$A_0 =$ 77 6E 4A A1 4C EA 22 74	$2^{-65.000}$	$2^{-62.425}$	$2^{-51.396}$
$A_1 =$ 45 15 21 B6 39 97 B9 CF	$2^{-63.641}$	$2^{-63.359}$	
$A_2 =$ 80 A3 B8 2F 17 28 83 F9	$2^{-62.425}$	$2^{-62.397}$	

This output table has the values R_i for three different approximations A_0, A_1, and A_2, and the key K. In the table the distances $|P_{\mathcal{N}} - P_{\mathcal{U}}|$ are shown for three different attack techniques with the same approximations (the triple approximation technique uses the A_0, A_1, and A_2 approximations).

The larger the bias gets, the less number of samples we need, i.e., the time complexity for the attack is low if the bias is large. Obviously, the approach with a triple approximation gives us the reduction of the number of subdistinguishers. An interesting question is whether this improvement can give significantly larger bias than our byte oriented attack or not. However, due to a very high computational complexity of these simulations, we could not give a definite answer on this question.

7.7 Summary

In this paper we have derived a linear distinguishing attack on Scream, and also suggested some improvements. The final distinguisher is presented in Figure 7.4. In this type of attack we approximate nonlinear parts, and introduce corresponding noise variables. When all the operations are made linear using linear approximations, it is possible to find some linear equation(s) on the output words. One side of that equation contains a linear combination of the output words, whereas the second side contains a linear equation of introduced noise variables. If these noise variables are biased, then a distinguisher can be built.

In the case of Scream, the only the nonlinear parts are two byte-oriented secret S-boxes. In our work we show how to use an approximation over the larger field $\mathbb{F}_{2^8}$, in particular, we write $S(x) = R(x) + N(x)$, where $R(x) = \sum_{i=0}^{7} r_i x^{2^i}$, $r_i \in \mathbb{F}_{2^8}$ and $N(x)$ is a noise variable. The distribution of the byte-oriented noise variable $N(x)$ is possible to calculate only if the S-box and $R(x)$ are known. The problem with Scream was that it has *secret* S-boxes, i.e., they are initialised with the secret key, and the effect (bias) of a choice of the function $R(x)$ is not possible to predict. One choice of R could give us a good bias of the noise variable N, and another choice can be not a good one.

However, we could treat the case when the distributions of the noise variables are unknown. To overcome this problem we suggest to build many similar subdistinguishers, but with different linear approximations. We can choose the number of subdistinguishers $m = 2^{20}$ such that with a very high probability (in our simulations this probability is 1) one of the approximations reaches the largest distance $\epsilon_{\max} = \epsilon_0/\lambda$ between the noise and the uniform distributions for a fixed secret key. This distance and the probability of error p_e of each subdistinguisher influent on the required length n of accessible keystream.

In Section 7.6 we consider two general ways to improve the attack. Actually, in our work we could manage to find two relations on the same output words. *The first idea* is to use these two different byte relations jointly, which theoretically should extract more information about the given system. In this case the noise variable is a 16 bit random variable. Our *second idea* is based on the equation (7.31), where the linear combination of the known part can be represented as a linear combination of the $R(x)$ function instances. We suggest to choose three different approximation functions, and consider a 24 bit symbol derived from the output stream, as shown in (7.40). If the choice of these approximations $R_i(x)$ is proper then in the random case this 24 bit symbol is from the uniform distribution. The corresponding 24 bit noise variable will have a stronger bias.

By simulations, we estimated the advantage of our distinguisher. For a given keystream of length 2^{100} there is a detectable advantage. When the keystream length is 2^{120}, the advantage of our distinguisher is very close to 1. By this we have shown that Scream can be distinguished faster than an exhaustive search. In our work we made simulations for the proposed 8 bit oriented case of the attack, and also show the effect of the improvements from Section 7.6. We believe that this kind of technique can be successfully applied to other ciphers.

Distinguisher:

1. Choose the number of subdistinguishers $m = 2^{20}$.
   The number of necessary samples is $n \cong 2^{7.45}/\epsilon_{\max}^2$, where $\epsilon_{\text{thr}} = 0.85\epsilon_{\max}$ for some fixed $\epsilon_{\max} \in [2^{-46} \ldots 2^{-58}]$. The advantage of the distinguisher is according to Table 7.3.
2. *Create data structures*
   - $T[0..255, 0..255, 0..255] \in \{0, 1, \ldots 255\}$ – the 3-dimensional matrix
   - $P[0..2^{20} - 1, 0..255] \in [0..1]$ – real valued probability table
   - $A[0..2^{20} - 1, 0..7] \in \{0, 1, \ldots 255\}$ – the coefficients of subdistinguishers
3. *Initialization*
   - $\forall x, y, z \in [0..255] \Rightarrow T[x, y, z] = 0$;
   - $\forall i \in [0..2^{16} - 1], j \in [0..255] \Rightarrow P[i, j] = 0.0$;
   - $\forall i \in [0..2^{16} - 1], j \in [0..7] \Rightarrow A[i][j] =$random value from $\{0, 1, \ldots 255\}$
4. *Calculation of the table* $T[x, y, z]$
   for $i = 1$ to n
   4.1 take i^{th} valid triple (k_i, l_i, r_i) satisfying (7.12);
   4.2 calculate 3 bytes: $b_1 = \theta_{k_i,l_i,r_i}, b_2 = \eta_{k_i,l_i,r_i}$ and $b_3 = \vartheta_{k_i,l_i,r_i}$ in the notation of the equation (7.31);
   4.3 $T[b_1, b_2, b_3] = T[b_1, b_2, b_3] + 1$;
   end for i;
5. *Calculate the probability table*
   for $b_1 = 0$ to 255
   for $b_2 = 0$ to 255
   for $b_3 = 0$ to 255
   5.1 for $i = 0$ to $2^{20} - 1$
   5.2 represent b_1, b_2, b_3 as elements from the field $\mathbb{F}_{2^8}$ with generating polynomial $g(x) = x^8 + x^7 + x^6 + x + 1$ and calculate the byte $\mathcal{B} = b_1 + R(b_2) + (\alpha^2 + \alpha + 1)R(b_3)$, where $R(x) = \sum_{j=0}^{7}(A[n, j] \cdot x^{2^j})$;
   5.3 Set $P[i, \mathcal{B}] = P[i, \mathcal{B}] + \frac{T[b_1,b_2,b_3]}{n}$;
   5.4 end for i;
   5.5. end for b_3, b_2, b_1;
6. *Calculate the distances for each subdistinguisher and make the final decision*
   6.1. for $i = 0$ to $2^{20} - 1$
   6.2. $\epsilon = \sum_{j=0}^{255} |P[i, j] - 2^{-8}|$
   6.3. if $\epsilon \geq \epsilon_0$ then $\texttt{SD}_i(\mathbf{O}) =$ “`OUTSIDE`” and *the final decision* is **A: the keystream is from Scream**. Stop the process.
   6.4. end for i;
7. All $\texttt{SD}_i(\mathbf{O}) =$ “`INSIDE`” and *the final decision* is **B: the keystream is Random**

Figure 7.4: Distinguisher for Scream.

8

Cryptanalysis of the "Grain" Family of Stream Ciphers

"Anything will give up its secrets if you love it enough. Not only have I found that when I talk to the little flower or to the little peanut they will give up their secrets, but I have found that when I silently commune with people they give up their secrets also – if you love them enough"

George Washington Carver

Recently, a new European project eSTREAM [ECR05] has started, and at the first stage of the project 35 new proposals were received by May 2005. Although many previous stream ciphers were broken, collected cryptanalysis experience allowed to strengthen new proposals significantly, and there are many of them that are strong against different kinds of attacks. One such good proposal was the new stream cipher Grain

The stream cipher Grain was developed by a group of researchers M. Hell, T. Johansson, and W. Meier, and was especially designed for being very small and fast in hardware implementation. It uses the key of length 80 bits and the IV is 64 bits, its internal state is of size 160 bits. Grain uses a *nonlinear feedback shift register* (NLFSR) and a *linear feedback shift register* (LFSR), and

the idea to use NLFSR is quite new in modern cryptography. The claimed security level of Grain is 2^{80}, and it was stated that there exist no attacks significantly faster than 2^{80}. When a key-recovering attack is considered, the idea behind the strength of Grain is amazingly based on the well-known *general decoding problem* (GDP), which is hard to solve. To use GDP in the design of a stream cipher was a quite clever idea, and it was also discussed before in, e.g. [JJ02]. Therefore, instead of analysing just Grain, in this chapter we study this design structure in general, and we call this class of stream ciphers as *the "Grain" family of stream ciphers*.

In this chapter we focus on statistical properties of the keystream, revealing two key-recovering and one distinguishing attack on Grain. We show possible weaknesses, if the Boolean functions used in the design are chosen improperly. We also show the relation between the strength of "Grain" against a key-recovery attack and the general decoding problem. For the proposed instance Grain [HJM05a] we found a statistical leakage in the keystream, which allowed us to mount a distinguishing attack with time complexity $O(2^{54})$, when keystream of length $O(2^{51})$ is available. This attack is definitely significantly faster than $O(2^{80})$. Moreover, we also present a key recovery attack against Grain which requires 2^{43} computations and 2^{38} keystream bits to determine the 80 bit key.

This chapter is structured as follows. In Section 8.1 the "Grain" family of stream ciphers is defined. In Section 8.2 the correlation between the keystream and the state of the LFSR is derived. In Section 8.4 we use the weakness of the Grain instance to turn it into a distinguishing attack. We show that the security level of this class of stream ciphers is related to the general decoding problem in Section 8.3. We show how to use fast Fourier techniques to recover the state of the LFSR and the NFSL in Sections 8.5 and 8.6. Finally, we summarize the results and make the conclusions in Section 8.8.

8.1 The "Grain" Family of Stream Ciphers

The "Grain" family of stream ciphers is a bit-oriented design, and its general structure is depicted in Figure 8.1. Let the LFSR have length l and its generating polynomial is $f(\cdot) = 0$. Its output is denoted as $\mathbf{y} = y_0, y_1, \ldots$, where the first l bits is the initial state of the LFSR. Let the length of the NLFSR be m. The feedback function $g(\cdot) = 0$ for the NLFSR is a Boolean function on the states of the NLFSR and the LFSR. We denote the output of the NLFSR as $\mathbf{x} = x_0, x_1, \ldots$, where the first m bits is the initial state of the NLFSR. The keystream sequence is $\mathbf{z} = z_0, z_1, \ldots$. At each time instance t, one bit of the keystream z_t is the result of a Boolean function $h(\cdot)$, the input of which are the bits from the states of the NLFSR and the LFSR, at the corresponding

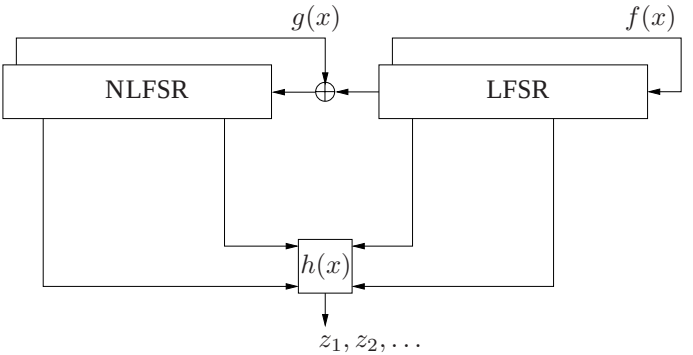


Figure 8.1: The structure of the “Grain” family of stream ciphers.

time t.

For some arbitrary Boolean function $\delta(\cdot)$ let $w_L(\delta(\cdot))$ and $w_N(\delta(\cdot))$ denote the number of operands, taken from the LFSR and from the NLFSR, respectively. The total number of variables for the function $\delta(\cdot)$ is denoted as $w(\delta(\cdot))$.

For the particular instance of Grain, let the current LFSR content be denoted by $Y_t = (y_t, y_{t+1}, \ldots, y_{t+79})$. The LFSR is governed by the linear recurrence

$$y_{t+80} = y_{t+62} \oplus y_{t+51} \oplus y_{t+38} \oplus y_{t+23} \oplus y_{t+13} \oplus y_t. \tag{8.1}$$

Let the current NLFSR content be denoted by $X_t = (x_t, x_{t+1}, \ldots, x_{t+79})$. The NLFSR feedback is disturbed by the output of the LFSR, so that the NLFSR content is governed by the recurrence

$$x_{t+80} = y_t \oplus g(x_t, x_{t+1}, \ldots, x_{t+79}), \tag{8.2}$$

where the expression of nonlinear feedback function g is given by

$$\begin{aligned} g :=& x_{t+63} \oplus x_{t+60} \oplus x_{t+52} \oplus x_{t+45} \oplus x_{t+37} \oplus x_{t+33} \oplus x_{t+28} \oplus x_{t+21} \oplus x_{t+15} \\ & \oplus x_{t+9} \oplus x_t \oplus x_{t+63}x_{t+60} \oplus x_{t+37}x_{t+33} \oplus x_{t+15}x_{t+9} \oplus x_{t+60}x_{t+52}x_{t+45} \\ & \oplus x_{t+33}x_{t+28}x_{t+21} \oplus x_{t+63}x_{t+45}x_{t+28}x_{t+9} \oplus x_{t+60}x_{t+52}x_{t+37}x_{t+33} \\ & \oplus x_{t+63}x_{t+60}x_{t+21}x_{t+15} \oplus x_{t+63}x_{t+60}x_{t+52}x_{t+45}x_{t+37} \\ & \oplus x_{t+33}x_{t+28}x_{t+21}x_{t+15}x_{t+9} \oplus x_{t+52}x_{t+45}x_{t+37}x_{t+33}x_{t+28}x_{t+21}. \end{aligned} \tag{8.3}$$

The cipher output bit z_t is derived from the current LFSR and NLFSR states as the exclusive or of the masking bit x_t and a nonlinear filtering func-

tion h as follows

$$\begin{aligned} z_t &= h(y_{t+3}, y_{t+25}, y_{t+46}, y_{t+64}, x_t, x_{t+63}) \\ &= x_t \oplus h'(y_{t+3}, y_{t+25}, y_{t+46}, y_{t+64}, x_{t+63}) \\ &= x_t \oplus x_{t+63} p_t \oplus q_t, \end{aligned} \tag{8.4}$$

where p_t and q_t are the functions of $y_{t+3}, y_{t+25}, y_{t+46}, y_{t+64}$ given by

$$\begin{aligned} p_t &= 1 \oplus y_{t+64} \oplus y_{t+46}(y_{t+3} \oplus y_{t+25} \oplus y_{t+64}), \\ q_t &= y_{t+25} \oplus y_{t+3}y_{t+46}(y_{t+25} \oplus y_{t+64}) \oplus y_{t+64}(y_{t+3} \oplus y_{t+46}). \end{aligned} \tag{8.5}$$

The Boolean function h' is correlation immune of the first order. As noticed in [HJM05a], "*this does not preclude that there are correlations of the output of $h(\cdot)$ to sums of inputs*", but the designers of Grain appear to have expected the NLFSR masking bit x_t to make it impractical to exploit such correlations.

The key and IV setup consists of loading the key bits in the NLFSR, loading the 64 bit IV followed by 16 ones in the LFSR, and clocking the cipher 160 times in a special mode where the output bit is fed back into the LFSR and the NLFSR. Once the key and IV have been loaded, the keystream generation mode described above is activated and the keystream sequence $\mathbf{z}$ is produced.

8.2 Deriving Linear Approximations of the LFSR Bits

8.2.1 Linear Approximations Used to Derive the LFSR Bits

The purpose of the attack is, based on a keystream sequence $(z_t)_{t=0\ldots n-1}$ corresponding to an unknown key K and a known IV value, to recover the key K. The initial step of the attack is to derive a sufficient number L of linear approximation equations involving the $l = 80$ bits of the initial LFSR state $Y_0 = (y_0, \ldots, y_{79})$ (or equivalently a sufficient number L of linear approximation equations involving bits of the sequence $\mathbf{y}^n$) to recover the value of Y_0. Hereafter, as will be shown in Section 8.6, the initial NLFSR state X_0 and the key K can then be easily recovered.

The starting point for the attack consists in noticing that though the NLFSR feedback function g is balanced, the function g' given by $g'(X_t) = g(X_t) \oplus x_t$ is unbalanced. We have

$$\Pr\{g'(X_t) = 1\} = \frac{522}{1024} = \frac{1}{2} + \epsilon_{g'}, \tag{8.6}$$

where $\epsilon_{g'} = \frac{5}{512}$. It is useful to notice that the restriction of g' to input values X_t such that $x_{t+63} = 0$ is totally balanced and that the imbalance of

the function g' is exclusively due to the imbalance of the restriction of g' to input values X_t such that $x_{t+63} = 1$.

If one considers one single output bit z_t, the involvement of the masking bit x_t in the expression of z_t makes it impossible to write any useful approximate relation involving only the Y_t bits. But if one considers the sum $z_t \oplus z_{t+80}$ of two keystream bits output at a time interval equal to the NLFSR length $n = 80$, the $x_t \oplus x_{t+80}$ contribution of the corresponding masking bits is equal to $g'(X_t) \oplus y_t$, and is therefore equal to y_t with probability $\frac{1}{2} + \epsilon_{g'}$. As for the other terms of $z_t \oplus z_{t+80}$, they can be approximated by linear functions of the bits of the sequence $\mathbf{y}$. In more details,

$$\begin{aligned} z_t \oplus z_{t+80} =& g'(X_t) \oplus y_t \oplus h(y_{t+3}, y_{t+25}, y_{t+46}, y_{t+64}, x_{t+63}) \\ & \oplus h(y_{t+83}, y_{t+105}, y_{t+126}, y_{t+144}, x_{t+143}). \end{aligned} \tag{8.7}$$

Since the restriction of $g'(X_t)$ to input values such that $x_{t+63} = 0$ is balanced, we can restrict our search to linear approximations of the term $h'(y_{t+3}, y_{t+25}, y_{t+46}, y_{t+64}, x_{t+63})$ to input values such that $x_{t+63} = 1$, which amounts to finding linear approximations of $p_t \oplus q_t$.

We found the set containing two the best linear approximations for this function, namely

$$\begin{aligned} \mathcal{L}_1 = \{&y_3 \oplus y_{25} \oplus y_{64} \oplus 1; \\ &y_{25} \oplus y_{46} \oplus y_{64} \oplus 1\}. \end{aligned} \tag{8.8}$$

Each of the approximations of L_1 is valid with a probability $\frac{1}{2} + \epsilon_1$, where $\epsilon_1 = \frac{1}{4}$.

