Cryptanalysis of Selected Stream Ciphers

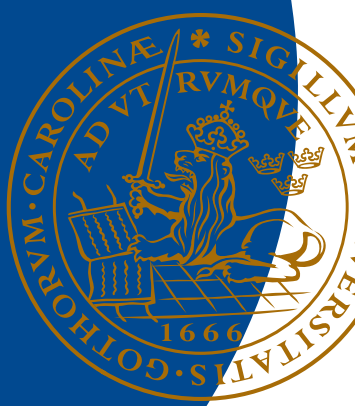
Stankovski, Paul

2013

Doctoral dissertation

Cryptanalysis of Selected Stream Ciphers

Paul Stankovski



Department of Electrical and Information Technology,
Faculty of Engineering, LTH, Lund University, 2013.

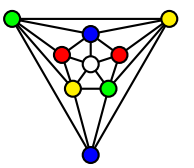
Cryptanalysis of Selected Stream Ciphers

Paul Stankovski

LUND UNIVERSITY

Doctoral Dissertation
Cryptology
Lund, June 2013

Paul Stankovski
Department of Electrical and Information Technology
Cryptology
Lund University
P.O. Box 118, S-221 00 Lund, Sweden



Series of licentiate and doctoral dissertations
ISSN 1654-790X; No. 50
ISBN 978-91-7473-526-0
Version 1.0
Compiled on May 23rd 2013

Typeset in Palatino and Helvetica using LaTeX 2ε, BibTeX and BibLaTeX.
Printed in Sweden by Tryckeriet i E-huset, Lund University, Lund.

Till Julie, Nike och Ella.

Bazinga!

Abstract

The aim of this dissertation is to show some cryptanalytical results on a selection of stream ciphers. We have grouped theory and results into three main parts.

The first part focuses on the FCSR-based constructions X-FCSR and F-FCSR-H v3. For the X-FCSR family of stream ciphers we perform a severe state recovery attack. This attack works for both X-FCSR-128 and X-FCSR-256.

We then develop a generalized birthday algorithm for finding linear relations in FCSRs. This algorithm applies to the most recent and general FCSR architecture, the ring FCSR, so it can be used for analyzing the FCSR of any FCSR-based design. We apply the algorithm to produce an efficient distinguisher for F-FCSR-H v3, which was previously unbroken.

The second part of the dissertation covers topics related to the HC family of stream ciphers. First, a very general treatment of sampling methods is presented. Surprisingly, perhaps, a positive result is given. We prove that an efficient sampling method based on sampling vector weights is optimal in a given context. This sampling technique is employed to produce the best known distinguisher for HC-128.

We go on to show a few theoretical results on functions that use word rotation and xor. These results are applied to a modified variant of HC-128, and this application shows how the theory could be used in a cryptanalytical scenario. It also shows the important role of the addition operator in HC-128, without which the cipher would be much less secure.

In the third part of the dissertation we analyze stream ciphers, and block ciphers to a lesser extent, using algebraic methods. We develop a simple and intuitive greedy algorithm for automatic security testing of cryptographic primitives. This is done in a black box fashion, without using any information

on the internal structure of the primitives. Despite this, it is shown how structural information is revealed very clearly under certain circumstances. The main features here are some nice results for the well-known stream ciphers Trivium, Grain-128 and Grain v1.

Contents

LIST OF DEFINITIONS

LIST OF THEOREMS

LIST OF ALGORITHMS

LIST OF FIGURES

LIST OF TABLES

Preface

This dissertation presents a highly filtered version of my research path and results at the department of Electrical and Information Technology at Lund University in Sweden. The treaded research path was once held in the dim light of uncertainty, but it is now exposed to full illumination. It is my hope that some of this light is reflected into and by this dissertation, which was derived from the publications listed below. The three main chapters, Chapters 3–5, were derived from [2,3,7], [1,4,5] and [6], respectively.

[1] P. Stankovski and M. Hell, »An Optimal Sampling Technique for Distinguishing Random S-boxes«, ISIT, July 2012, pp. 846–850, http://dx.doi.org/10.1109/ISIT.2012.6284680.

The author of this dissertation performed most of the analysis and wrote the paper.

[2] P. Stankovski, M. Hell and T. Johansson, »An Efficient State Recovery Attack on X-FCSR-256«, FSE 2009, ed. by O. Dunkelman, vol. 5665, Lecture Notes in Computer Science, Springer-Verlag, 2009, pp. 23–37, ISBN: 978-3-642-03316-2, http://dx.doi.org/10.1007/978-3-642-03317-9_2.