Now the term $h'(y_{t+83}, y_{t+105}, y_{t+126}, y_{t+144}, x_{t+143})$ is equal to either $p_{t+80} \oplus q_{t+80}$ or q_{t+80}, with a probability $\frac{1}{2}$ for both expressions. We found the set of the 8 best simultaneous linear approximations for these two expressions, namely

$$\begin{aligned} \mathcal{L}_2 = \{&y_{t+83} \oplus y_{t+144} \oplus 1; \\ &y_{t+83} \oplus y_{t+126} \oplus y_{t+144}; \\ &y_{t+83} \oplus y_{t+105}; \\ &y_{t+83} \oplus y_{t+105} \oplus y_{t+126}; \\ &y_{t+83} \oplus y_{t+105} \oplus y_{t+126} \oplus y_{t+144} \oplus 1; \\ &y_{t+83} \oplus y_{t+105} \oplus y_{t+144} \oplus 1; \\ &y_{t+105} \oplus y_{t+144}; \\ &y_{t+105} \oplus y_{t+126} \oplus y_{t+144} \oplus 1\}. \end{aligned} \tag{8.9}$$

Each of the 8 approximations of $\mathcal{L}_2$ has an average probability $\epsilon_2 = \frac{1}{8}$ of being valid.

Thus, we have found 16 linear approximations of $z_t \oplus z_{t+80}$, namely all the linear expressions of the form

$$y_t \oplus l_1(y_{t+3}, y_{t+25}, y_{t+46}, y_{t+64}) \oplus l_2(y_{t+83}, y_{t+105}, y_{t+126}, y_{t+144}), \quad (8.10)$$

where $l_1 \in \mathcal{L}_1$ and $l_2 \in \mathcal{L}_2$. Each of these approximations is valid with a probability $\frac{1}{2} + \epsilon$, where ϵ is derived from $\epsilon_{g'}$, ϵ_1, and ϵ_2 using the Piling-up Lemma 3.5:

$$\epsilon = \frac{1}{2} \cdot 2^2 \cdot \epsilon_{g'} \cdot \epsilon_1 \cdot \epsilon_2 = \frac{5}{4096} \simeq 2^{-9.67}. \quad (8.11)$$

The extra multiplicative factor of $\frac{1}{2}$ takes into account the fact that the considered approximations are only valid when $x_{t+63} = 1$. The LFSR derivation attacks of Section 8.5 exploit these 16 linear approximations.

8.2.2 Generalisation of the Attack Method

In this section, we try to generalise the previous approximation method. The purpose is not to find better approximations than those identified in Section 8.2.1, but to derive some design criteria on the Boolean functions g and h. However in the previous approximation, we used the fact that the bias of g depends on the value of x_{t+63}, so that the approximations of g and h are not correct independently. We do not take this phenomenon into account in this section. Therefore, we only provide a simplified picture of a potential generalised attack.

The function $g(X_t, Y_t)$ operates on $w(g) = w_L(g) + w_N(g)$ variables taken from the LFSR and the NLFSR, where $w_L(g)$ is the number of variables taken from the LFSR and $w_N(g)$ the number of variables taken from the NLFSR. Let the function $A_g(X_t, Y_t)$ be a linear approximation of the function g, i.e.,

$$A_g(X_t, Y_t) = \bigoplus_{i=0}^{w_N(g)-1} d_i x_{t+\phi_g(i)} \oplus \bigoplus_{j=0}^{w_L(g)-1} c_j y_{t+\psi_g(j)}, \quad c_j, d_i \in \mathbb{F}_2, \quad (8.12)$$

such that the distance between $g(\cdot)$ and $A_g(\cdot)$ defined by

$$d_g = \#\{x \in \mathbb{F}_2^{w(g)} : A_g(x) \neq g(x)\} > 0, \quad (8.13)$$

is strictly larger than zero. Then, we have

$$\Pr\{A_g(x) \neq g(x)\} = \frac{1}{2^{w(g)}} d_g, \quad (8.14)$$

i.e.

$$\Pr\{A_g(x) + g(x) = 0\} = 1/2 + \epsilon_g, \quad (8.15)$$

where the bias is

$$\epsilon_g = 1/2 - 2^{-w(g)} d_g. \tag{8.16}$$

Similarly, the function $h(X_t, Y_t)$ can also be approximated by some linear expressions of the form

$$A_h(X_t, Y_t) = \bigoplus_{i=0}^{w_N(h)-1} k_i x_{t+\phi_h(i)} \oplus \bigoplus_{j=0}^{w_L(h)-1} l_j y_{t+\psi_h(j)}, \quad k_j, l_i \in \mathbb{F}_2. \tag{8.17}$$

Recall, $z_t \stackrel{p}{=} A_h(\cdot)_t$ with some probability p. Having the expressions (8.12) and (8.17), one can sum up together $w_N(A_g(\cdot))$ expressions of $A_h(\cdot)$ at different times t, in such a way that all terms X_t will be eliminated (just because the terms X_t will be cancelled due to the parity check function $A_g(\cdot)$, leaving the terms Y_t and noise variables only). Note also that any linear combination of $A_h(\cdot)$ is a linear combination of the keystream bits z_t.

The sum of $w_N(A_g(\cdot))$ approximations $A_h(\cdot)$ will introduce $w_N(A_g(\cdot))$ independent noise variables due to the approximation at different time instances. Moreover, the cancellation of the terms X_t in the sum will be done by the parity check property of the approximation $A_g(\cdot)$. If the function $A_h(\cdot)$ contains $w_N(A_h)$ terms from X_t, then the parity cancellation expression $A_g(\cdot)$ will be applied $w_N(A_h)$ times. Each application of the cancellation expression $A_g(\cdot)$ will introduce another noise variable due to the approximation $n_g : g(\cdot) \to A_g(\cdot)$. Therefore, the application of the expression $A_g(\cdot)$ $w_N(A_h)$ times will introduce $w_N(A_h)$ additional noise variables n_g. Accumulating all above and following the Piling-up Lemma, the final correlation of such a sum (of the linear expression on Y_t) is given by the following Theorem.

Theorem 8.1: There always exists a linear relation in terms of bits from the state of the LFSR and the keystream, which have the bias:

$$\epsilon = 2^{(w_N(A_h)+w_N(A_g)-1)} \cdot \epsilon_g^{w_N(A_h)} \cdot \epsilon_h^{w_N(A_g)}, \tag{8.18}$$

where $A_g(\cdot)$ and $A_h(\cdot)$ are linear approximations of the functions $g(\cdot)$ and $h(\cdot)$, respectively, and:

$$\Pr\{A_g(\cdot) = g(\cdot)\} = 1/2 + \epsilon_g, \qquad \Pr\{A_h(\cdot) = h(\cdot)\} = 1/2 + \epsilon_h. \tag{8.19}$$

This theorem gives us a criteria for a proper choice of the functions $g(\cdot)$ and $h(\cdot)$. The biases ϵ_g and ϵ_h are related to the *nonlinearity* of these Boolean functions, and the values $w_N(A_g)$ and $w_N(A_h)$ are related to the *correlation immunity* property; however, there is a well-known trade-off between these two properties [Sie84]. Unfortunately, in the case of Grain the functions $g(\cdot)$ and $h(\cdot)$ were improperly chosen.

8.3 Relation to the General Decoding Problem

The GDP was explicitly discussed in Section 3.5.3. Recall the equation 3.63, i.e., the initial state of the stream cipher Grain could theoretically be recovered if $n \approx 18604655 \approx 2^{24.2}$ bits of the keystream are available. It has also been shown that the decoding problem is hard. However, for some special cases different decoding algorithms can be applied, that can recover the initial state much faster than exhaustive search. These techniques are usually called *fast correlation attacks*, some of them are introduced in, e.g. [JJ99a, JJ00, CJS00, MFI02, CJM02], and other literature.

In this section we show that cryptanalysis of the "Grain" family of stream ciphers can easily be converted to the general decoding problem, with the same bias, the one derived in the previous section.

Let $Y_t = (y_t, \ldots, y_{t+n-1})$ be the state of the LFSR at time instance t. I.e., the initial state is Y_0. Any state Y_t can be expressed via the initial state Y_0 by the multiplication with some $l \times l$ matrix A several times (see Section 2.5.3) as follows

$$Y_t = Y_0 \times A^t. \tag{8.20}$$

In the previous section we have shown how to find a linear relation between the keystream bits $z_t, z_{t+1}, \ldots$, and the output bits produces from the LFSR $y_t, y_{t+1}, \ldots$, for any time instance t. This relation u_t can also be expressed as

$$u_t = \sum_{i \in \mathcal{B}} z_{t+i} \stackrel{p}{=} \sum_{j \in \mathcal{A}} y_{t+j}, \tag{8.21}$$

where $\mathcal{A}$ and $\mathcal{B}$ are some sets of indices. Thus, $\mathbf{u}$ is another stream directly derived from $\mathbf{z}$. We apply (8.20) to derive the relation to the decoding problem as follows

$$\begin{aligned} U_t = (u_t, \ldots, u_{t+n-1}) &\stackrel{p}{=} \sum_{j \in \mathcal{A}} Y_0 \times A^{t+j} \\ &= (Y_0 \times \sum_{j \in \mathcal{A}} A^j) \times A^t = Q_0 \times A^t. \end{aligned} \tag{8.22}$$

I.e., we have shown that the sequence $\mathbf{u}$ is the sequence from the same LFSR, but with another initial state Q_0, which is uniquely related to the original initial state Y_0. The probability p is the correlation probability derived in the previous section, and it remains unchanged. We state this result in the form of the following Proposition.

Proposition 8.2: Key-recovering cryptanalysis of the "Grain" family of stream ciphers can be converted to analysis of the general decoding problem. Therefore, the strength of the ciphers in this class is based on the difficulty to solve the general DP. □

For the stream cipher Grain, a new sequence $\mathbf{u}$ is defined as ($\mathcal{B} = \{0, 80\}$)

$$u_t = z_t + z_{t+80}. \tag{8.23}$$

The matrix $M = \sum_{j \in \mathcal{A}} A^j$ is of full rank (which is not necessary always the case). Therefore, by observing the sequence $\mathbf{u}$ we actually observe the output from the LFSR with the same generating polynomial $f(x)$, but another initial state $Q_0 = Y_0 \times M$.

8.4 Distinguishing Attack on Grain

In a *linear distinguishing attack* one observes the keystream and collects the samples from it. These samples form the *type* $P_{\texttt{Type}}$, or empirical distribution. If the observed sequence is from the cipher, then the type will converge to the *cipher distribution* $P_{\texttt{Cipher}}$. If, otherwise, the stream is from a truly random generator, then the type will converge to the *random distribution* $P_{\texttt{Random}}$. The convergence is as close to the original distribution as the number of samples L goes to infinity. For more details see Section 3.4.

The distinguishing attack that we briefly describe in this section is given more in detail in, e.g. [EJ04]. Let us now study the generating function for the LFSR $f(\cdot)$ closer. From [PK95] a ω-weight multiple will have the degree roughly

$$d = (\omega - 1)!^{1/(\omega-1)} 2^{n/(\omega-1)}, \tag{8.24}$$

where l is the size of the LFSR, and the weight $\omega \geq 3$. It means that one could find another parity check equation $f'(\cdot) = 0$ for the stream $\mathbf{u}$ that will be valid with the bias

$$\Delta = 2^{\omega-1}\epsilon^{\omega}. \tag{8.25}$$

Therefore, considering the required number of samples from (3.29), and the degree of the parity check polynomial, we derive the required length of the keystream n to be

$$n \approx d + L = (\omega - 1)!^{\frac{1}{\omega-1}} \cdot 2^{\frac{l}{\omega-1}} + \frac{1}{(2^{\omega-1}\epsilon^{\omega})^2}. \tag{8.26}$$

For the stream cipher Grain we have $l = 80$ and $\epsilon = 2^{-9.678}$, and with $\omega = 3$ the length of the keystream needed is

$$n \approx 2^{40.5} + 2^{54}. \tag{8.27}$$

Recall, in Section 8.2.1 we actually found 16 approximations, each of which will lead to a different sequence ${}_i\mathbf{u}, i = 1, \ldots, 16$. But the parity check function $f'(\cdot)$ will still remain the same. It means that at every time instance

t eight samples can be received, instead of one. Therefore, the final length of the keystream required for the successful distinguishing attack is

$$n_{\text{Grain}} \approx 16^{-1} \cdot 2^{54} = 2^{50}. \tag{8.28}$$

We specify the distinguisher in Table 8.1.

Assume we found $f'(x) = 1 + x^a + x^b = 0$, then:
$I := 0$
for $t = 0 \ldots 2^{50}$ *(time instances)*
for $i = 1, \ldots, 16$ *(16 approximations)*
if $({}_iu_t + {}_iu_{t+a} + {}_iu_{t+b}) = {}_ic$ **then** $I := I + 1$
if $(I/2^{54} - 1/2) > (2^{-9.678}/2)$
then output : Grain
else output : Random source
where ${}_ic = \begin{cases} 0, & \text{if approx. (8.10) has positive bias} \\ 1, & \text{otherwise} \end{cases}$

Table 8.1: The distinguisher for Grain.

8.5 Deriving the LFSR Initial State

In the former section, we have shown how to derive an arbitrary number R of linear approximation equations in the $l = 80$ initial LFSR bits, of bias $\epsilon \simeq 2^{-9.67}$ each, from a sufficient number of keystream bits. Let us denote these equations by

$$\bigoplus_{i=0}^{n-1} \alpha_i^j \cdot y_i = b^j, j = 1, \ldots, R, \tag{8.29}$$

where $\alpha_i^j \in \mathbb{F}_2$.

In this section we show how to use these relations to derive the initial LFSR state Y_0. This can be seen as a decoding problem, up to the fact that the code length is not fixed in advance and one has to find an optimal trade-off between the complexities of deriving a codeword (i.e., collecting an appropriate number of linear approximation equations) and decoding this codeword.

An estimate of the number L of linear approximation equations needed for the right value of the unknown to maximize the indicator

$$I = \#\left\{ j \in \{1, \ldots, L\} \;\middle|\; \bigoplus_{i=0}^{n-1} \alpha_i^j \cdot y_i = b^j \right\}, \tag{8.30}$$

or at least to be very likely to provide say one of the two or three highest values of I, can be determined as follows.

Under the heuristic assumption that for the correct (resp. incorrect) value of Y_0, I is the sum of L independent binary variables x_i distributed according to the Bernoulli law of parameters $p = \Pr\{x_i = 1\} = \frac{1}{2} - \epsilon$ and $q = \Pr\{x_i = 0\} = \frac{1}{2} + \epsilon$ (resp. the Bernoulli law of parameters $\Pr\{x_i = 1\} = \Pr\{x_i = 0\} = \frac{1}{2}$, mean value $\mu = \frac{1}{2}$, and standard deviation $\sigma = \frac{1}{2}$), L can be derived by introducing a threshold of say $T = L(\frac{1}{2} + \frac{3\epsilon}{4})$ for I and requiring:

(i) that the probability that I is larger than T for an incorrect value of Y_0 is less than a suitably chosen false alarm probability p_{fa}.

(ii) that the probability that I is lower than T for the correct value is less than a non detection probability p_{nd} of say 1%.

For practical values of p_{fa}, the first condition is by far the most demanding. Setting the false alarm rate to $p_{fa} = 2^{-l}$ ensures that the number of false alarms is less than 1 in average.

Due to the central limit theorem, $\frac{\sum x_i - L\mu}{\sqrt{L}\sigma}$ is distributed as the normal distribution, so that

$$\Pr\{\frac{1}{L}\sum x_i - \mu > \frac{3\epsilon}{4}\} = \Pr\{\frac{\sum x_i - L\mu}{\sqrt{L}\sigma} > \frac{3\sqrt{L}\epsilon}{4\sigma}\} \tag{8.31}$$

can be approximated by $\frac{1}{\sqrt{2\pi}} \int_{\lambda}^{+\infty} e^{-\frac{t^2}{2}} dt$, where $\lambda = \frac{3\sqrt{L}\epsilon}{2}$. Consequently, if L is selected in such a way that $\frac{3\sqrt{L}\epsilon}{2} = \lambda$, i.e.

$$L = \left(\frac{2\lambda}{3\epsilon}\right)^2, \tag{8.32}$$

where λ is given by:

$$\frac{1}{\sqrt{2\pi}} \int_{\lambda}^{+\infty} e^{-\frac{t^2}{2}} dt = p_{fa} = 2^{-l}, \tag{8.33}$$

then inequality (8.31) is satisfied.

A naive LFSR derivation method would consist of collecting L approximate equations, computing the indicator I independently for each of the 2^l possible values of Y_0 and retaining those Y_0 candidates leading to a value of I larger than the $L(\frac{1}{2}+\frac{3\epsilon}{4})$ threshold. This method would require a low number of keystream bits (say $\frac{L+80}{16}$) but the resulting complexity $L \cdot 2^{80}$ would be larger than the one of exhaustive key search.

In the rest of this section, we show that much lower complexities can be obtained by using the fast Walsh transform algorithm and a few extra filtering techniques in order to speed up computations of correlation indicators. Former examples of applications of similar fast Fourier transform techniques in order to significantly decrease the total complexity of correlation attacks can be found in [Dod03, CJM02].

8.5.1 Use of the Fast Walsh Transform to Speed Up Correlation Computations

8.5.1.1 Basic Method

Let us consider the following problem. Given a sufficient number M of linear approximation equations of bias ϵ involving m binary variables y_0 to y_{m-1}, how to efficiently determine these m variables? Let us denote these M equations by $\sum_{i=0}^{m-1} \alpha_i^j \cdot y_j = b^j, j = 1, \ldots, M$. For a sufficiently large value of M, one can expect the right value of $(y_0, \ldots, y_{m-1})$ to be the one maximizing the indicator

$$\begin{aligned} I(y_0, \ldots, y_{m-1}) =& \#\left\{ j \in \{1, \ldots, M\} \;\middle|\; \sum_{i=0}^{m-1} \alpha_i^j \cdot y_j = b^j \right\} \\ =& \frac{L}{2} + 2 \cdot S(y_0, \ldots, y_{m-1}), \end{aligned} \tag{8.34}$$

where:

$$\begin{aligned} S(y_0, \ldots, y_{m-1}) =& \#\left\{ j \in \{1, \ldots, M\} \;\middle|\; \sum_{i=0}^{m-1} \alpha_i^j \cdot y_i = b^j \right\} \\ & - \#\left\{ j \in \{1, \ldots, M\} \;\middle|\; \sum_{i=0}^{m-1} \alpha_i^j \cdot y_i \neq b^j \right\}. \end{aligned} \tag{8.35}$$

Equivalently one can expect $(y_0, \ldots, y_{m-1})$ to be the value which maximizes the indicator $S(y_0, \ldots, y_{m-1})$. Instead of computing all of the 2^m values of $S(y_0, \ldots, y_{m-1})$ independently, one can derive these values in a combined way using fast Walsh transform computations in order to save time.

Let us recall the definition of the Walsh transform. Given a real function of m binary variables $f(x_1, \ldots, x_{m-1})$, the Walsh transform of f is the real function of m binary variables $F = W(f)$ defined by

$$F(u_0, \ldots, u_{m-1}) = \sum_{x_0,\ldots,x_{m-1} \in \{0,1\}^m} f(x_0, \ldots, x_{m-1})(-1)^{u_0 x_0 + \ldots + u_{m-1} x_{m-1}}. \tag{8.36}$$

Let us define the function $s(\alpha_0, \ldots, \alpha_{m-1})$ by:

$$\begin{aligned} s(\alpha_0, \ldots, \alpha_{m-1}) =& \#\big\{ j \in \{1, \ldots, M\} \mid (\alpha_0^j, .., \alpha_{m-1}^j) \\ & = (\alpha_0, \ldots, \alpha_{m-1}) \wedge b_j = 1 \big\} \\ & - \#\big\{ j \in \{1, \ldots, M\} \mid (\alpha_0^j, .., \alpha_{m-1}^j) \\ =& (\alpha_0, \ldots, \alpha_{m-1}) \wedge b_j = 0 \big\}. \end{aligned} \tag{8.37}$$

The function s can be computed in M steps. Moreover, it is easy to check that the Walsh transform of s is S, i.e.,

$$\forall (y_0, \ldots, y_{m-1}) \in \{0,1\}^m, W(s)(y_0, \ldots, y_{m-1}) = S((y_0, \ldots, y_{m-1})). \tag{8.38}$$

Therefore, the computational cost of the estimation of all the 2^m values of S using fast Walsh transform computations is $M + m \cdot 2^m$; the required memory is 2^m.

8.5.1.2 Improved Hybrid Method

More generally, if $m_1 < m$, one can use the following hybrid method between exhaustive search and Walsh transform in order to save space.

For each of the 2^{m-m_1} values of $(y_{m_1}, \ldots, y_{m-1})$, define the associated restriction S' of S as the m_1 bit Boolean function given by

$$\begin{aligned} S'(y_0, \ldots, y_{m_1-1}) =& \#\left\{ j \in \{1, \ldots, M\} \;\middle|\; \sum_{i=0}^{m_1-1} \alpha_i^j \cdot y_i = \sum_{i=m_1}^{m} \alpha_i^j \cdot y_i \oplus b^j \right\} \\ & - \#\left\{ j \in \{1, \ldots, M\} \;\middle|\; \sum_{i=0}^{m_1-1} \alpha_i^j \cdot y_i \neq \sum_{i=m_1}^{m} \alpha_i^j \cdot y_i \oplus b^j \right\}. \end{aligned} \tag{8.39}$$

It is easy to see that if we define

$$\begin{aligned} s'(\alpha_0,\ldots,\alpha_{m_1-1}) =& \#\Big\{ j\in\{1,\ldots,M\} \;\Big|\; (\alpha_0^j,\ldots,\alpha_{m_1-1}^j) \\ & = (\alpha_0,\ldots,\alpha_{m_1-1}) \wedge \sum_{i=m_1}^{m} \alpha_i^j \cdot y_i \oplus b^j = 1 \Big\} \\ & - \#\Big\{ j\in\{1,\ldots,M\} \;\Big|\; (\alpha_0^j,\ldots,\alpha_{m_1-1}^j) \\ & = (\alpha_0,\ldots,\alpha_{m_1-1}) \wedge \sum_{i=m_1}^{m} \alpha_i^j \cdot y_i \oplus b^j = 0 \Big\}, \end{aligned} \tag{8.40}$$

then S' is the Walsh transform of s'.

Therefore, the computational cost of the estimation of all the 2^m values of S using this method is $2^{m-m_1}L + m_1 \cdot 2^{m_1}$. If we compare this with the former basic Walsh transform method, we see that the required memory decreases from 2^m to 2^{m_1}, whereas the time complexity remains negligible as long as $m_1 \ll log_2(M)$.

8.5.2 First LFSR Derivation Technique

In order to reduce the LFSR derivation complexity when compared with the naive method of complexity $L\cdot 2^l$, we can exploit more keystream to produce more linear approximation equations in the unknowns y_0 to y_{l-1}, and retain only those equations involving the $m < l$ variables y_0 to y_{m-1}, i.e., which coefficients in the $l-m$ variables y_m to y_{l-1} are equal to 0.

Thus a fraction of about 2^{m-l} of the relations are retained and we have to collect about $L2^{l-m}$ approximate relations to retain L relations. This requires a number of keystream bits of

$$\frac{L2^{l-m}+80}{16}. \tag{8.41}$$

As seen in the former section, once the relations have been filtered, the computational cost of the derivation of the values of these m variables using fast Walsh transform computations is about $m2^m$ for the basic method, and more generally $2^{m-m_1}(N + m_1 2^{m_1})$ if fast Walsh transform computations are applied to a restricted set $m_1 < m$ variables.

Thus, the overall time complexity of this method is

$$L2^{l-m} + m2^m, \tag{8.42}$$

and more generally

$$L2^{l-m} + 2^{m-m_1}(L + m_1 2^{m_1}). \tag{8.43}$$

Once the m variables y_0 to y_{m-1} have been recovered, one can either reiterate the same technique for other choices of the m unknown variables, which increases the complexity by a factor of less than 2 if $m \geq \frac{l}{2}$, or test each of the 2^{l-m} candidates in the next step of the attack (NLFSR and key derivation).

An estimate of the number L of equations needed is given by

$$L = \left(\frac{2\lambda}{3\epsilon}\right)^2, \tag{8.44}$$

where λ is determined by the condition

$$\frac{1}{\sqrt{2\pi}} \int_{\lambda}^{+\infty} e^{-\frac{t^2}{2}} dt = 2^{-m}. \tag{8.45}$$

This condition ensures that the expected number of false alarm is less than 1.

The minimal complexity is obtained for $m = 49$. For this parameter value, we have $\lambda = 7.87$ and $L = 2^{24}$. The attack complexity is about 2^{55}, the number of keystream bits required is around 2^{51}, and the memory needed is about 2^{49}.

8.5.3 Second LFSR Derivation Technique

An alternative method is to derive new linear approximation equations (of lower bias) involving $m < l$ unknown variables y_0 to y_{m-1} by combining the R available approximate equations of bias ϵ pair-wise, and retaining only those pairs of relations for which the $l - m$ last coefficients collide. One obtains in this way about $L' = R^2 \cdot 2^{m-l-1}$ new affine equations in y_0 to y_{m-1}, of bias $\epsilon' = 2\epsilon^2$. The allocation of the m variables maximizing the number of satisfied equations can be found by fast Walsh computations as explained in the former Section.

The number L' of relations needed is about $\left(\frac{2\lambda}{3\epsilon'}\right)^2$, where λ is determined by the condition $\frac{1}{\sqrt{2\pi}} \int_{\lambda}^{+\infty} e^{-\frac{t^2}{2}} dt = 2^{-m}$. The required number R of relations of bias ϵ is therefore $R = (L'2^{l-m-1})^{\frac{1}{2}}$, and the number of keystream bits required is about $\frac{R+80}{16}$. The complexity of the derivation of the L' relations is $\max(R, L') = \max((L'2^{l-m-1})^{\frac{1}{2}}, L')$.

Once the L' relations have been derived, the computational cost of the derivation of the values of these m variables using fast Walsh transform computations is about $m \cdot 2^m$ for the basic method, and more generally $2^{m-m_1}(L' + m_1 \cdot 2^{m_1})$ if fast Walsh transform computations are applied to a restricted set $m_1 < m$ variables.

Thus the total complexity of the derivation of the m LFSR bits is:

$$\max((L'2^{l-m-1})^{\frac{1}{2}}, L') + m2^m, \tag{8.46}$$

and more generally:

$$\max((L'2^{l-m-1})^{\frac{1}{2}}, L') + 2^{m-m_1}(L' + m_1 2^{m_1}). \tag{8.47}$$

The minimal complexity is obtained for $m = 36$. For this parameter value, we have $\lambda = 6.65$ and $L' = 2^{41}$. The attack complexity is about 2^{43}, the number of keystream bits required is about 2^{38} and the memory required is about 2^{42}.

8.6 Recovering the NLFSR Initial State and the Key

Once the initial state of the LFSR has been recovered, we want to recover the initial state $(x_0, \ldots, x_{79})$ of the NLFSR. Fortunately, the knowledge of the LFSR removes the nonlinearity of the output function and we can express each keystream bit z_i by one of the following four equations depending on the initial state of the LFSR:

$$\begin{aligned} z_i &= x_i, \\ z_i &= x_i \oplus 1, \\ z_i &= x_i \oplus x_{63+i}, \\ z_i &= x_i \oplus x_{63+i} \oplus 1. \end{aligned} \tag{8.48}$$

Since functions p and q underlying h' are balanced, each equation has the same occurrence probability. We are going to use the non linearity of the output function to recover the initial state of the NLFSR by writing the equations corresponding to the first keystream bits.

The 16 first equations are linear equations involving only bits of the initial state of the NLFSR because $63 + i$ is lower than 80.

To recover all the bits of the initial state, we introduce a technique which consists of building chains of keystream bits. The equations for keystream bits z_{17} to z_{79} involve either one bit of the LFSR ($z_i = x_i$ or $z_i = x_i \oplus 1$) or two bits ($z_i = x_i \oplus x_{63+i}$ or $z_i = x_i \oplus x_{63+i} \oplus 1$). An equation involving only

one bit allows us to instantly recover the value of the corresponding bit of the initial state. This can be considered as a chain of length 0. On the other hand, an equation involving two bits does not allow this because we do not know the value of x_{63+i} (for $i > 16$).

However, by considering not only the equations for z_i but also all the equation for $z_{k\cdot 63+i}$ for $k \geq 1$, we can cancel the bits we do not know and retrieve the value of x_i. With probability $\frac{1}{2}$, the equation for z_{63+i} involves one single unknown bit. Then it provides the value of x_{63+i} and consequently the value of x_i. Here the chain is of length 1, since we have to consider one extra equation to retrieve x_i. The equation for z_{63+i} can also involve two bits with probability $\frac{1}{2}$. Then we have to consider the equation of $z_{2\cdot 63+i}$, which can also either involve only one bit (we have a chain of length 2) or two bits and we have to consider more equations to solve. Each equation has a probability $\frac{1}{2}$ to involve 1 or 2 bits. Consequently the probability that a chain is of length l is $\frac{1}{2^{l+1}}$ and the probability that a chain is of length strictly larger than l is $\frac{1}{2^{l+1}}$.

We want to recover the values of $x_{17}, \ldots, x_{79}$. We have to build 64 different chains. Let us consider $n = 63 \cdot l$ bits of keystream. The probability that one of the chains is of length larger than n is less than $64 \cdot 2^{-l-1}$ and therefore less than 2^{-l+5}. If we want this probability to be bounded by 2^{-10}, then $l > 15$ and $n > 945$ suffices. Consequently a few thousands of keystream bits are needed to retrieve the initial state of the NLFSR and the complexity of the operation is bounded by $64 \cdot l$.

Since the internal state transition function associated to the special key and IV setup mode is one to one, the key can be efficiently derived from the NLFSR and LFSR states at the beginning of the keystream generation by running this function backward.

8.7 Simulations and Results

To confirm that our cryptanalysis is correct, we ran several experiments. First we checked the bias ϵ of Section 8.2.1 by running the cipher with a known initial state of both the LFSR and the NLFSR, computing the linear approximations, and counting the number of fulfilled relations for a very large number of relations. For instance we found that one linear approximation is satisfied 19579367 times out of 39060639, which gives an experimental bias of $2^{-9.63}$, to be compared with the theoretical bias $\epsilon = 2^{-9.67}$.

To check the two proposed LFSR reconstruction methods of Section 8.5, we considered a reduced version of Grain in order to reduce the memory and time required by the attack on a single computer: we shortened the LFSR by a factor of 2. We used an LFSR of size 40 with a primitive feedback

polynomial and we reduced by two the distances for the tap entries of function h': we selected taps number 3, 14, 24, and 33, instead of 3, 25, 46, and 64 for Grain.

The complexity of the first technique for the actual Grain is 2^{55} which is out of reach of a single PC. For our reduced version, the complexity given by the formula of Section 8.5.1 is only 2^{35}. We exploited the 16 linear approximations to derive relations colliding on the first 11 bits. Consequently the table of the Walsh transform is only of size 2^{29}. We used $15612260 \simeq 2^{23}$ relations, which corresponds to a false alarm probability of 2^{-29}. Our implementation required around one hour to recover the correct value of the LFSR internal state on a computer with a Intel Xeon processor running at 2.5 GHz with 3 GB of memory. The Walsh transform computation took only a few minutes.

For the actual Grain, the second technique requires only 2^{43} operations which is achievable by a single PC. However it also requires 2^{42} of memory which corresponds to 350 GB of memory. We do not have such an amount of memory but for the reduced version the required memory is only 2^{29}. Since the complexity given by the formula of Section 8.5.3 is dominated by the required number of relations to detect the bias, our simulation has a complexity close to 2^{43}. In practice, we obtained a result after 4 days of computation on the same computer as above and $2.5 \cdot 10^{12} \simeq 2^{41}$ relations where considered and allowed to recover the correct LFSR initial state.

Finally, we implemented the method of Section 8.6 to recover the NLFSR. Given the correct initial state of the LFSR, and the first thousand keystream bits, our program recovers the initialisation of the NLFSR in a few seconds for a large number of different initialisations of both the known LFSR and unknown NLFSR. We also confirmed the failure probability assessed in Section 8.6 for this method (which corresponds to the occurrence probability of at least one chain of length larger than 15).

8.8 Summary

We have presented a key-recovery attack against Grain which requires 2^{43} computations, 2^{42} bits of memory, and 2^{38} keystream bits. This attack suggests that the following slight modifications of some of the Grain features might improve its strength:

- Introduce several additional masking variables from the NLFSR in the keystream bit computation.
- Replace the nonlinear feedback function g in such a way that the associated function g' be balanced (e.g. replace g by a 2-resilient function). However this is not necessarily sufficient to thwart all similar attacks.

- Modify the filtering function h in order to make it more difficult to approximate.
- Modify the function g and h to increase the number of inputs.

Following recent cryptanalysis of Grain including the key recovery attack reported here and distinguishing attacks based on the same kind of linear approximations as those presented in Section 8.2 [Max06, KHK05] the authors of Grain proposed a tweaked version of their algorithm [HJM05b], where the functions g and h have been modified. This new version of Grain appears to be much stronger and is immune against the statistical attacks presented in this chapter.

The results presented in this chapter have also influenced the design of Grain-128 [HJMM06], a new 128 bit version of the cipher, where I am a co-author.

9

Statistical Analysis of "Dragon"

"If I could be a bird, I think I'd be a penguin, because then I could walk around on two feet with a lot of other guys like me"

Jack Handey

Dragon is a word oriented stream cipher [CHM+05] submitted to the eSTREAM project, designed by a group of researchers, Ed Dawson et al. It is a word oriented stream cipher that operates on key sizes of 128 and 256 bits. The original idea of the design is to use a nonlinear feedback shift register (NLFSR) and a linear part (counter), combined by a filter function to generate a new state of the NLFSR and produce the keystream. The internal state of the cipher is 1088 bits, which is updated by a nonlinear function denoted by F. This function is also used as a filter function producing the keystream. The idea to use a NLFSR is quite modern, and there are not many cryptanalysis techniques on NLFSRs yet developed. Since the internal state of the cipher is large (1088 bits), any kinds of TMTO attacks are not applicable.

The authors of Dragon propose to *resynchronise* the stream after 2^{64} words are produced, i.e., every pair (K, IV) can be used to generate a keystream portion of size no more than 2^{64} words.

REMARK: Let Dragon_0 be the cipher Dragon *without* the resynchronisation requirement.

In this chapter we study the keystream from Dragon_0 and show a statistical weakness. One can find samples from the keystream with the bias around $2^{-77.5}$. We then propose two statistical distinguishers that distinguish Dragon_0 from a random source both requiring around $O(2^{155})$ words of the keystream. In the first scenario the time complexity is around $O(2^{155+32})$ with the memory complexity $O(2^{32})$, whereas the second scenario needs only $O(2^{155})$ of time, but $O(2^{96})$ of memory. The attack is based on a statistical weakness introduced into the keystream by the filter function F.

When we have full Dragon, we can construct an advanced distinguisher as described in Section 3.4.9, which has a positive advantage.

The outline of the chapter is the following. In Section 9.1 a short description of the stream cipher Dragon is given. Afterward, in Section 9.2, we derive linear relations and build our distinguisher. In Section 9.3 we summarize different attack scenarios on Dragon, and finally, in Section 9.4 we present our results, make conclusions and discuss possible ways to overcome the attack.

Notation and Cryptanalysis Assumptions

For notation purposes we use $\oplus$ and $\boxplus$ to denote 32 bit parallel XOR and arithmetical addition modulo 2^{32}, respectively. By $x \gg k$ we denote a binary shift of x by k bits to the right. We write $x^{(t)}$ to refer the value of a variable x at the time instance t. By $P_{\texttt{Expr}}$ we denote a distribution of a random variable or an expression "`Expr`".

To build the distinguishers, standard statistical assumptions can be made, similar to Section 3.4.2.

(a) Assume that at any time t the internal state of NLFSR is from the uniform distribution, i.e., the words B_i are considered independent and uniformly distributed.

(b) Assume that the samples collected from the keystream are independent.

9.1 Short Description of Dragon

Dragon is a stream cipher constructed using a large nonlinear feedback shift register, an update function denoted by F, and a memory denoted by M [1]. It is a word oriented cipher operating on 32 bit words, the NLFSR is 1024 bits long, i.e., 32 words long. The words in the internal state are denoted by $B_i,\ 0 \leq i \leq 31$. The memory M (counter) contains 64 bits, which is used as a linear part with the period of 2^{64}. The cipher handles two key sizes, namely 128 bits keys and 256 bit keys, in our attack we disregard the initialisation procedure and just assume that the initial state of the NLFSR is truly random.

Each round the F function takes six words as input and produces six words of output, as shown in Figure 9.1.

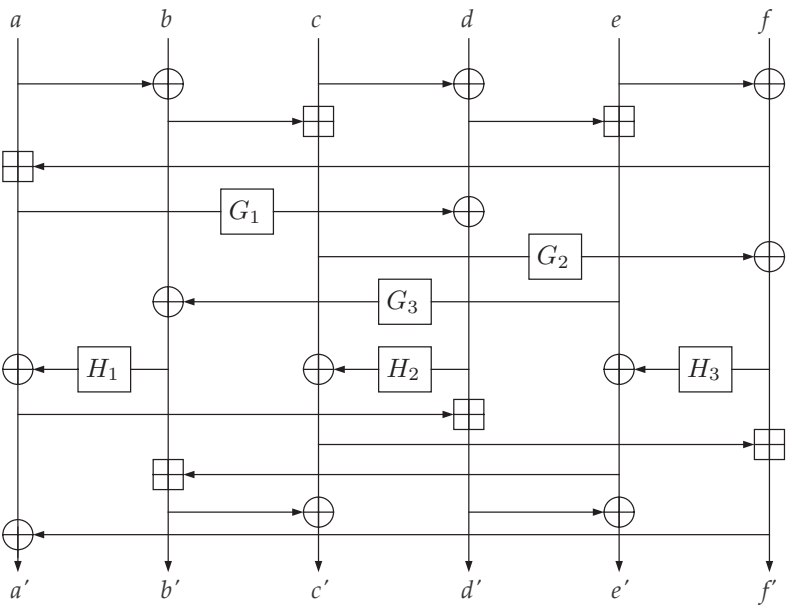


Figure 9.1: The F-function of Dragon.