The author of this dissertation performed most of the analysis and wrote the paper, which was selected as one of the top three publications at FSE 2009, earning an invitation to publish in Journal of Cryptology.

[3] P. Stankovski, M. Hell and T. Johansson, »An Efficient State Recovery Attack on the X-FCSR Family of Stream Ciphers«, Journal of Cryptology, Online First™ (November 2012), pp. 1–22, ISSN: 0933-2790, http://dx.doi.org/10.1007/s00145-012-9130-9.

The author of this dissertation performed most of the analysis and wrote the paper. Extended version of [2].

[4] P. Stankovski, M. Hell and T. Johansson, »Analysis of Xorrotation with Application to an HC-128 Variant«, ACISP12, ed. by W. Susilo, Y. Mu and J. Seberry, vol. 7372, LNCS, Springer Berlin Heidelberg, 2012, pp. 419–425, ISBN: 978-3-642-31447-6, http://dx.doi.org/10.1007/978-3-642-31448-3_31.

The author of this dissertation performed most of the analysis and wrote the paper.

[5] P. Stankovski, S. Ruj, M. Hell and T. Johansson, »Improved Distinguishers for HC-128«, Designs, Codes and Cryptography 63 (Feb 2012), pp. 225–240, ISSN: 0925-1022, http://dx.doi.org/10.1007/s10623-011-9550-9.

The author of this dissertation and S. Ruj together performed most of the analysis and wrote the paper.

[6] P. Stankovski, »Greedy Distinguishers and Nonrandomness Detectors«, Progress in Cryptology—INDOCRYPT 2010, ed. G. Gong and K. C. Gupta, vol. 6498, Lecture Notes in Computer Science, Springer-Verlag, 2010, pp. 210–226, http://dx.doi.org/10.1007/978-3-642-17401-8_16.

The author of this dissertation is the sole author.

[7] H. Wang, P. Stankovski and T. Johansson, »A Generalized Birthday Approach for Efficiently Finding Linear Relations in ℓ-sequences«, accepted for publication in Designs, Codes and Cryptography on March 23rd, 2013.

The author of this dissertation and H. Wang together performed most of the analysis and wrote the paper.

The research included in the above publications and this dissertation was sponsored in part by the Swedish Research Council (Vetenskapsrådet) through grant 621-2006-5249.

Acknowledgments

Given the page count of the dissertation and the fact that it has taken some four odd years to produce it, one may say that the average output per day is not very impressive. However, the page count alone does not tell the whole story. While the dissertation itself is technical by nature, it would be unjust to let the human dimension go without mention. I will now try to remedy this situation by crediting the many people that in one way or another have contributed to the creation of this dissertation.

First and foremost, I want to thank my supervisor Thomas Johansson for believing in me when we first met, and for continuing to believe in me. Thomas is very generous with his vast knowledge and creative ideas, and his intuition is sharp to the point of making him seem almost clairvoyant at times.

I am also indebted to my assistant supervisor Martin Hell for always being there to explain, enthuse and encourage. His continuous support is immensely treasured.

Thomas and Martin H. have together managed to provide a wonderful study and work environment that is relaxed, stimulating and rewarding. This has made me look forward to going to work every day during the past four and a half years.

I want to thank Martin Ågren for being a very good friend, for the wonderful times that we had during our PhD studies, for proofreading the entire dissertation, for lots of things. I hope that we will get to experience many more adventures in the future.

Many thanks to Lennart Brynielsson, who took the time to comment on the statistics parts of the dissertation manuscripts.

Florian Hug provided the original LaTeX templates that were used for pre-

paring the dissertation, and they have been very useful. Florian was also part of the steady crowd of PhD students that helped generate the enjoyable atmosphere at the department. Many more deserve to be mentioned here. I thank you all for participating in the Wednesday fika group and Friday lunches, and for providing discussions, smiles and what not.

Many thanks to my coauthor and friend Hui Wang, whom I hope to revisit whenever opportunity arises.

The technical and administrative staff at the department also deserves my deepest gratitude for helping out in every way.

Mom and dad, thank you for your endless support and encouragement. You are the world's best parents!

My brothers Martin and Philip, thanks for always being there for me.

Last but not least, Julie, Nike and Ella. Nike and Ella, you have a healthy way of putting things in perspective. I wish you very healthy, long and happy lives, I love you!