These six words, denoted by a, b, c, d, e, f, are formed from words of the NLFSR and the memory register M, as explained in (9.1), where $M =$

[1] This is a rather new way to design stream ciphers, when two independent linear and nonlinear parts are combined by a filter function. Similar ideas are used in other proposals to eSTREAM, e.g., stream cipher Grain and others.

Input $= \{B_0 || \ldots || B_{31}, M\}$
1. $(M_L || M_R) = M$.
2. $a = B_0,\ b = B_9,\ c = B_{16},\ d = B_{19},\ e = B_{30} \oplus M_L,\ f = B_{31} \oplus M_R$.
3. $(a', b', c', d', e', f') = F(a, b, c, d, e, f)$.
4. $B_0 = b',\ B_1 = c'$
5. $B_i = B_{i-2},\ 2 \leq i \leq 31$.
6. $M++$
7. $k = a' || e'$

Output $= \{k, B_0, \ldots, B_{31}, M\}$

Figure 9.2: Dragons's Keystream Generation Function.

$(M_L || M_R)$.

$$\begin{array}{lll} a = B_0 & b = B_9 & c = B_{16} \\ d = B_{19} & e = B_{30} \oplus M_L & f = B_{31} \oplus M_R \end{array} \tag{9.1}$$

The F function uses six $\mathbb{Z}_{2^{32}} \rightarrow \mathbb{Z}_{2^{32}}$ S-boxes $G_1,\ G_2,\ G_3,\ H_1,\ H_2$ and H_3, the purpose of which is to provide high algebraic immunity and nonlinearity. These S-boxes are constructed from two other $\mathbb{Z}_{2^8} \rightarrow \mathbb{Z}_{2^{32}}$ S-boxes, S_1 and S_2, as shown below.

$$\begin{aligned} G_1(x) &= S_1(x_0) \oplus S_1(x_1) \oplus S_1(x_2) \oplus S_2(x_3) \\ G_2(x) &= S_1(x_0) \oplus S_1(x_1) \oplus S_2(x_2) \oplus S_1(x_3) \\ G_3(x) &= S_1(x_0) \oplus S_2(x_1) \oplus S_1(x_2) \oplus S_1(x_3) \\ H_1(x) &= S_2(x_0) \oplus S_2(x_1) \oplus S_2(x_2) \oplus S_1(x_3) \\ H_2(x) &= S_2(x_0) \oplus S_2(x_1) \oplus S_1(x_2) \oplus S_2(x_3) \\ H_3(x) &= S_2(x_0) \oplus S_1(x_1) \oplus S_2(x_2) \oplus S_2(x_3), \end{aligned} \tag{9.2}$$

where 32 bits of input, x, is represented by its four bytes as $x = x_0 || x_1 || x_2 || x_3$.

The exact specification of the S-boxes can be found in [CHM+05]. The output of the function F is denoted as (a', b', c', d', e', f'), from which the two words a' and e' forms 64 bits of keystream as $k = a' || e'$. Two other output words from the filter function are used to update the NLFSR as follows $B_0 = b'$, $B_1 = c'$, the rest of the state is updated as $B_i = B_{i-2},\ 2 \leq i \leq 31$. A short description of the keystream generation function is summarized in Figure 9.2.

9.2 Linear Analysis of Dragon$_0$

9.2.1 Linear Approximation of the Function F

Recall, at time t the input to the function F is a vector of six words

$$(a, b, c, d, e, f) = (B_0, B_9, B_{16}, B_{19}, B_{30} \oplus M_L, B_{31} \oplus M_R). \tag{9.3}$$

The output from the function is (a', b', c', d', e', f'). To simplify further expressions let us introduce new variables.

$$\begin{aligned} b'' &= b \oplus a = B_9 \oplus B_0, \\ c'' &= c \boxplus (a \oplus b) = B_{16} \boxplus (B_9 \oplus B_0), \\ d'' &= d \oplus c = B_{19} \oplus B_{16}, \\ f'' &= f \oplus e = B_{30} \oplus B_{31} \oplus M_L \oplus M_R. \end{aligned} \tag{9.4}$$

If the words denoted by Bs are independent, then these new variables will also be independent (since B_{19} is independent of B_{16} and random, then d'' is independent and random as well; similarly, independence of B_{16} lead to the independence of c'', etc.).

The output from F can be expressed via $(a, b'', c'', d'', e, f'')$ as follows.

$$\begin{aligned} a' &= (a \boxplus f'') \oplus H_1(b'' \oplus G_3(e \boxplus d''))\oplus \\ &\qquad \Big((f'' \oplus G_2(c'')) \boxplus \Big(c'' \oplus H_2(d'' \oplus G_1(a \boxplus f''))\Big)\Big), \\ e' &= (e \boxplus d'') \oplus H_3(f'' \oplus G_2(c''))\oplus \\ &\qquad \Big((d'' \oplus G_1(a \boxplus f'')) \boxplus \Big((a \boxplus f'') \oplus H_1(b'' \oplus G_3(e \boxplus d''))\Big)\Big). \end{aligned} \tag{9.5}$$

Let us now analyse the expression for a'. The variable b'' appears only once (in the input of H_1), which means that this input is independent from other terms of the expression, i.e., the term $H_1(\ldots)$ can be substituted by $H_1(r_1)$, where $r_1 = f'' \oplus G_2(c'')$ is some independent and uniformly distributed random variable. A similar situation will happen when the function H_2 is considered.

We would like to approximate the expression for a' as

$$a' = a \oplus N_a, \tag{9.6}$$

where N_a is some non uniformly distributed noise variable. If we XOR both sides with a and then substitute a' with the expression from (9.5), we derive

$$N_a = a \oplus (a \boxplus f'') \oplus H_1(r_1) \oplus \Big((f'' \oplus G_2(c'')) \boxplus (c'' \oplus H_2(r_2))\Big). \tag{9.7}$$

Despite the fact that G and H are $\mathbb{Z}_{2^{32}} \rightarrow \mathbb{Z}_{2^{32}}$ functions, they are not likely to be one-to-one mappings (consider the way the S-boxes are used as $\mathbb{Z}_{2^8} \rightarrow \mathbb{Z}_{2^{32}}$ functions [2]) . It means that even if the input to a G or a H function is completely random, then the output will still be biased. Moreover, the output from the expressions ($x \oplus G_i(x)$ and similarly $x \oplus H_i(x)$) is also biased, since x in these expressions play a role in the approximation of the G_i and the H_i functions. These observations mean that the noise variable N_a, is also biased if the input variables are independent and uniformly distributed.

By a similar observation, the expression for e' can also be approximated as follows.

$$e' = e \oplus N_e, \tag{9.8}$$

where N_e is the noise variable. The expression for N_e can similarly be derived as

$$N_e = e \oplus (e \boxplus d'') \oplus H_3(r_3) \oplus \Big((d'' \oplus G_1(a'')) \boxplus (a'' \oplus H_1(r_4)\Big), \tag{9.9}$$

where $a'' = a \boxplus f''$ is a new random variable, which is also independent since it has f'' as its component, and f'' does not appear anywhere else in the expression (9.9). The two new variables r_3 and r_4 are also independent and uniformly distributed random variables by similar reasons.

9.2.2 Building the Distinguisher

The key observation for our distinguisher, is that one of the input words to the filter function F, at time t is repeated as a part of the input to F at time $t + 15$, i.e.,

$$e^{(t+15)} = a^{(t)} \oplus M_L^{(t+15)}. \tag{9.10}$$

Let us consider the following sum of two words from the keystream.

$$\begin{aligned} s^{(t)} &= a'^{(t)} \oplus e'^{(t+15)} = (a^{(t)} \oplus N_a^{(t)}) \oplus (a^{(t)} \oplus M_L^{(t+15)} \oplus N_e^{(t+15)}) \\ &= \underbrace{N_a^{(t)} \oplus N_e^{(t+15)}}_{N_{tot}^{(t)}} \oplus M_L^{(t+15)}. \end{aligned} \tag{9.11}$$

By this formula we show how to sample from a given keystream, so that the samples $s^{(t)}$ are from some nonuniform distribution $P_{\texttt{Dragon}}$ of the noise variable $N_{tot}^{(t)}$ (later this distribution is also referred to as $P_{N_{tot}^{(t)}}$). The collected n samples $s^{(t)}$, $t = 1, 2, \ldots, n$ form a type $P_{\texttt{Type}}$. Let the noise distribution be $P_{\texttt{Dragon}}$, and the uniform distribution be $P_{\texttt{Random}}$). We use (3.29) for a

[2] The cipher Turing uses similar $\mathbb{Z}_{2^{32}} \rightarrow \mathbb{Z}_{2^{32}}$ functions based on $\mathbb{Z}_{2^8} \rightarrow \mathbb{Z}_{2^{32}}$ S-boxes, which can be regarded as a source of weakness. However, no attack was found on Turing so far.

rough estimate of the required number of samples n. We refer to Section 3.4 and, e.g., to [CHJ02, Gol94] for more details.

The remaining question is how to deal with the counter value M_L. Below we present a set of possible solutions that one could consider.

(1) Guess the initial state of the counter $M^{(0)}$ (in total 2^{64} combinations), and then construct 2^{64} types from the given keystream, assuming the value $M_L^{(t)}$ in correspondence to the guessed initial value of $M^{(0)}$. However, it will increase the time complexity of the distinguisher by a factor of 2^{64}.

(2) Guess the first 32 bits $M_R^{(0)}$ of the initial value of the counter $M^{(0)}$, i.e., 2^{32} values. I.e., at any time t we can express $M_L^{(t)}$ as follows.

$$M_L^{(t)} = M_L^{(0)} \boxplus \Delta^{(t)}, \tag{9.12}$$

where $\Delta^{(t)}$ is known at each time, since $M_R^{(t)}$ is known. Recall (9.11), the noise variable $N_{tot}^{(t)}$ is expressed as $s^{(t)} \oplus M_L^{(t+15)}$. However, using (9.12) this expression can be approximated as

$$s^{(t)} \oplus (M_L^{(0)} \boxplus \Delta^{(t+15)}) \rightarrow s^{(t)} \oplus (M_L^{(0)} \oplus \Delta^{(t+15)}) \oplus N_2, \tag{9.13}$$

where N_2 is a new noise variable due to the approximation of the kind "$\boxplus \Rightarrow \oplus$". Since $M_L^{(0)}$ can be regarded as a constant for every sample $s^{(t)}$, it only "shifts" the distribution, and will not change the bias. Consider that a shift of the uniform distribution is again the uniform distribution, so, the distance between the noise and the uniform distributions will remain the same. This solution requires $O(2^{32})$ guesses, and also introduce a new noise variable N_2.

(3) Consider the sum of two consecutive samples $s^{(t)} \oplus s^{(t+1)}$. Since M_L changes slowly, then with probability $(1-2^{-32})$ we have $M_L^{(t)} = M_L^{(t+1)}$, and this term will be eliminated from the expression for that new sample. Unfortunately, this method will decrease the bias significantly, and then the number of required samples N will be much larger than in the previous cases.

In our attack we tried different solutions, and based on simulations we decided to choose solution (2) for our attack, as it has the lowest attack complexity.

9.2.3 Calculation of the Noise Distribution

Consider the expression for the noise variable

$$s^{(t)} \oplus M_L^{(t+15)} = N_a^{(t)} \oplus N_e^{(t+15)}. \tag{9.14}$$

For simplicity in the formula, we omit time instances for variables.

$$N_{tot}^{(t)} = N_a^{(t)} \oplus N_e^{(t+15)} = (a \boxplus f'') \oplus (a \boxplus d'') \oplus H_1(r_1) \oplus H_3(r_3) \oplus$$
$$\oplus \Big(\big(f'' \oplus G_2(c'')\big) \boxplus \big(c'' \oplus H_2(r_2)\big) \Big) \oplus \Big(\big(d'' \oplus G_1(a'')\big) \boxplus \big(a'' \oplus H_1(r_4)\big) \Big) \tag{9.15}$$

We propose two ways to calculate the distribution of the total noise random variable $N_{tot}^{(t)}$. Let us truncate the word size to k bits (when we consider the expression modulo 2^k), then in the first case the computational complexity is $O(2^{4k})$. This complexity is too high and, therefore, requires the noise variable to be truncated by some number of bits $k \ll 32$, much less than 32 bits. The second solution has a better complexity $O(k2^k)$, but introduces two additional approximations into the expression, which makes the calculated bias smaller than the real value, i.e., by this solution we can verify the lower bound for the bias of the noise variable. Below we describe two methods and give our simulation results on the bias of the noise variable $N_{tot}^{(t)}$.

(I) Consider the expression (9.15) taken modulo 2^k, for some $k = 1 \ldots 32$. Then the distribution of the noise variable can be calculated by the following steps.

a) Construct three distributions, two of them are conditioned

$$P_{(G_2(c'') \mod 2^k | c'')}, \quad P_{(G_1(a'') \mod 2^k | a'')}, \quad P_{(H_1(x) \mod 2^k)}{}^3.$$

The algorithm requires one loop for c'' (a'' and x) of size 2^{32}. The time required is $O(3 \cdot 2^{32})$.

b) Afterwards, construct two more conditioned distributions

$$P_{(d'' \oplus G_1(a'')) \boxplus (a'' \oplus H_1(r_4)) \mod 2^k | d'')}$$

and

$$P_{(f'' \oplus G_1(c'')) \boxplus (c'' \oplus H_2(r_2)) \mod 2^k | f'')}.$$

This requires four loops for $d'', a'', x (= G_1(a'') \mod 2^k)$, and $y (= H_1(r_4) \mod 2^k)$, which takes time $O(2^{4k})$ (and similar for the second distribution).

c) Then, calculate another two conditioned distributions

$$P_{(\mathtt{Expr}_1 | a)} = P_{((a \boxplus f'') \oplus (f'' \oplus G_1(c'')) \boxplus (c'' \oplus H_2(r_2)) \mod 2^k | a)},$$
$$P_{(\mathtt{Expr}_2 | a)} = P_{((a \boxplus d'') \oplus (d'' \oplus G_1(a'')) \boxplus (a'' \oplus H_1(r_4)) \mod 2^k | a)}.$$

Each takes time $O(2^{3n})$.

[3] If the inputs to the H_i functions is random, their distributions are the same, i.e., $P_{H_1} = P_{H_2} = P_{H_3}$.

d) Finally, combine the results, partially using FHT, and then calculate the bias of the noise:

$$P_{N_{tot}} = P_{(\mathtt{Expr}_1|a)} \oplus P_{(\mathtt{Expr}_2|a)} \oplus P_{H_1} \oplus P_{H_3}.$$

This will take time $O(2^{3k} + 3k \cdot 2^k)$.

This algorithm calculates the exact distribution of the noise variable taken modulo 2^k, and has the complexity $O(2^{4k})$. Due to such a high computational complexity we could only manage to calculate the bias of the noise when $k = 8$ and $k = 10$:

$$\begin{aligned} \epsilon_I|_{k=8} &= 2^{-80.59} \\ \epsilon_I|_{k=10} &= 2^{-80.57}. \end{aligned} \tag{9.16}$$

(II) Consider two additional approximations of the second $\boxplus$ to $\oplus$ in (9.15). Then, the total noise can be expressed as

$$\begin{aligned} N_{tot}^{(t)} =& H_1(r_1) \oplus H_2(r_2) \oplus H_3(r_3) \oplus H_1(r_4) \oplus \big(G_2(c'') \oplus c''\big) \\ & \oplus \big(G_1(a'') \oplus a''\big) \oplus N_3 \oplus N_{2,a} \oplus N_{2,e}, \end{aligned} \tag{9.17}$$

where

$$N_3 = (a \boxplus f'') \oplus (a \boxplus d'') \oplus f'' \oplus d'',$$

and $N_{2,a}$ and $N_{2,e}$ are two new noise variables due to the approximation $\boxplus \Rightarrow \oplus$, i.e., $N_{2,a} = (x \boxplus y) \oplus (x \oplus y)$, for some random inputs x and y, and similar for $N_{2,e}$. Introduction of two new noise variables will statistically make the bias of the total noise variable smaller, but it can give us a lower bound of the bias, and also allow us to operate with distributions of size 2^{32}.

First, calculate the distributions $P_{(H_i)}$, $P_{(G_1(a'')\oplus a'')}$ and $P_{(G_1(c'')\oplus c'')}$, each taking time $O(2^{32})$. Afterward, note that the expressions for $N_{2,a}, N_{2,e}$ and N_3 belong to the class of *pseudo-linear functions modulo* 2^k (PLFM), which were introduced in Section 4.1. In the same chapter, algorithms for construction of their distributions were also provided, which take time around $O(\delta \cdot 2^n)$, for some small δ. The last step is to perform the convolution of precomputed distribution tables via FHT in time $O(k2^k)$. Algorithms (PLFM distribution construction and computation of convolutions) and data structures for operating on large distributions are given in Section 4.1. If we consider $k = 32$, then the total time complexity to calculate the distribution table for N_{tot} will be around $O(2^{38})$ operations, which is feasible for a common PC. It took us a few days to accomplish such calculations on a usual

PC with memory 2Gb and 2×200Gb of HDD, and the received bias of N_{tot} was

$$\epsilon_{II}|_{k=32} = 2^{-74.515}. \tag{9.18}$$

If we also approximate $(M_L^{(0)} \boxplus \Delta^{(t)}) \to (M_L^{(0)} \oplus \Delta^{(t)}) \oplus N_2$, and add the noise N_2 to N_{tot}, we receive the bias

$$\epsilon_{II}^{\Delta}|_{k=32} = 2^{-77.5}, \tag{9.19}$$

which is the lower bound. Our distinguisher requires approximately $O(2^{155})$ words of keystream, according to (3.29).

9.3 Attack Scenarios

9.3.1 On Truncated Dragon$_0$

In the previous section we have shown how to sample from the given keystream, where 32 bit samples are drawn from the noise distribution with the bias $\epsilon_{II}^{\Delta}|_{k=32} = 2^{-77.5}$. I.e., our distinguisher needs around $O(2^{155})$ words of keystream to successfully distinguish the cipher from random. Unfortunately, to construct the type correctly we have to guess the initial value of the linear part of the cipher, the lower 32 bits $M_R^{(0)}$ of the counter M. This guess increases the time complexity of our attack to $O(2^{187})$, and requires memory $O(2^{32})$. The algorithm of our distinguisher for Dragon is given in Table 9.1.

We, however, can also show that time complexity can easily be reduced downto $O(2^{155})$, if memory of size $O(2^{96})$ is available. Assume we first construct a table $T[\Delta][s] = \#\{t \equiv \Delta \mod 2^{64}, s^{(t)} = s\}$, where the samples are taken as $s^{(t)} = a'^{(t)} \oplus e'^{(t+15)}$. Afterwards, for each guess of $M_L^{(0)}$ the type $P_{\mathtt{Type}}(\cdot)$ is then constructed from the table T in time $O(2^{96})$. Hence, the total time complexity will be $O(2^{155} + 2^{32} \cdot 2^{96}) \approx O(2^{155})$. This scenario is given in Table 9.2.

9.3.2 On Full Dragon

One can derive an *advanced distinguisher* (see Section 3.4.9) for Dragon based on multiple distinguishers for Dragon$_0$ built in the previous sections. The derived bias of sampling is $2^{-77.5}$, and the time for "guessing" in Scenario I takes the multiple of $O(2^{32})$ time. Resynchronisation runs after every 2^{64} keystream words.

According to (3.61) for success with probability close to 1 we should have around $n \approx O(2^{278})$ samples, which is over the exhaustive search. However, if, say, $n \approx O(2^{253})$ samples are available, then this distinguisher will still

for $0 \leq M_R^{(0)} < 2^{32}$
  $P_{\texttt{Type}}(x) = 0, \ \ \forall x \in \mathbb{Z}_{2^{32}}$
  $\Delta = 0$ (or $= -1$, if $M_R^{(0)} = 0$)
  for $t = 0, 1, \ldots, 2^{155}$
    if $(M_R^{(0)} \boxplus t) = 0$ **then** $\Delta = \Delta \boxplus 1$
    $s^{(t)} = a'^{(t)} \oplus e'^{(t+15)} \oplus \Delta$
    $P_{\texttt{Type}}(s^{(t)}) = P_{\texttt{Type}}(s^{(t)}) + 1$
  $I = \sum_{x \in \mathbb{Z}_{2^{32}}} P_{\texttt{Type}}(x) \cdot \log_2(P_{\texttt{Dragon}}(x)/2^{-32})$
  If $I \geq 0$ **break** and **output** : Dragon
output : Random source

Table 9.1: The distinguisher for Dragon_0 (Scenario I).

for $0 \leq t < 2^{155}$
  $T[t \mod 2^{64}][a'^{(t)} \oplus e'^{(t+15)}] + +$
for $M_R^{(0)} = 0, \ldots, 2^{32} - 1$
  for $\Delta = 0, \ldots, 2^{64} - 1$
    for $x = 0, \ldots, 2^{32} - 1$
      $P_{\texttt{Type}}\Big(x \oplus \big((\Delta \boxplus M_R^{(0)}) \gg 32\big)\Big) + = T[\Delta][x]$
  $I = \sum_{x \in \mathbb{Z}_{2^{32}}} P_{\texttt{Type}}(x) \cdot \log_2(P_{\texttt{Dragon}}(x)/2^{-32})$
  If $I \geq 0$ **break** and **output** : Dragon
output : Random source

Table 9.2: Distinguisher for Dragon_0 with lower time complexity (Scenario II).

have some advantage. The success probability can be calculated via formulas from Section (3.5.1). Roughly, the advantage of such a distinguisher for Dragon will then be at least $O(2^{-25})$. When the key of 256 bits is used, this is slightly faster than the brute-force attack.

9.4 Summary

Two versions of a distinguishing attack on Dragon$_0$ were found. The first scenarios requires a computational complexity of $O(2^{187})$ and needs memory $O(2^{32})$. However, the second scenario has a lower time complexity around $O(2^{155})$, but requires a larger amount of memory $O(2^{96})$.

However, when only one pair (K, IV) can be used to generate the keystream of size 2^{64}, we could only find a distinguisher which has an advantage around $O(2^{-25})$, and the keystream is $O(2^{253})$, which is a slightly less efforts than for an exhaustive search. For a cipher with a huge internal state it still shows a potential weakness in the keystream, and shows its divergence from a purely random number generator.

We give a few suggestions now to prevent Dragon from the described weakness:

1) The linear part M changes predictably, when the initial state is known. It might be more difficult to mount the attack if the update of M would depend on some state of the NLFSR.

2) Another leakage is that two words $a'||e'$ are accessible to the attacker. If we would have an access only to a', or, may be, some other combination of the output from F (like, the output $a'||d'$, instead), then it might also protect the cipher from this attack. However, both these suggestions have weaknesses for different reasons.

3) One more weakness are poor G_i and H_i S-boxes. Perhaps, they can be constructed in a different way, such that S-boxes to be closer to a one-to-one mapping function.

A few other new stream cipher proposals to eSTREAM are based on NLFSRs, which has not been extensively investigated so far. We believe that it is important to study such primitives, since it could be an interesting replacement for widely used LFSR based stream ciphers.

We believe that a potentially stronger chosen IV distinguishing attack can be attached to the full Dragon. An attack with support of exploiting IV difference might be more efficient than the one derived in this chapter.

10

Concluding Remarks

"Happiness is not that blind as it seems"

Russian Emperor, Ekaterina II

This thesis tries to accumulate linear cryptanalysis techniques on stream ciphers; it presents information about theoretical analysis methods and statistical processing of information extracted from the keystream of a stream cipher. Different analysis issues are described for various scenarios. Additionally, various extended distinguishers are introduced, for example when the noise distribution is unknown.

Moving from a binary to a multidimensional analysis yields a much better result in cryptanalysis. However, when moving to a multidimentional analysis one can meet technical problems with the implementation and the evaluation of analysis algorithms. For several scenarios that cover most cases in cryptanalysis, we provide necessary algorithms and data structures that allowed us to overcome such technical problems. These techniques gave us new results in cryptanalysis of different primitives, the examples of which were also presented.

However, there are still open problems. For example, if a *pseudo-linear function* could be extended with the operators $\lll, \ll, \ggg, \gg$ (shifts and rotations), then we might have a powerful tool for finding collisions for hash functions such as MD4/5 and SHA-0/1. Although nowadays stream ciphers attract much of scientific attention, hash functions will obviously receive a closer look in the near future.

A

Common Statistics in Cryptanalysis

"I only know that I know nothing"

Socrates

A.1 One Sample Point Inference

Let us denote μ and σ to be the *mean* and the *variance* of the probability mass function P, from where the sample $\mathbf{x}^n$ is drawn. For the type $P_{\mathbf{x}}$ (which is constructed from the sample $\mathbf{x}^n$) we define the *sample mean* and *sample unbiased variance* as follows.

$$\begin{aligned} \overline{\mathbf{x}} &= \frac{1}{n}\sum_{i=1}^{n} x_i \\ s^2 &= \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{\mathbf{x}})^2. \end{aligned} \tag{A.1}$$

Theorem A.1 (Central Limit Theorem (CLT)): A random sample $\mathbf{x}^n$ satisfies

$$\frac{\overline{\mathbf{x}} - \mu}{\sigma/\sqrt{n}} \overset{n\to\infty}{\sim} \mathcal{N}(0,1), \tag{A.2}$$

i.e.,

$$\lim_{n\to\infty} \Pr\{\frac{\overline{\mathbf{x}} - \mu}{\sigma/\sqrt{n}} \leq z\} = \Phi(z). \tag{A.3}$$

□

In the Theorem above $\Phi(x)$ is the cumulative density function for the normal distribution $\mathcal{N}(0,1)$, defined as

$$\begin{aligned}\Phi(x) &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}y^2} dy \\ &= \frac{1}{2}\big(1 + \text{erf}(x/\sqrt{2})\big), \\ \Phi^{-1}(p) &= \sqrt{2} \cdot \text{erf}^{-1}(2p-1),\end{aligned} \tag{A.4}$$

and erf is the *error function* defined as

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_{0}^{x} e^{-t^2} dt. \tag{A.5}$$

Actually, when σ is unknown but the number of samples n is large, people often approximate the value $\sigma \approx s$, i.e., the variation is itself *estimated* along with the mean.

A.1.1 z-statistics

This statistic is applied when the variance σ is known. The null-hypothesis H_0 is $\mu = \mu_0$, and there can exist three different alternative hypothesis

- $H_1 : \mu > \mu_0$ – is a one-sided from the right hypothesis.
- $H_1 : \mu < \mu_0$ – is a one-sided from the left hypothesis.
- $H_1 : \mu \neq \mu_0$ – is a two-sided hypothesis.

The z-statistics is the value

$$z = \frac{\overline{\mathbf{x}} - \mu_0}{\sigma/\sqrt{n}} \overset{n\to\infty}{\sim} \mathcal{N}(0,1), \tag{A.6}$$

which is $\mathcal{N}(0,1)$ distributed if the hypothesis is true. The p value is the *area under the curve* for PDF of the normal distribution, for x larger or smaller than z, or $|x|$ just larger than z, depending on the alternative hypothesis choice.

EXAMPLE *A.1 (Two-Sided Z-Statistics):* A sample of 40 sales receipts from a grocery store has $\hat{\mu} = \$137$ and $\sigma = \$30.2$. We have two hypothesis

$$\begin{aligned} H_0 &: \mu = 150, \\ H_1 &: \mu \neq 150, \end{aligned} \tag{A.7}$$

and require the significance level to be $p_\alpha = 0.01$, i.e., 1% of a wrong answer is allowed.

For the decision purpose, we first calculate the statistic z

$$z = \frac{137 - 150}{30.2\sqrt{40}} = -2.72. \tag{A.8}$$

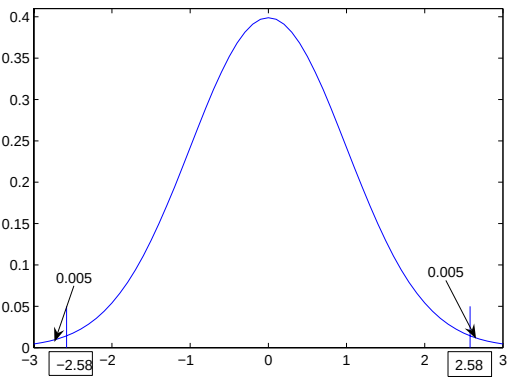


Figure A.1: The probability mass function of $\mathcal{N}(0,1)$, and the rejection regions for the two-sided decision rule.

From Figure A.1 it is easy to see that we need to put half of p_α in the left tail, and the other half of p_α in the right tail. Thus, the decision rule is

$$\delta = \begin{cases} H_0, & \text{if } |z| \leq 2.58 \\ H_1, & \text{if } |z| > 2.58 \end{cases} \tag{A.9}$$

Since $|-2.72| > 2.58$, we conclude that the mean is significantly different from \$150, and reject the null hypothesis in favour of the alternative. □

The confidence level p_α and the value of z are related as

$$p_\alpha = \Phi(-|z|). \tag{A.10}$$

Typical values for z and the corresponding p-values are as follows.

Confidence interval $1-p_\alpha$	p_α	$z_{p_\alpha/2}$
0.90	0.10	1.645
0.95	0.05	1.96
0.99	0.01	2.58

When σ is not known, but the number of samples n is large ($n > 30$), then usually people approximate $\sigma \approx s$, and continue with the z-statistic.

A.1.2 t-statistics

When the value of σ is not known *and* the sample size n is small ($n < 30$), then a t-statistic is used:

$$t = \frac{\overline{\mathbf{x}} - \mu_0}{s\sqrt{n}} \sim t_{n-1}, \tag{A.11}$$

where t_{n-1} is the Student's t-distribution with degree of freedom $n-1$. This is a quite seldom example in cryptanalysis, and we give it for completeness.

A.2 One Sample Multi-Dimensional Inference

In the previous sections we have shown how to perform inference when one parameter was enough to estimate. However, when many parameters have to be tested, other methods can be used.

A.2.1 Chi-Square Test

Proving that two distributions are the same or different is an important task for cryptanalysis. Chi-square test of significance provides necessary technique to establish a relevant hypothesis.

Let us have k bins (assume $|\mathcal{X}| = k$) and n samples $x_1, x_2, \ldots, x_n$ from the population P. The samples $\mathbf{x}^n$ form the type $P_{\mathbf{x}}$. Let o_i be the *number of observed* events in the i^{th} bin, and u_i is the *number of expected* events in the same bin, according to P. Then the chi-square statistic is

$$\chi^2 = \sum_{i=1}^{k} \frac{(o_i - u_i)^2}{u_i} = \sum_{x\in\mathcal{X}} \frac{(P_{\mathbf{x}}(x) - P(x))^2}{P(x)/n} \overset{n\to\infty}{\sim} P_{\chi^2_\nu}. \tag{A.12}$$

The *chi-square probability density function* is as follows

$$P_{\chi^2_\nu}(x) = \frac{x^{r/2-1}e^{-x/2}}{\Gamma(r/2)2^{r/2}}, \tag{A.13}$$

where $\Gamma(x)$ is the standard *gamma function* [1]

$$\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt. \tag{A.14}$$

The cumulative distribution function is then

$$Q_{\chi^2_\nu}(x) = \frac{\gamma(\nu/2, x/2)}{\Gamma(\nu/2)}, \tag{A.15}$$

where $\gamma(a, z)$ is the *incomplete gamma function* defined as

$$\gamma(a, z) = \int_0^z t^{a-1}e^{-t}dt. \tag{A.16}$$

In other words, it is the probability that the sum of ν independent random *normal* variables of unit variance will be larger than χ^2. Because of the *central limit theorem*, when n is large, the variable $(o_i - u_i)^2/u_i$ tends to behave as normaly distributed. However, since this is an approximation of the real distribution of the statistic, thus,

This statistic requires either a large number of bins k, or a large number of samples n;

then the chi-square statistic behaves rather close to χ^2_ν. If the number of samples n is much larger than the number of bins k, then this statistics can be fairly applied. The parameter ν is the *degree of freedom*. If the sum of o_i's is equal to the sum of μ_i's (which is the usual case), then

$$\nu = k - 1. \tag{A.17}$$

If we make a hypothesis that the type $P_{\mathbf{x}}$ is drawn from some priori distribution P, then the error probability (or level of confidence) of such a hypothesis is strongly related to the threshold that is chosen to compare the statistic χ^2 with. That threshold is also called the *critical value* χ^2_{crit} for the decision. For example, when $\nu = 2$ and maximum 5% of a error decision is allowed, the threshold for the chi-square statistic should be at least 5.99, as it can be seen in Figure A.2.

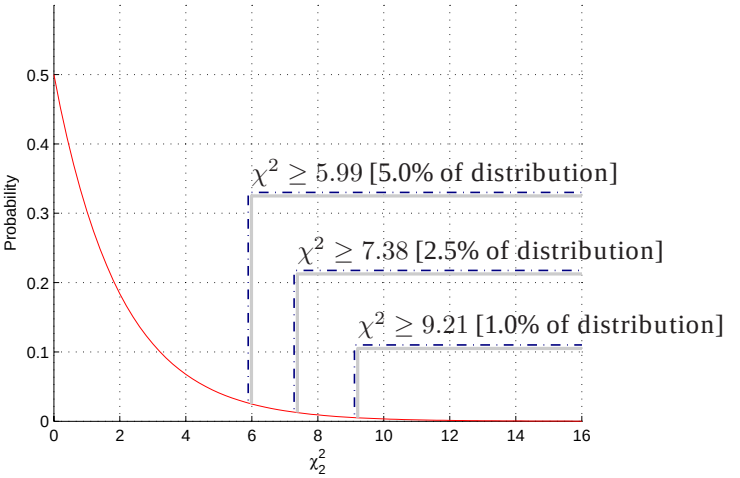


Figure A.2: Theoretical sampling distribution of Chi-square when the degree of freedom is 2.

In cryptanalysis, typical values for ν are powers of 2, such as $1, 2, 4, 8, \ldots$. In Figure A.3 the probability (PDF) and cumulative (CDF) density functions of χ^2_ν for typical values of freedom degree ν are shown.

EXAMPLE *A.2 (χ^2 Test of Fitting [Rob]):* Suppose at a sneaky casino they use dice that are slightly biased as given in the expected frequency column below. When the pit-boss gets a new die he performs the test to determine whether or not the die is biased in the same way or not. After $n = 120$ times of rolling the die, the pit-boss made up the following table ($k = 6$).

Value	Observed frequency	Expected frequency
1	20	24
2	23	24
3	19	18
4	26	22
5	18	14
6	14	18

[1] When x is a positive integer, $\Gamma(x) = (x-1)!$.

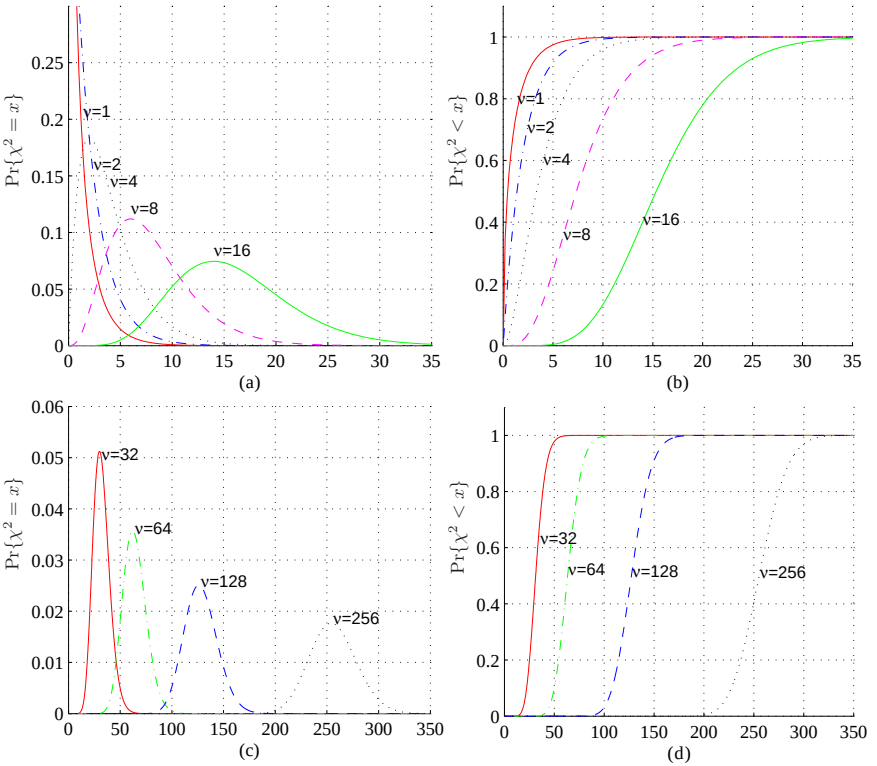


Figure A.3: Probability and cumulative density functions for χ^2_ν: (a) PDF $\chi^2_{\nu=1,2,4,8,16}$; (b) CDF $\chi^2_{\nu=1,2,4,8,16}$; (c) PDF $\chi^2_{\nu=32,64,128,256}$; (d) CDF $\chi^2_{\nu=32,64,128,256}$.

So now we have two hypothesis as follows

$$\begin{aligned} H_0: &\quad \text{observed values are like expected} \\ H_1: &\quad \text{observed values are different than expected.} \end{aligned} \tag{A.18}$$

Let us have a significance level $p_\alpha = 0.05$. Then we have the χ^2 value

$$\begin{aligned} \chi^2 =& \frac{(20-24)^2}{24} + \frac{(23-24)^2}{24} + \frac{(19-18)^2}{18} \\ &+ \frac{(26-22)^2}{22} + \frac{(18-14)^2}{14} + \frac{(14-18)^2}{18} \\ \approx& 3.5229, \end{aligned} \tag{A.19}$$

where the degree of freedom is $\nu = 5$. Consulting the tables for the χ^2_ν distribution we see that the p-value is around 70% (here, $p = \Pr\{\chi^2_5 > 3.5229\} \approx 0.7$), and it is more than 5%. Hence, we accept H_0. □

A.2.2 Kolmogorov-Smirnoff Test

The measure d in the K-S test is simple: d is the *maximum value* of the absolute difference between two cumulative distribution functions, i.e.,

$$d = \max_{-\infty<x_0<\infty} |P_{\mathbf{x}}(X < x_0) - P(X < x_0)|, \tag{A.20}$$

where X is some random variable.