Julie, none of this would have been possible without your support and encouragement, I love you!

Paul
Lund, May 2013

Scope

This dissertation is scoped in two ways; context and notation. Depending on the preferences of the reader, this approach may cause anything from pain and anxiety to joyous euphoria, so let us explain this. The ultimate purpose of doing this is simplification, both for the reader and the author.

We have divided the main material in this dissertation into three parts. The reader will not, as is customary in many other dissertations, find any introductory section with all the collected background material neatly compiled into a single chapter. Instead, we have aimed for scope.

The background material that is relevant for one specific chapter is placed in the beginning of that chapter. Plain and simple. We feel that this is a very natural approach that ties the contextually relevant material together for easy access.

The notation in this dissertation is also scoped. That is, we do not aspire to maintain a global notation that is valid throughout the body of this work. When you start reading a new chapter, variables and other notation from earlier chapters are purged and may be redefined in this or the coming chapters.

We suggest that you not only accept this notational scoping concept because you have to, which you do, but instead take the opportunity to also grow fond of it. This approach solves several problems.

It is, if not nearly impossible, at least difficult or cumbersome to maintain a perfectly coherent global notation scheme in a body the size of a book such as the one you are now holding in your hands.

We find notational deviance more annoying when a global scheme is implicitly or explicitly employed. Permitting reuse for common variable names, indices and such, eases the author burden, and can make the text much more

readable. The reader—you—will not need to resort to using all letters in the Greek alphabet or learning new and creative ways of indexing variables. Variable names will be more readable and less obscure.

However, some notation, the one given in Table 0.1, is guaranteed to be used in a uniform way through the dissertation, simply because it is global by choice or by nature.

Table 0.1: Global notation.

$\oplus$	xor.
$\|$	Context dependent. Concatenation operator *or* parameter separator for the divergence measure D.
$\|\cdot\|_1$	1-Norm, Hamming weight.
D	Divergence measure, Definition 2.4.
H	Entropy measure, Definition 2.1.
log	Logarithm base 2.
LSB	Least Significant Bit.
MSB	Most Significant Bit.
$\mathbb{N}$	The natural numbers.
φ	Euler totient function.
w_H	Hamming weight.
$\mathbb{Z}$	The integers.
$\mathbb{Z}^+$	The positive integers.
$\mathbb{Z}/2\mathbb{Z}$	The ring of integers modulo 2.
$\mathbb{Z}/n\mathbb{Z}$	The ring of integers modulo n.
$(\mathbb{Z}/n\mathbb{Z})^*$	The group of invertible elements in $\mathbb{Z}/n\mathbb{Z}$.
$\mathbb{Z}_2$	The ring of 2-adic integers, Definition 3.4.

1

Introduction

Cryptography
χρυπτός γράφειν

The word cryptography is derived from the greek words χρυπτός (kryptos) and γράφειν (graphein), meaning *hidden* and *writing*, respectively. In this context, hiding refers to concealing the content or the meaning of the message, not hiding the existence[1] of the message itself.

It is, of course, impossible to say when, how or why cryptography was first utilized, but it is conceivable that the need for hidden writing was spawned not long after writing itself was. We know that the Romans utilized a simple and, for its time, probably efficient substitution cipher, but the oldest documented findings of scrambled writing go as far back in time as to ancient Egypt in 1900 B.C. [WikiHis].

The field of cryptography has gone through a remarkable evolution since then. Unfortunately, we cannot cover this development in any detail, so we will fast-forward to modern times. We do not feel too bad about leaving the reader in a historical void, as some of this emptiness is brilliantly bridged by Simon Singh's »The Code Book« [Sin00].

[1] The art and science of hiding the existence of a message is called steganography, from the greek work στεγανός (steganos), meaning covered.

1.1 KERCKHOFFS'S PRINCIPLE

To us, modern times start in 1883 when Auguste Kerckhoffs publishes »La Cryptographie Militaire« in Journal des Sciences Militaires [Ker83]. In this paper, Kerckhoffs presents a set of rules that should be followed when designing new cryptographic systems. The most important one of these—Kerckhoffs's principle—states the following.

Kerckhoffs's principle: Il faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi.

Freely translated into English, it states that the security of a cryptosystem must not rely on keeping its design secret, as the cryptosystem can be stolen and analyzed (reverse-engineered) by the enemy. Instead, the security must lie in that the value of a key is kept secret.