A central feature of K-S test is that it is invariant under reparameterisation of X. The function that enters into the calculation of the confidence level is the following sum

$$Q_{KS}(\lambda) = 2\sum_{j-1}^{\infty}(-1)^{j-1}e^{-2j^2\lambda^2}, \tag{A.21}$$

which is a monotonic function with $Q_{KS}(0) = 1$ and $Q_{KS}(\infty) = 0$. In terms of this function, the significance level of d is then approximated as

$$\Pr\{d > \hat{d}\} = Q_{KS}\left([\sqrt{n'} + 0.12 + 0.11/\sqrt{n'}]d\right), \tag{A.22}$$

where n' is the effective number of data points, $n' = n$ for the case of one distribution, and $n' = \frac{n_1 n_2}{n_1+n_2}$ for the case when two types are compared with n_1 and n_2 samples, respectively.

A.3 Convergence in Distribution

In this section we try to establish the relation between the sample size n, the distance from the type $P_{\mathbf{x}}$ to the priori population distribution P, and the error probability of a decision rule p_α for specific cases.

A.3.1 Information Theoretical Approach

We again assume that we have some i.i.d. $\mathbf{x}^n = (x_1, x_2, \ldots, x_n)$ sample of n events from some distribution P over the domain $\mathcal{X}$. The sample forms the type $P_{\mathbf{x}}$. For some appropriately chosen parameter ρ_0 we define the decision rule as follows.

$$\delta(P_{\mathbf{x}}) = \begin{cases} H_0, & \text{if } \rho(P_{\mathbf{x}}||P) \le \rho_0 \\ H_1, & \text{if } \rho(P_{\mathbf{x}}||P) > \rho_0 \end{cases} \tag{A.23}$$

Then the divergence between $P_{\mathbf{x}}$ and P depends on n and its upper bound is expressed by the following theorem.

Theorem A.2 (Theorem 12.2.1 from [CT91]):

$$p_\alpha = \Pr\{\rho(P_{\mathbf{x}}||P) > \rho_0\} \leq 2^{-n\left(\rho_0 - |\mathcal{X}|\frac{\log(n+1)}{n}\right)} \tag{A.24}$$

and consequently, $\rho(P_{\mathbf{x}}||P) \to 0$ with probability 1 as $n \to \infty$. □

From the theorem above the sample size n is easy to derive if one consider

$$n \overset{m.b.}{\geq} \frac{-\log_2 p_\alpha + |\mathcal{X}|\log(n+1)}{\rho_0}. \tag{A.25}$$

For example, if n is around 2^{128}, then one should assume the relation

$$n \geq \frac{-\log_2 p_\alpha + 128|\mathcal{X}|}{\rho_0}. \tag{A.26}$$

The decision rule can be bounded via another metric, statistical distance of degree 1. If we set

$$\epsilon = \sqrt{\frac{\rho_0 \ln 2}{2}}, \tag{A.27}$$

then we can derive the following expression:

$$\begin{aligned}
\Pr\{|P_{\mathbf{x}} - P| > \epsilon\} &= \Pr\{|P_{\mathbf{x}} - P|^2 > \ln 2 \cdot \rho_0/2\} \\
&\overset{(\text{Lem.3.3})}{\leq} \Pr\{\rho(P_{\mathbf{x}}||P) > \rho_0\} \\
&\overset{(\text{Th.A.2})}{\leq} 2^{-n\left(\rho_0 - |\mathcal{X}|\frac{\log(n+1)}{n}\right)} \\
&\overset{(\text{A.27})}{=} 2^{-n\left(\frac{2\epsilon^2}{\ln 2} - |\mathcal{X}|\frac{\log(n+1)}{n}\right)}.
\end{aligned} \tag{A.28}$$

A.3.2 Through the Relation to χ^2 Test for γ-Flat Distributions

Consider the case when the priori probability mass function is a γ-*flat function*, defined as follows.

Definition A.1: (γ-Flat Distribution) Distribution P over the probability space $\mathcal{X}$ is called γ-flat if $\forall x \in \mathcal{X} \Rightarrow P(x) = \frac{1}{|\mathcal{X}|} + \gamma_x$, where $|\gamma_x| \leq \gamma$, for some fixed $\gamma \geq 0$. □

Note that any distribution is 1-flat, and 0-flat distribution is the uniform distribution. When the sample $\mathbf{x}^n$ is drawn according to P from n trials, we are interested in the error probability

$$p_\alpha = \Pr\{|P_\mathbf{x} - P| > \epsilon\}, \tag{A.29}$$

where ϵ is some parameter chosen in advance. I.e., the decision rule for our hypothesis is

$$\delta(P_\mathbf{x}) = \begin{cases} H_0: & \text{if } |P_\mathbf{x} - P| \le \epsilon \\ H_1: & \text{if } |P_\mathbf{x} - P| > \epsilon. \end{cases} \tag{A.30}$$

To start with, consider the following intermediate Chebyshev's lemma.

Lemma A.3 (Chebyshev's Sum Inequalities): If $a_1 \ge a_2 \ge \ldots \ge a_k$ and $b_1 \ge b_2 \ge \ldots \ge b_k$ then

$$k \sum_{i=1}^{k} a_i b_i \ge \left(\sum_{i=1}^{k} a_i \right) \left(\sum_{i=1}^{k} b_i \right). \tag{A.31}$$

Similarly, if $a_1 \ge a_2 \ge \ldots \ge a_k$ and $b_1 \le b_2 \le \ldots \le b_k$ then

$$k \sum_{i=1}^{k} a_i b_i \le \left(\sum_{i=1}^{k} a_i \right) \left(\sum_{i=1}^{k} b_i \right). \tag{A.32}$$

□

From these inequalities it follows that

$$\sum_{i=1}^{k} x_i^2 \ge \frac{1}{k} \left(\sum_{i=1}^{k} |x_i| \right)^2, \tag{A.33}$$

for any values of $x_1, \ldots, x_k$, since we can simply sort x_i's and then set $a_i = b_i = x_i$. Recall the χ^2 statistic from Section A.2.1. It behaves as $P_{\chi^2_\nu}$ as long as the number of samples n tends to infinity (actually, it is enough to have $n \ge 35\nu$). For a typical cryptanalysis case, the degree of freedom $\nu = |\mathcal{X}| - 1$ is never large; for a byte-wise distribution it is $\nu = 255$. On the other hand, the number of samples n is typically huge, such as $n = 2^{20}, 2^{50}, 2^{100} \ldots$. Therefore, in cryptanalysis we can apply the χ^2 statistic rather safely. Recall

the expression for the χ^2 statistic,

$$\begin{aligned}\chi^2 &= n \cdot \sum_{x\in\mathcal{X}} \frac{(P_{\mathbf{x}}(x) - P(x))^2}{P(x)} \\ &\overset{\text{(Def.A.1)}}{\geq} \frac{n}{\frac{1}{|\mathcal{X}|} + \gamma} \sum_{x\in\mathcal{X}} (P_{\mathbf{x}}(x) - P(x))^2 \\ &\overset{\text{(A.33)}}{\geq} \frac{n}{|\mathcal{X}|\left(\frac{1}{|\mathcal{X}|} + \gamma\right)} (2 \cdot |P_{\mathbf{x}} - P|)^2, \end{aligned} \tag{A.34}$$

behaving as $P_{\chi^2_\nu}$ when n is large, *independently* of the priori distribution P. Demanding that $|P_{\mathbf{x}} - P|$ should be at most ϵ for the null hypothesis to be accepted, the corresponding threshold for the statistic is then derived as

$$\chi^2_{\text{crit}} \geq \frac{4n\epsilon^2}{1 + \gamma|\mathcal{X}|}, \tag{A.35}$$

and, therefore, the decision rule is $\delta(P_{\mathbf{x}}) = H_0$ if $\chi^2 \leq \chi^2_{\text{crit}}$.

When γ is *negligible* (i.e., $\gamma|\mathcal{X}|$ is much less than 1), we can approximate that decision rule as $\delta(P_{\mathbf{x}}) = H_0$ if $\chi^2 \leq 4n\epsilon^2$. I.e., the probability of success is at least the probability $\Pr\{\chi^2 < 4n\epsilon^2\}$. Thus, we have

$$p_\alpha \leq 1 - Q_{\chi^2_{|\mathcal{X}|-1}}(4n\epsilon^2) = 1 - \frac{\gamma(q, 2n\epsilon^2)}{\Gamma(q)}, \tag{A.36}$$

where $q = \frac{|\mathcal{X}|-1}{2}$.

From Figure A.3 one can observe that *negligible error probability is achieved when n is taken such that* [2]: $4n\epsilon^2 \geq 2|\nu|$, from where we conclude that the sample size n is *well enough* to be around

$$n \geq \frac{|\mathcal{X}|}{2\epsilon^2}. \tag{A.37}$$

EXAMPLE *A.3 (Distribution Fitting Through Statistical Distance via χ^2_ν):* Assume we collect n samples x^n from some priori probability density function P over the probability space $\mathcal{X}$ of size $|\mathcal{X}| = 256$. The distribution P is a γ-flat distribution with $\gamma \ll 1/|\mathcal{X}|$. We would like to know how large the number of samples n should be, so that the statistical distance $|P_{\mathbf{x}} - P|$ is not more than $\epsilon = 2^{-55}$; what is the error probability?

[2] Actually, from Figure A.3 it is easy to see that the error probability is rapidly changed from 1 to 0 when the critical threshold χ^2_{crit} is around ν. Thus, letting $\chi^2_{\text{crit}} = 2|\nu|$ we receive a negligible error probability, around 2^{-50} (calculated in Matlab). This probability is larger when ν is small, and gets smaller when ν increases.

We just take n according to (A.37) as

$$n \geq \frac{256}{2 \cdot (2^{-55})^2} = 2^{117}. \tag{A.38}$$

The error probability is then calculated according to (A.36) as

$$p_\alpha \leq 1 - Q_{\chi^2_{255}}(4 \cdot 2^{117} \cdot (2^{-55})^2) \leq 2^{-100}. \tag{A.39}$$

However, if we would take $n = 2^{116}$ (that corresponds to $\chi^2_{\text{crit}} \approx \nu$), then the error probability will be significant, $p_\alpha \leq 0.4706$. This example also shows that the found bound is rather tight. □

A.4 Statistics for Correlation Attacks

A.4.1 On Bit Estimation

In Section 3.5.1 a classical scheme for a bit estimation is given. The probability of error for such a scenario is when the number of wrong bits in the sequence $\mathbf{z}^n$ is more than $n/2$. There are, basically, two possible ways to calculate this probability.

A.4.1.1 Via Combinatorics

Let $n_0 = n/2$ and $\#\mathbf{z}^n$ denotes the number of ones in the sequence, then we have [MW06]:

$$\begin{aligned} p_{\mathtt{err}} &= \Pr\{\#\mathbf{z}^n \geq n_0 | X = 0\} \\ &= \sum_{k=n_0}^{n} \binom{n}{k} (1-p)^k p^{n-k} \\ &= I_{1-p}(n_0, n - n_0 + 1), \end{aligned} \tag{A.40}$$

where $I_p(a, b)$ is the *incomplete beta function* defined as

$$I_p(a, b) = \frac{1}{B(a, b)} \int_0^p t^{a-1}(1-t)^{b-1} dt, \tag{A.41}$$

and $B(a, b)$ is the *beta function* defined as

$$B(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}, \tag{A.42}$$

where $\Gamma(n)$ is the *gamma function* defined in (A.14).

When n is large (and this is the usual case), then the easiest way to get success and error probabilities is to use the tools of *Matlab*, such as

$$\begin{aligned} p_{\mathtt{err}} &= \mathtt{betainc}(1-\mathtt{p}, \mathtt{n}_0, \mathtt{n}-\mathtt{n}_0+1) \\ &= \mathtt{betainc}(1-\mathtt{p}, \mathtt{n}/2, \mathtt{n}/2+1) \\ p_{\mathtt{succ}} &= \mathtt{betainc}(\mathtt{p}, \mathtt{n}/2+1, \mathtt{n}/2) \end{aligned} \tag{A.43}$$

A.4.1.2 Via Probability Theory

Let $p = 1/2 + \epsilon$ and then $1 - p = 1/2 - \epsilon$, for some $\epsilon > 0$. Note that the error probabilities for $X = 0$ and $X = 1$, are the same. Therefore, we assume that $X = 1$. Then, the samples x_i satisfy Bernoulli distribution $\mathrm{Be}(\frac{1}{2} + \epsilon, (\frac{1}{2} + \epsilon)(\frac{1}{2} - \epsilon))$.

Let $S_n = \sum_{i=1}^{n} z_i$, then $S_n \sim \mathrm{Bin}_{\frac{1}{2}+\epsilon}(k, n)$. When n is large then the *central limit theorem (CLT)* tells us that

$$\frac{\sqrt{n}(\frac{1}{n}S_n - \mu)}{\sigma} \to \mathcal{N}(0, 1).$$

I.e., $\mathrm{Bin}_{\frac{1}{2}+\epsilon}(k, n) \to \mathcal{N}(\mu, \sigma)$, with $\mu = n(\frac{1}{2} + \epsilon)$, and $\sigma^2 = n(\frac{1}{4} - \epsilon^2)$. The error probability is then calculated as

$$\begin{aligned} p_{\mathrm{err}} = \Pr\{S_n < n/2 | X = 1\} &< \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{n/2} e^{-(y-\mu)^2/(2\sigma^2)} dy \\ = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{n/2-\mu}{\sigma\sqrt{2}}\right)\right] &= \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{-\epsilon\sqrt{n/2}}{\sqrt{\frac{1}{4}-\epsilon^2}}\right)\right], \end{aligned} \tag{A.44}$$

where $\mathrm{erf}(z)$ is the *error function* defined in (A.5).

EXAMPLE *A.4 (Bit Estimation):* Let $p = 1/2 + 2^{-20}$ and $n = 2^{38}$. Then by the first method (combinatorics) we have

$$p_{\mathrm{err}} \approx 0.1586 \quad p_{\mathrm{succ}} \approx 0.8414. \tag{A.45}$$

Whereas, by the second method we receive

$$p_{\mathrm{err}} < 0.1587, \tag{A.46}$$

which is the upper bound for the error probability. Note that the values from both methods are quite close to each other, it happens because when n is large the binomial distribution is well approximated by the normal distribution. □

Trough the central limit theorem (see Th. A.1), it is easy to show that to achieve low error probability the number of samples should be of size as in (3.29).

A.4.2 Fast Correlation Attack

Let the generating polynomial for the LFSR be

$$g(x) = 1 + c_1 x + c_2 x^2 + \ldots + c_l x^l, \tag{A.47}$$

and let the weight of the polynomial be $w = \sum_{i=1}^{l} c_i$, i.e., the number of taps, or nonzero positions from the LFSR that are used in generation of the next state.

In [MS88,MS89] Meier and Staffelbach (MS) proposed to consider special parity check equations, to estimate the bits of the LFSR one by one. Since the weight of $g(\cdot)$ is w, then for any bit we can get $w+1$ parity check equations by just shifting the linear relation equation inspired from $g(\cdot)$.

Secondly, using the fact that $g(x)^j = g(x^j)$ for any $j = 2^k$, other parity check equations are generated by repeated squaring of $g(x)$. The total number of such parity check equations, $N_{\text{pce}}^{(\text{MS})}$, that can be derived in this way is determined as

$$N_{\text{pce}}^{(\text{MS})} \approx (w+1)\log_2\left(\frac{n}{2l}\right). \tag{A.48}$$

Parity check equations including the bit s_i can be written as

$$\begin{aligned} s_i + b_1 &= 0 \\ s_i + b_2 &= 0 \\ &\vdots \\ s_i + b_{N_{\text{pce}}} &= 0, \end{aligned} \tag{A.49}$$

where b_j's are linear expression from the bits of the stream $\mathbf{s}$, which can be approximated by bits from the accessible keystream $\mathbf{z}$. The bits s_i can be estimated by counting the number of b_j's equal to zero and one. The majority value is the estimation value for s_i. For the bit estimation see Section 3.5.1.

If $\Pr\{z_i = s_i\} = p = 1/2 + \epsilon$, then, according to Lemma 3.5, we have

$$\Pr\{b_1 = 0\} = 1/2 + 2^{w-1}\epsilon^w. \tag{A.50}$$

I.e., if the weight of the generating polynomial $g(x)$ is large, then the estimation of the bit s_i is difficult, because the cross-over probability is then very close to $1/2$.

For the complete description of this decoding method see [MS89].

A.4.3 The Algorithm by Mihaljević et al., and Chose et al.

In the recent past many algorithms were proposed. An algorithm based on *convolution codes* was proposed at Eurocrypt 1999 by T. Johansson and F. Jönsson in [JJ99c]. Another one based on *turbo codes* was presented at Crypto 1999 by the same authors in [JJ99b]. After these two new techniques were published, many new research results have been appeared, such as in [CT00, CJS00, MFI00, MFI02, CJM02], and others.

In this section we study in brief the method proposed by M. Mihaljević, M. Fossorier, and H. Imai at FSE 2001 [MFI02], together with the improvement proposed by P. Chose, A. Joux, and M. Mitton at Eurocrypt 2002 [CJM02], as one of the most recent result. This method is currently the most powerful.

The idea is simple. We need to estimate l bits of the stream $\mathbf{s}$ (having n bits of the stream $\mathbf{z}$) to reconstruct the initial state of the LFSR. One first guess the first B bits, and we hope that the guess is correct. Then we choose some parameter $D > l - B$ and our plan is to estimate D bits (more than necessary). The algorithm has two phases: the *precomputation phase*, and the *decoding phase*.

In the *precomputation phase* for each bit i out of D bits, which we plan to estimate, construct a set Ω_i of parity check linear equations, each containing w terms, including the i^{th} bit. If the length of the keystream is n, then the average number of parity checks can be found is

$$\Omega \approx 2^{B-l}\binom{n}{w-1}. \tag{A.51}$$

When $w \leq 4$, the basic square-root time-memory tradeoff has the time and memory complexities $O(Dn^{\lceil (w-1)/2 \rceil} \log n)$ and $O(n^{\lfloor (w-1)/2 \rfloor})$, respectively.

In the *decoding phase* the lists Ω_i are used to estimate D bits. Additionally, there is a threshold θ, and if during the estimation the difference between the number of predicted bits and the number of unpredicted bits is less than θ, then the estimation is considered bad, and the bit is discarded. Thus, we require that at least $l - B$ bits out of D to be estimated correctly.

The next idea is to set a small integer δ, and require that at least $l - B + \delta$ bits out of D are estimated correctly, instead. Thus, additionally, one can assume that at most δ bits out of $l - B + \delta$ can actually be wrong. Checking one set of $l - B$ predicted bits takes roughly $O(1/(1 - h(p)))$ of time. The total time for the decoding part is

$$O\Big(2^B D \log_2 \Omega + (1 + p_{\mathtt{err}}(2^B - 1))\binom{l - B + \delta}{\delta}\frac{1}{1 - h(p)}\Big), \tag{A.52}$$

where p is the crossover probability for the BSC, $p_{\mathtt{err}}$ is the probability that a wrong guess gives us at least $l - B + \delta$ estimated bits, and $h(p)$ is the binary entropy.

The choice of optimal parameters (B, D, θ and δ) is related to the probabilities of success of the attack $p_{\mathtt{succ}}$, and the probability of error $p_{\mathtt{err}}$.

Let $q = (\frac{1}{2} + 2^{w-2}\epsilon^{w-1})$ be the probability that one parity check holds the correct prediction. Then the probability that $\Omega - t$ parity checks give us the correct result is

$$p_1(t) = \sum_{j=\Omega-t}^{\Omega} (1-q)^{\Omega-j} q^j \binom{\Omega}{j}, \tag{A.53}$$

where t is the smallest integer (it is also related to θ as $\theta = \Omega - 2t$) such that

$$p_1(t) \geq \frac{l - B + \delta}{D} < 1. \tag{A.54}$$

The probability that at least $\Omega - t$ parity-check equations will produce a wrong result is

$$p_2(t) = \sum_{j=\Omega-t}^{\Omega} q^{\Omega-j} (1-q)^j \binom{\Omega}{j}. \tag{A.55}$$

If we set

$$p_v = \frac{p_1(t)}{p_1(t) + p_2(t)}, \tag{A.56}$$

then the success probability is derived as

$$p_{\mathtt{succ}} = \sum_{j=0}^{\delta} \binom{l - B + \delta}{j} p_v^{l-B+\delta-j} (1-p_v)^j. \tag{A.57}$$

The probability that with a wrong guess more than $l-B+\delta$ bits are estimated is

$$p_{\mathtt{err}} = \sum_{j=l-B+\delta}^{D} \binom{D}{j} e(t)^j (1-e(t))^{D-j}, \tag{A.58}$$

where

$$e(t) = \frac{1}{2^{\Omega-1}} \sum_{j=\Omega-t}^{\Omega} \binom{\Omega}{j}. \tag{A.59}$$

For more detailed information on the algorithm we refer to [MFI02] and [CJM02].

When Ω is large and the probability q is close to $1/2$ it is computationally hard to calculate the values $t, p_1(t), p_2(t)$, since the equations for them operate on small biases, large binomials and powers. For example, in the equation (A.54), the change of the parameter t from 0 to Ω changes the value of $p_1(t)$ from 0 to 1. We need to find a smallest value of t such that the threshold in (A.54) is satisfied. However, since the probabilities q and $1-q$ are very close to $1/2$, their biases are therefore important. The precision of these calculations must be around $\texttt{prec} \approx O(-(w-1)\Omega \log \epsilon)$, which is hard to implement.

To compute these difficult sums we can use Matlab for evaluating the *incomplete beta function* in a similar manner as in (A.41). However, for large biases ϵ one can also use the Chernoff bound technique, instead of performing the sums computations.

Bibliography

[Bab95] S. Babbage. A space / time tradeoff in exhaustive search attacks on stream ciphers. In *European Convention on Security and Detection*, volume 408 of *IEE Conference Publication*, 1995.

[BB05] E. Barkan and E. Biham. Conditional estimators: An effective attack on A5 / 1. In B. Preneel and S. Tavares, editors, *Selected Areas in Cryptography—SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 1–19. Springer-Verlag, 2005.

[BBK03] E. Barkan, E. Biham, and N. Keller. Instant ciphertext only cryptanalysis of GSM encrypted communication. In D. Boneh, editor, *Advances in Cryptology—CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 600–616. Springer-Verlag, 2003.

[BD00] E. Biham and O. Dunkelman. Cryptanalysis of the A5 / 1 GSM stream cipher. In B. E. Roy and E. Okamoto, editors, *Progress in Cryptology—INDOCRYPT 2000*, volume 1977 of *Lecture Notes in Computer Science*, pages 43–51. Springer-Verlag, 2000.

[BF00] E. Biham and V. Furman. Impossible differential on 8-round MARS' core. In *AES Candidate Conference*, pages 186–194, 2000.

[BGM06] C. Berbian, H. Gilbert, and A. Maximov. Cryptanalysis of Grain. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/019 (2006-01-02), 2006. *http://www.ecrypt.eu.org/stream*.

[BGW99] M. Briceno, I. Goldberg, and D. Wagner. A pedagogical implementation of A5/1. Available at *http://jya.com/a51-pi.htm* (accessed August 18, 2003), 1999.

[Blu03] SIG Bluetooth. Bluetooth specification. Available at *http://www.bluetooth.com* (accessed August 18, 2003), 2003.

[BP84] T. Beth and F. Piper. The Stop-and-Go generator. In T. Beth, N. Cot, and I. Ingemarsson, editors, *Advances in Cryptology—EUROCRYPT'84*, volume 209 of *Lecture Notes in Computer Science*, pages 88–92. Springer-Verlag, 1984.

[BS90] E. Biham and A. Shamir. Differential cryptanalysis of DES-like cryptosystems. In A. J. Menezes and S. A. Vanstone, editors, *Advances in Cryptology—CRYPTO'90*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer-Verlag, 1990.

[BS00] A. Biryukov and A. Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In T. Okamoto, editor, *Advances in Cryptology—ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 2000.

[BS05] E. Biham and J. Seberry. Py (Roo) : A fast and secure stream cipher using rolling arrays. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/023 (2005-04-29), 2005. *http://www.ecrypt.eu.org/stream/ciphers/py/py.ps*.

[BSW00] A. Biryukov, A. Shamir, and D. Wagner. Real time cryptanalysis of A5/1 on a PC. In B. Schneier, editor, *Fast Software Encryption 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 2000.

[BVP+03] M. Boesgaard, M. Vesterager, T. Pedersen, J. Christiansed, and O. Scavenius. Rabbit: A new high-performance stream cipher. In T. Johansson, editor, *Fast Software Encryption 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 307–329. Springer-Verlag, 2003.

[BW99] A. Biryukov and D. Wagner. Slide attacks. In L. Knudsen, editor, *Fast Software Encryption'99*, volume 1636 of *Lecture Notes in Computer Science*, pages 245–259. Springer-Verlag, 1999.

[BW00] A. Biryukov and D. Wagner. Advanced slide attacks. In B. Preneel, editor, *Advances in Cryptology—EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 589–606. Springer-Verlag, 2000.

[CB90] G. Casella and R. L. Berger. *Statistical Inference*. Wadsworth and Brooks / Cole, Pacific Grove, CA, 1990. ISBN 0-534-24312-6.

[CG86] A. H. Chan and R. A. Games. On the linear span of binary sequences obtained from finite geometries. In A. M. Odlyzko, editor, *Advances in Cryptology—CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 405–417. Springer-Verlag, 1986.

[CG88a] W. G. Chambers and D. Gollmann. Generators for sequences with near-maximal linear equivalence. In *IEE Proceedings, v.135, Pt. E, n.1, Jan 1988*, pages 67–69, 1988.

[CG88b] W. G. Chambers and D. Gollmann. Lock-in effect in cascades of clock-controlled shift-registers. In C. G. Günter, editor, *Advances in Cryptology—EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 331–343. Springer-Verlag, 1988.

[CHJ02] D. Coppersmith, S. Halevi, and C. S. Jutla. Cryptanalysis of stream ciphers with linear masking. In M. Yung, editor, *Advances in Cryptology—CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 515–532. Springer-Verlag, 2002.

[CHM+05] K. Chen, M. Henricksen, W. Millan, J. Fuller, L. Simpson, E. Dawson, H. Lee, and S. Moon. Dragon: A fast word based stream cipher. eSTREAM, ECRYPT Stream Cipher Project, Report 2005 / 006 (2005-04-29), 2005.

[CJM02] P. Chose, A. Joux, and M. Mitton. Fast correlation attacks: An algorithmic point of view. In L. Knudsen, editor, *Advances in Cryptology—EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 209–221. Springer-Verlag, 2002.

[CJS00] V. Chepyzhov, T. Johansson, and B. Smeets. A simple algorithm for fast correlation attacks on stream ciphers. In B. Schneier, editor, *Fast Software Encryption 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 181–195. Springer-Verlag, 2000.

[CKM93] D. Coppersmith, H. Krawczyk, and Y. Mansour. The shrinking generator. In D. R. Stinson, editor, *Advances in Cryptology—CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 22–39. Springer-Verlag, 1993.

[CKPS00] N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In B. Preneel, editor, *Advances in Cryptology—EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Springer-Verlag, 2000.

[CLRS01] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001. ISBN 81-203-2141-3.

[CM03] N. Courtois and W. Meier. Algebraic attacks on stream ciphers with linear feedback. In E. Biham, editor, *Advances in Cryptology—EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer-Verlag, 2003.

[Cou03] N. Courtois. Higher order correlation attacks, XL algorithm and cryptanalysis of Toyocrypt. In P. J. Lee and C. H. Lim, editors, *Information Security and Cryptology – ICISC 2002*, volume 2587 of *Lecture Notes in Computer Science*, pages 182–199. Springer-Verlag, 2003.

[CP02] N. Courtois and J. Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In Y. Zheng, editor, *Advances in Cryptology—ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer-Verlag, 2002.

[CP03] N. Courtois and J. Patarin. About the XL algorithm over GF(2). In M. Joye, editor, *Topics in Cryptology—CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 141–157. Springer-Verlag, 2003.

[CP05] C. De Canniére and B. Preneel. TRIVIUM – a stream cipher construction inspired by block cipher design principles. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/030 (2005-04-29), 2005. *http://www.ecrypt.eu.org/stream*.

[CS91] V. Chepyzhov and B. Smeets. On a fast correlation attack on certain stream ciphers. In D. W. Davies, editor, *Advances in Cryptology—EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 176–185. Springer-Verlag, 1991.

[CT91] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. A Wiley-Interscience Publication, 1991. ISBN 0-471-06259-6.

[CT00] A. Canteaut and M. Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In B. Preneel, editor, *Advances in Cryptology—EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 573–588. Springer-Verlag, 2000.

[DGM06] D. K. Dalai, K. C. Gupta, and S. Maitra. Notion of algebraic immunity and its evaluation related to fast algebraic attacks. Cryptology ePrint Archive, Report 2006/018, 2006. *http://eprint.iacr.org/*.

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, 1976.

[Dic58] L. Dickson. *Linear Groups with an Exposition of the Galois Field Theory*. Dover, 1958. ISBN 0-486-43914-3.

[DK05] J. Daemen and P. Kitsos. Submission to ECRYPT call for stream ciphers: the self-synchronizing stream cipher MOSQUITO. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/018 (2005-04-29), 2005. *http://www.ecrypt.eu.org/stream/ciphers/mosquito/mosquito.pdf*.

[DKDM04] K. C. Gupta D. K. Dalai and S. Maitra. Results on algebraic immunity for cryptographically significant Boolean functions. In A. Canteaut and K. Viswanathan, editors, *Progress in Cryptology—INDOCRYPT 2004*, volume 3348 of *Lecture Notes in Computer Science*. Springer-Verlag, 2004.

[DKL+98] J. Dhem, F. Koeune, P. Leroux, P. Mestr, J-J. Quisquater, and J. Williams. A practical implementation of the timing attack. Technical Report CG-1998/1, UCL Crypto Group Technical Report Series, 1998.

[Dod03] M. W. Dodd. *Applications of the Discrete Fourier Transform in Information Theory and Cryptology*. PhD thesis, Royal Holloway, University of London, Information Security Group, Egham, Surrey TW20 0EX, UK, 2003.

[Dra67] A. W. Drake. *Fundamentals of Applied Probability Theory*. McGraw-Hill Book Company, New York, NY, 1967. ISBN 0-070-17815-1.

[DXS91] C. Ding, G. Xiao, and W. Shan. *The Stability Theory of Stream Ciphers*, volume 561. Springer-Verlag, New York, NY, 1991. ISBN 3-540-54973-0, 0-387-54973-0.

[ECR05] eSTREAM: ECRYPT Stream Cipher Project, IST-2002-507932. Available at *http://www.ecrypt.eu.org/stream/* (accessed September 29, 2005), 2005.

[EJ00] P. Ekdahl and T. Johansson. SNOW — a new stream cipher. In *Proceedings of First Open NESSIE Workshop*, 2000.

[EJ01] P. Ekdahl and T. Johansson. Another attack on A5/1. In *Proceedings of International Symposium on Information Theory*, page 160. IEEE, 2001.

[EJ02a] P. Ekdahl and T. Johansson. Distinguishing attacks on SOBER-t16 and SOBER-t32. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 210–224. Springer-Verlag, 2002.

[EJ02b] P. Ekdahl and T. Johansson. A new version of the stream cipher SNOW. In K. Nyberg and H. Heys, editors, *Selected Areas in Cryptography—SAC 2002*, volume 2595 of *Lecture Notes in Computer Science*, pages 47–61. Springer-Verlag, 2002.

[EJ04] H. Englund and T. Johansson. A new simple technique to attack filter generators and related ciphers. In H. Handschuh and A. Hasan, editors, *Selected Areas in Cryptography—SAC 2004*, volume 3357 of *Lecture Notes in Computer Science*, pages 39–53. Springer-Verlag, 2004.

[Ekd03] P. Ekdahl. *On LFSR Based Stream Ciphers: Analysis and Design*. PhD thesis, Lund University, Department of Information Technology, P.O. Box 118, SE–221 00, Lund, Sweden, 2003. ISBN 91-628-5868-8.

[EM05] H. Englund and A. Maximov. Attack the Dragon. In S. Maitra, V. Madhavan, and R. Venkatesan, editors, *Progress in Cryptology—INDOCRYPT 2005*, volume 3797 of *Lecture Notes in Computer Science*, pages 130–142. Springer-Verlag, 2005.

[FL01] S. R. Fluhrer and S. Lucks. Analysis of the E_0 encryption system. In S. Vaudenay and A. M. Youssef, editors, *Selected Areas in Cryptography—SAC 2001*, volume 2259 of *Lecture Notes in Computer Science*, pages 38–48. Springer-Verlag, 2001.

[Flu01] S. R. Fluhrer. Cryptanalysis of the SEAL 3.0 pseudorandom function family. In M. Matsui, editor, *Fast Software Encryption 2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 135–143. Springer-Verlag, 2001.

[FM00] S. R. Fluhrer and D. A. McGrew. Statistical analysis of the alleged RC4 keystream generator. In B. Schneier, editor, *Fast Software Encryption 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 19–30. Springer-Verlag, 2000.

[For88] R. Forré. The strict avalanche criterion: Spectral properties of Boolean functions and an extended definition. In S. Goldwasser, editor, *Advances in Cryptology—CRYPTO'88*, volume 403 of *Lecture Notes in Computer Science*, pages 450–468. Springer-Verlag, 1988.

[Fra94] J. B. Fraleigh. *A First Course in Abstract Algebra*. Addison-Wesley, New York, NY, 1994. ISBN 0-201-76390-7.

[FWS+03] N. Ferguson, D. Whiting, B. Schneier, J. Kelsey, S. Lucks, and T. Kohno. Helix: Fast encryption and authentication in a single cryptographic primitive. In T. Johansson, editor, *Fast Software Encryption 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 330–346. Springer-Verlag, 2003.

[Gef73] P. R. Geffe. How to protect data with ciphers that are really hard to break. In *Electronics, v.46, n.1, Jan 1973*, pages 99–101, 1973.

[GH05] J. Dj. Golić and P. Hawkes. Vectorial approach to fast correlation attacks. *Designs, Codes, and Cryptography*, 35(1):5–19, 2005.

[GM04] J. Dj. Golić and G. Morgari. Vectorial fast correlation attacks. *Cryptology ePrint Archive, Report 2004/247*, 2004.

[Gol82] S. Golomb. *Shift Register Sequences*. Aegean Park, revised edition, 1982. ISBN 0-89412-048-4.

[Gol93] J. Dj. Golić. Correlation via linear sequential circuit approximation of combiners with memory. In R.A. Rueppel, editor, *Advances in Cryptology—EUROCRYPT'92*, volume 658 of *Lecture Notes in Computer Science*, pages 113–123. Springer-Verlag, 1993.

[Gol94] J. Dj. Golić. Intrinsic statistical weakness of keystream generators. In J. Pieprzyk and R. Safavi-Naini, editors, *Advances*

in Cryptology—ASIACRYPT'94, volume 917 of *Lecture Notes in Computer Science*, pages 91–103. Springer-Verlag, 1994.

[Gol96] J. Dj. Golić. Linear models for keystream generators. *IEEE Transactions on Computers*, 45(1):41–49, January 1996.

[Gol97a] J. Dj. Golić. Cryptanalysis of alleged A5/1 stream cipher. In W. Fumy, editor, *Advances in Cryptology—EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 239–255. Springer-Verlag, 1997.

[Gol97b] J. Dj. Golić. Linear statistical weakness of alleged RC4 keystream generator. In W. Fumy, editor, *Advances in Cryptology—EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 226–238. Springer-Verlag, 1997.

[Gol99] J. Dj. Golić. Linear models for a time-variant permutation generator. *IEEE Transactions on Information Theory*, 45(7):2374–2382, 1999.

[Gol00] J. Dj. Golić. Cryptanalysis of three mutually clock-controlled Stop/Go shift registers. *IEEE Transactions on Information Theory*, 46(3):1081–1090, 2000.

[Gol04] J. Dj. Golić. A weakness of the linear part of stream cipher MUGI. In B. Roy and W. Meier, editors, *Fast Software Encryption 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 178–192. Springer-Verlag, 2004.

[Gup04] K. C. Gupta. *Cryptographic and Combinatorial Properties of Boolean Functions and S-boxes*. PhD thesis, Indian Statistical Institute, Applied Statistical Unit, 203, B. T. Road, Calcutta 700 108, India, 2004.

[Gut95] A. Gut. *An Intermediate Course in Probability*. Springer-Verlag, 1995. ISBN 0-387-94507-5.

[HCJ02] S. Halevi, D. Coppersmith, and C. S. Jutla. Scream: A software-efficient stream cipher. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 195–209. Springer-Verlag, 2002.

[HJM05a] M. Hell, T. Johansson, and W. Meier. Grain — a stream cipher for constrained environments. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/010 (2005-04-29), 2005. *http://www.ecrypt.eu.org/stream/ciphers/grain/grain.pdf*.

[HJM05b] M. Hell, T. Johansson, and W. Meier. Grain V.1. — a stream cipher for constrained environments, 2005. *http://www.it.lth.se/grain/grainV1.pdf*.