The modern cryptographic society has adopted Kerckhoffs's principle as sound practice. The principle is very natural in modern terms, but it may well have been regarded with a fair amount of skepticism in the not-so-modern times. In the late 1800's, keeping a system secure meant keeping everything about it secret. With that mindset it may be quite difficult to realize that one can actually benefit from the openness that Kerckhoffs's analysis of military cryptosystems suggested.

The process of scrambling a readable *plaintext* is called *encryption*, and this process produces an unreadable *ciphertext*. The reverse *decryption* process recovers the plaintext from the ciphertext. An encryption *algorithm* is a step-by-step instruction that describes how to perform the actual scrambling, and an encryption algorithm typically takes a secret key as a parameter. The secret key determines exactly how the scrambling is performed. Note that the scrambling process must be reversible, in order for decryption to be possible, see Figure 1.1.

According to Kerckhoffs's principle, everything that there is to know about a well-designed encryption algorithm can very well be publicly announced, but the key that is used as a parameter for encryption must be kept secret.

Why is this principle so important? First of all, it simplifies the cryptosystem. Instead of keeping every part of a cryptographic communication system secret, we can design the system with openness in mind so that only a relatively small secret key needs to be handled with care.

Secondly, analysis and public scrutiny of an encryption algorithm may reveal potential security flaws in the design. If an algorithm is openly available for analysis and no serious flaws are found despite serious efforts, then this raises our confidence in the security level of the algorithm.

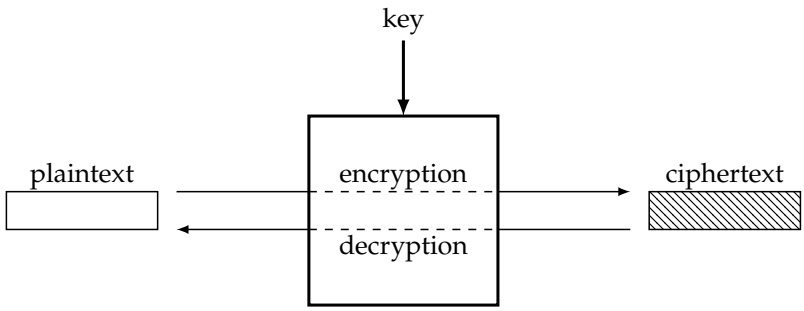


Figure 1.1: An encryption/decryption device.

Consider the opposing approach for a moment. A small group of experts have designed an encryption algorithm that they claim to be secure. The design is kept secret, so nobody can analyze the algorithm. Anyone who wants to use the encryption algorithm must trust the assertion of the expert group. Would you trust such a system to be secure? History tells us that you shouldn't. Even expert designers are human beings, and making mistakes is a human trait. A small group of experts can easily overlook some very important aspect of the design. After all, the designers must prevent all types of exploits on their design, while an attacker need only find one of potentially many weaknesses.

Thirdly, the arguably most important aspect of Kerckhoffs's principle is the effect that the encouragement of public cryptanalysis has. On a grander scale, continuous promotion of cryptanalysis drives and pushes the development of cryptographic designs so that we can expect to obtain better and better algorithms as the field of cryptology evolves.

Cryptology is the scientific study of cryptography and cryptanalysis. Cryptography is often taken to mean the art of encrypting and decrypting messages, which covers design of algorithms and protocols, and everything that relates to understanding construction and usage of cryptographic functions (or primitives). Cryptanalysis relates to breaking and understanding a cryptographic system and its underlying parts.

Something of a struggle has quite naturally developed between cryptographers and cryptanalysts, with each end trying to outsmart the other. Research in cryptology was, perhaps, particularly fueled by the invention and employment of the Enigma machine during World War II. It is not clear to which extent the cryptanalytical efforts targeted at the Enigma machine decided the outcome of the war, but most historians seem to agree that it did play a significant part.

1.2 ASYMMETRIC ENCRYPTION

Before the mid 70's, all encryption was *symmetric*. That is, the encryption algorithms used to encrypt and decrypt messages before that time all used a secret key as a parameter, and the same secret key was used at both ends of the communication. The key is used by the algorithm to determine the exact nature of the encryption process, and the sender and the receiver have to agree on a common secret key that is to be used at both ends.

In 1976, Diffie and Hellman [DH76] constructed a different kind of cryptosystem, one in which every participant has two keys, a public one used for encryption and a private one used for decryption—a key pair.