[HJMM06] M. Hell, T. Johansson, A. Maximov, and W. Meier. A stream cipher proposal: Grain-128. To appear at *IEEE International Symposium on Information Theory—2006, Seattle, USA*, 2006.

[HR00a] P. Hawkes and G. G. Rose. Primitive specification and supporting documentation for SOBER-t16 submission to NESSIE. In *Proceedings of First Open NESSIE Workshop*, 2000. Available at *http://www.cryptonessie.org* (accessed October 5, 2003).

[HR00b] P. Hawkes and G.G. Rose. Primitive specification and supporting documentation for SOBER-t32 submission to NESSIE. In *Proceedings of First Open NESSIE Workshop*, 2000. Available at *http://www.cryptonessie.org* (accessed October 5, 2003).

[HS05] J. Hong and P. Sarkar. Rediscovery of the time-memory tradeoff. Available at *http://eprint.iacr.org/2005/090/* (accessed February 21, 2006), 2005.

[Jen80] S. M. Jennings. *A Special Class of Binary Sequences*. PhD thesis, Royal Holloway, University of London, Information Security Group, Egham, Surrey TW20 0EX, UK, 1980.

[Jen82] S. M. Jennings. Multiplexed sequences: Some properties of the minimum polynomial. In T. Beth, editor, *Advances in Cryptology—EUROCRYPT'82*, volume 1440 of *Lecture Notes in Computer Science*, pages 189–206, 1982.

[JJ99a] T. Johansson and F. Jönsson. Fast correlation attacks based on turbo code techniques. In M. J. Wiener, editor, *Advances in Cryptology—CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 181–197. Springer-Verlag, 1999.

[JJ99b] T. Johansson and F. Jönsson. Fast correlation attacks based on turbo code techniques. In M. J. Wiener, editor, *Advances in Cryptology—CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 181–197. Springer-Verlag, 1999.

[JJ99c] T. Johansson and F. Jönsson. Improved fast correlation attacks on stream ciphers via convolutional codes. In J. Stern, editor, *Advances in Cryptology—EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 347–362. Springer-Verlag, 1999.

[JJ00] T. Johansson and F. Jönsson. Fast correlation attacks through reconstruction of linear polynomials. In M. Bellare, editor, *Advances in Cryptology—CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 300–315. Springer-Verlag, 2000.

[JJ02] T. Johansson and F. Jönsson. On the complexity of some cryptographic problems based on the general decoding problem. *IEEE Transactions on Information Theory*, 48(10):2669–2678, 2002.

[JM03] T. Johansson and A. Maximov. A linear distinguishing attack on Scream. In *Information Symposium in Information Theory—ISIT 2003*, page 164. IEEE, 2003.

[Jön02] F. Jönsson. *Some Results on Fast Correlation Attacks*. PhD thesis, Lund University, Department of Information Technology, P.O. Box 118, SE–221 00, Lund, Sweden, 2002. ISBN 91-7167-024-6.

[Kah67] D. A. Kahn. *The CodeBreakers: The Story of Secret Writing*. Macmillan Publishing, New York, NY, 1967. ISBN 0-684-83130-9.

[KB96] L. R. Knudsen and T. A. Berson. Truncated differentials of SAFER. In D. Gollmann, editor, *Fast Software Encryption'96*, volume 1039 of *Lecture Notes in Computer Science*, pages 15–26. Springer-Verlag, 1996.

[Key76] E. Key. An analysis of the structure and complexity of nonlinear binary sequence generators. *IEEE Transactions on Information Theory*, 22:732–736, 1976.

[KHK05] S. Khazaei, M. Hassanzadeh, and M. Kiaei. Distinguishing attack on Grain. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/071 (2005-10-14), 2005.

[KJJ99] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. J. Wiener, editor, *Advances in Cryptology—CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer-Verlag, 1999.

[KMP+98] L. R. Knudsen, W. Meier, B. Preneel, V. Rijmen, and S. Verdoolaege. Analysis methods for (alleged) RC4. In K. Ohta and D. Pei, editors, *Advances in Cryptology—ASIACRYPT'98*, volume 1998 of *Lecture Notes in Computer Science*, pages 327–341. Springer-Verlag, 1998.

[Knu94] L. R. Knudsen. Truncated and higher order differentials. In B. Preneel, editor, *Fast Software Encryption'94*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer-Verlag, 1994.

[Koc96] P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Advances in Cryptology—CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer-Verlag, 1996.

[KR95] B. Kaliski and M. Robshaw. Message authentication with MD5. *CryptoBytes*, 1(1):5–8, Spring 1995.

[Kra02] M. Krause. BDD-based cryptanalysis of keystream generators. In L. R. Knudsen, editor, *Advances in Cryptology—EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 222–237. Springer-Verlag, 2002.

[KS99] A. Kipnis and A. Shamir. Cryptanalysis of the HFE public key cryptosystem. In M. J. Wiener, editor, *Advances in Cryptology—CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 19–30. Springer-Verlag, 1999.

[KS03] A. Klimov and A. Shamir. A new class of invertible mappings. In *CHES '02: Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, pages 470–483, London, UK, 2003.

[KW02] L. R. Knudsen and D. Wagner. Integral cryptanalysis. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 112–127. Springer-Verlag, 2002.

[LALM04] H. J. Lee, M. Ahn, S. Lim, and S.-J. Moon. A study on smart card security evaluation criteria for side channel attacks. In *ICCSA (1)*, pages 517–526, 2004.

[LeV77] W. J. LeVeque. *Fundamentals of Number Theory*. Dover Publications, Inc., 1977. ISBN 0-486-68906-9.

[Lip02] H. Lipmaa. On differential properties of pseudo-Hadamard transform and related mappings. In A. Menezes and P. Sarkar, editors, *Progress in Cryptology—INDOCRYPT 2002*, volume 2551 of *Lecture Notes in Computer Science*, pages 48–61. Springer-Verlag, 2002.

[Llo92] S. Lloyd. Counting binary functions with certain cryptographic properties. *Journal of Cryptology*, 5(2):107–131, 1992.

[LM02] H. Lipmaa and S. Moriai. Efficient algorithms for computing differential properties of addition. In M. Matsui, editor, *Fast Software Encryption 2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 336–350. Springer-Verlag, 2002.

[LWD04] H. Lipmaa, J. Wallén, and P. Dumas. On the additive differential probability of exclusive-or. In B. Roy and W. Meier, editors, *Fast Software Encryption 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 317–331. Springer-Verlag, 2004.

[Mas69] J. L. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15:122–127, 1969.

[Mat94] M. Matsui. Linear cryptanalysis method for DES cipher. In T. Helleseth, editor, *Advances in Cryptology—EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer-Verlag, 1994.

[Max04] A. Maximov. On linear approximation of modulo sum. In B. Roy and W. Meier, editors, *Fast Software Encryption 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 483–484. Springer-Verlag, 2004.

[Max05] A. Maximov. Two linear distinguishing attacks on VMPC and RC4A and weakness of RC4 family of stream ciphers. In H. Gilbert and H. Handschuh, editors, *Fast Software Encryption 2005*, volume 3557 of *Lecture Notes in Computer Science*, pages 342–358. Springer-Verlag, 2005.

[Max06] A. Maximov. Cryptanalysis of the "Grain" family of stream ciphers. In *ACM Symposium on InformAtion, Computer and Communications Security (ASIACCS'06)*, pages 283–288, 2006.

[MFI00] M. Mihaljević, M. Fossorier, and H. Imai. A low-complexity and high-performance algorithm for the fast correlation attack. In B. Schneier, editor, *Fast Software Encryption 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 196–212. Springer-Verlag, 2000.

[MFI02] M. J. Mihaljević, M. Fossorier, and H. Imai. Fast correlation attack algorithm with list decoding and an application. In M. Matsui, editor, *Fast Software Encryption 2001*, volume 2355, pages 196–210. Springer-Verlag, 2002.

[MG90] M. Mihaljević and J. Dj. Golić. A fast iterative algorithm for a shift register initial state reconstruction given the noisy output sequence. In J. Seberry and J. Pieprzyk, editors, *Advances in Cryptology—AUSCRYPT'90*, volume 453 of *Lecture Notes in Computer Science*, pages 165–175. Springer-Verlag, 1990.

[MHLL02] D. Moon, S. Hong, S. Lee, and J. Lim. Cryptanalysis of LILI-128 with overdefined systems of equations. In *Proceedings of KoreaCrypt'02, Seoul, KOREA*, 2002.

[Mih96] M. Mihaljević. A faster cryptanalysis of the self-shrinking generator. In J. Pieprzyk and J. Seberry, editors, *First Australasian Conference on Information Security and Privacy ACISP'96*, volume 1172 of *Lecture Notes in Computer Science*, pages 182–189. Springer-Verlag, 1996.

[MJ05] A. Maximov and T. Johansson. Fast computation of large distributions and its cryptographic applications. In B. Roy, editor, *Advances in Cryptology—ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 313–332. Springer-Verlag, 2005.

[MJB04] A. Maximov, T. Johansson, and S. Babbage. An improved correlation attack on A5/1. In H. Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography—SAC 2004*, volume 3357 of *Lecture Notes in Computer Science*, pages 1–18. Springer-Verlag, 2004.

[Moh01] T. T. Moh. On the method of XL and its inefficiency against TTM. Available at *http://eprint.iacr.org/2001/047/* (accessed August 18, 2003), 2001.

[MPC04] W. Meier, E. Pasalic, and C. Carlet. Algebraic attacks and decomposition of Boolean functions. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology—EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 474–491. Springer-Verlag, 2004.

[MR84] J. L. Massey and R. A. Rueppel. Linear ciphers and random sequence generators with multiple clocks. In T. Beth, N. Cot, and I. Ingemarsson, editors, *Advances in Cryptology—EUROCRYPT'84*, volume 209 of *Lecture Notes in Computer Science*, pages 74–87. Springer-Verlag, 1984.

[MS88] W. Meier and O. Staffelbach. Fast correlation attacks on stream ciphers. In C. G. Günter, editor, *Advances in Cryptology—EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 301–316. Springer-Verlag, 1988.

[MS89] W. Meier and O. Staffelbach. Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1(3):159–176, 1989.

[MS90] W. Meier and O. Staffelbach. Correlation properties of combiners with memory in stream ciphers. In I. B. Damgård, editor, *Advances in Cryptology—EUROCRYPT'90*, volume 473 of *Lecture Notes in Computer Science*, pages 204–213. Springer-Verlag, 1990.

[MS94] W. Meier and O. Staffelbach. The self-shrinking generator. In A. De Santis, editor, *Advances in Cryptology—EUROCRYPT'94*, volume 905 of *Lecture Notes in Computer Science*, pages 205–214. Springer-Verlag, 1994.

[MS01] I. Mantin and A. Shamir. Practical attack on broadcast RC4. In M. Matsui, editor, *Fast Software Encryption 2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 152–164. Springer-Verlag, 2001.

[MvOV96] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL, 1996. ISBN 0-849-38523-7.

[MW06] Mathworld: Binomial distribution, 2006. *http://mathworld.wolfram.com/BinomialDistribution.html* (accessed February 23, 2006).

[NES99] NESSIE: New European Schemes for Signatures, Integrity, and Encryption. Available at *http://www.cryptonessie.org* (accessed August 18, 2003), 1999.

[NW06] K. Nyberg and J. Wallén. Improved linear distinguishers for SNOW 2.0. In *Fast Software Encryption 2006*, Lecture Notes in Computer Science, pages 163–182. Springer-Verlag, 2006.

[Nyb92] K. Nyberg. On the construction of highly nonlinear permutations. In R. A. Rueppel, editor, *Advances in Cryptology—EUROCRYPT'92*, volume 658 of *Lecture Notes in Computer Science*, pages 92–98. Springer-Verlag, 1992.

[OI05] K. Okeya and T. Iwata. Side channel attacks on message authentication codes. In *European Workshop on Security and Privacy in Ad-hoc and Sensor Networks—ESAS'05*, pages 205–217, 2005.

[P. 04] P. Stănică, S. Maitra and J. Clark. Results on rotation symmetric bent and correlation immune Boolean functions. In B. Roy and W. Meier, editors, *Fast Software Encryption 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 161–177. Springer-Verlag, 2004.

[Pas03] E. Pasalic. *On Boolean Functions in Symmetric-Key Ciphers*. PhD thesis, Lund University, Department of Information Technology, P.O. Box 118, SE–221 00, Lund, Sweden, 2003. ISBN 91-7167-027-0.

[Pha05] R. C.-W. Phan. Advanced slide attacks revisited: Realigning slide on DES. In *Progress in Cryptology — Mycrypt 2005*, volume 3715 of *Lecture Notes in Computer Science*, pages 263–276, 2005.

[PK95] W. T. Penzhorn and G. J. Kühn. Computation of low-weight parity checks for correlation attacks on stream ciphers. In C. Boyd, editor, *Cryptography and Coding – 5th IMA Conference*, volume 1025 of *Lecture Notes in Computer Science*, pages 74–83. Springer-Verlag, 1995.

[PP03] S. Paul and B. Prenel. Analysis of non-fortuitous predictive states of the RC4 keystream generator. In T. Johansson and S. Maitra, editors, *Progress in Cryptology—INDOCRYPT 2003*, volume 2904 of *Lecture Notes in Computer Science*, pages 52–67. Springer-Verlag, 2003.

[PP04] S. Paul and B. Preneel. A new weakness in the RC4 keystream generator and an approach to improve the security of the cipher. In B. Roy and W. Meier, editors, *Fast Software Encryption 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 245–259. Springer-Verlag, 2004.

[Rao73] C. R. Rao. *Linear Statistical Inference and Its Applications*. Wiley, New York, NY, second edition, 1973. ISBN 0-471-70823-2.

[RC94] P. Rogaway and D. Coppersmith. A software-optimised encryption algorithm. In R. J. Anderson, editor, *Fast Software Encryption'93*, volume 809 of *Lecture Notes in Computer Science*, pages 56–63. Springer-Verlag, 1994.

[RH03] G. G. Rose and P. Hawkes. Turing: A fast stream cipher. In T. Johansson, editor, *Fast Software Encryption 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 290–306. Springer-Verlag, 2003.

[Rob] A. Robertson. Hypothesis testing examples, math 102 / Core 14. *http://math.colgate.edu/ aaron/Math102/HypTestExamples.pdf* (accessed 2006-03-21).

[Rob94] M. Robshaw. MD2, MD4, MD5, SHA and other hash functions. Technical Report TR 101, RSA Laboratories, July 1994.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *j-CACM*, 21(2):120–126, February 1978.

[Rue87] R. A. Rueppel. When shift registers clock themselves. In D. Chaum and W. L. Price, editors, *Advances in Cryptology—EUROCRYPT'87*, volume 304 of *Lecture Notes in Computer Science*, pages 53–64. Springer-Verlag, 1987.

[Saa02] M-J. O. Saarinen. A time-memory tradeoff attack against LILI-128. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 231–236. Springer-Verlag, 2002.

[Sch96] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley&Sons, New York, NY, 2nd edition, 1996. ISBN 0-471-11709-9.

[Sha49] C. E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 27:656–715, 1949.

[Sie84] T. Siegenthaler. Correlation-immunity of non-linear combining functions for cryptographic applications. *IEEE Transactions on Information Theory*, 30:776–780, 1984.

[Sie85] T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Transactions on Computers*, 34:81–85, 1985.

[SKE05] *Symmetric Key Encryption Workshop—SKEW'05*. Available at *http://www2.mat.dtu.dk/people/Lars.R.Knudsen/stvl/* (accessed August 6, 2005), 2005.

[SM00a] P. Sarkar and S. Maitra. Construction of nonlinear Boolean functions with important cryptographic properties. In B. Preneel, editor, *Advances in Cryptology—EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 485–506. Springer-Verlag, 2000.

[SM00b] P. Sarkar and S. Maitra. Nonlinearity bounds and construction of resilient Boolean functions. In M. Bellare, editor, *Advances in Cryptology—CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 515–532. Springer-Verlag, 2000.

[Sma03] N. Smart. *Cryptography: An Introduction*. McGraw-Hill Education, 2003. ISBN 0-077-09987-7.

[ST06] A. Shamir and E. Tromer. Acoustic cryptanalysis, 2006. *http://www.wisdom.weizmann.ac.il/ tromer/acoustic/* (accessed February 25, 2006).

[Sti95] D. R. Stinson. *Cryptography: Theory and Practice*. CRC Press, 1995. ISBN 1-584-88508-4.

[SZZ94] J. Seberry, X. M. Zhang, and Y. Zheng. Relationships among nonlinearity criteria. In A. De Santis, editor, *Advances in Cryptology—EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 376–388. Springer-Verlag, 1994.

[vS00] ETSI EN 300 963 v8.0.1 (2000-11) Standard. Digital cellular telecommunications system (Phase 2+) (GSM); Full rate speech; Comfort noise aspect for full rate speech traffic channels (GSM 06.12 version 8.0.1 Release 1999), 2000.

[Wag99] D. Wagner. The boomerang attack. In L. Knudsen, editor, *Fast Software Encryption'99*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer-Verlag, 1999.

[WBC03] D. Watanabe, A. Biryukov, and C. De Canniere. A distinguishing attack of SNOW 2.0 with linear masking method. In M. Matsui and R. Zuccherato, editors, *Selected Areas in Cryptography—SAC 2003*, volume 3006 of *Lecture Notes in Computer Science*, pages 222–233. Springer-Verlag, 2003.

[WFY+02] D. Watanabe, S. Furuya, H. Yoshida, K. Takaragi, and B. Preneel. A new keystream generator MUGI. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 179–194. Springer-Verlag, 2002.

[Wik06a] Wikipedia: Block cipher modes of operation, 2006. *http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation*.

[Wik06b] Wikipedia, the free encyclopedia, 2006. *http://en.wikipedia.org*.

[WSLM05] D. Whiting, B. Schneier, S. Lucks, and F. Muller. Phelix - fast encryption and authentication in a single cryptographic primitive. eSTREAM, ECRYPT Stream Cipher Project, Report 2005 / 020 (2005-04-29), 2005. *http://www.ecrypt.eu.org/stream*.

[WYY05a] X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In V. Shoup, editor, *Advances in Cryptology—CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer-Verlag, 2005.

[WYY05b] X. Wang, H. Yu, and Y. L. Yin. Efficient collision search attacks on SHA-0. In V. Shoup, editor, *Advances in Cryptology—CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, 2005.

[YWZW05] H. Yu, G. Wang, G. Zhang, and X. Wang. The second-preimage attack on MD4. In *Proceedings of the International Conference on Cryptology and Network Security—CANS'05*, pages 1–12, 2005.

[Zol04] B. Zoltak. VMPC one-way function and stream cipher. In B. Roy and W. Meier, editors, *Fast Software Encryption 2004*, volume 3017 of *Lecture Notes in Computer Science*, pages 210–225. Springer-Verlag, 2004.

[ZYR89] K. Zeng, C.-H. Yang, and T. R. N. Rao. On the linear consistency test (LCT) in cryptanalysis with applications. In G. Brassard, editor, *Advances in Cryptology—CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 164–174. Springer-Verlag, 1989.