Consider the following problem. You and a friend are sitting and talking in a room full of people that quite rudely are listening in on your conversation, hearing every word you are saying. How can your friend pass a secret message to you without revealing it to any of the eavesdroppers?

You cannot use symmetric cryptography to solve this problem, because you would need to agree on which key to use. The eavesdroppers would learn which key you chose, so they can easily decrypt your communication. If we cannot use symmetric encryption, what can we use?

If you have never heard of asymmetric or public key cryptography before, it is very hard to solve this problem. As you have probably understood, asymmetric cryptography *does* solve the problem, and it does so in the following way. You first need to generate a public/private key pair. The public and private keys are mathematically linked so that encryption using the public key can only be decrypted using the private key.

You store the private key securely and reveal it to nobody. Your public key, however, can be publicly broadcast to anyone and everyone, friend and foe alike.

Your friend can now use your public key to encrypt a secret message, and only you can decrypt it using your private key. In fact, anyone can now send you a secret message by encrypting it with your public key. And if you want to send a message back to you friend, she needs to get a public/private key pair of her own so that you can use his public key.

Communication on the Internet resembles having a conversation in a room full of eavesdroppers. Using public key cryptography, a secure communication channel can be arranged despite the hostile environment.

1.3 WHAT IS SECURITY?

Early use of cryptography may quite often seem to have been of military nature, but the applications of today are naturally centered around computers and the Internet. We want our applications to be secure, but what does it mean

to be secure, what do we mean by security in this context?

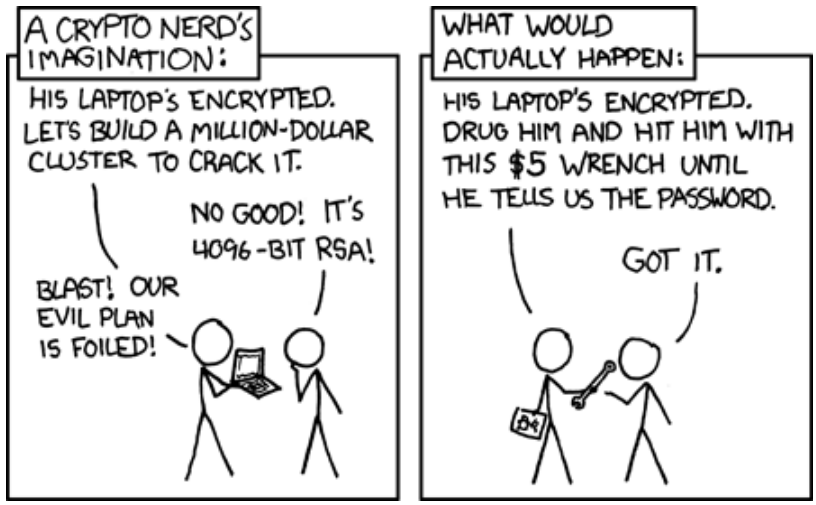


Figure 1.2: Security according to xkcd [xkcd].

It is impossible to give one clear cut definition of security. The notion of security is context dependent, but allow us to give a few examples.

1.3.1 UNCONDITIONAL SECURITY

A cryptosystem is unconditionally secure if an adversary with infinite computing power cannot break it. The term unconditional security is often regarded as synonymous to information-theoretical security, which, as the name suggests, is an information theoretical construct dating back to Shannon. A special case of it is called perfect security, which states that the ciphertext must not reveal any information about the plaintext, unless you have the key, of course.

1.3.2 PROVABLE SECURITY

In some situations it is possible to prove that a cryptosystem has certain desirable properties. This is called provable security. A number of different assumptions may be made for these proofs, but a proven property is the common factor.

For example, the security of some cryptosystems can be coupled with a difficult mathematical problem, so that breaking the cryptosystem relates to solving the difficult mathematical problem. The asymmetric encryption scheme RSA is one example of a cryptosystem that is provably secure, and in this case

it is provably secure with respect to a problem that is related to the difficulty of factoring large integers (see Section 2.1.3).

In other cases it may be possible to prove that a protocol is secure under the assumption that an underlying cryptographic primitive behaves perfectly. The random oracle model captures this concept of replacing a practical but imperfect primitive with a mathematically perfect version of it. The practicality of the resulting proof may be the question of some debate.

1.3.3 EMPIRICAL SECURITY

For a symmetric cipher with an n-bit key k, an attack is compared to the average number of keys that must be inspected when mounting an exhaustive key search, which is 2^{n-1}. A primitive is said to provide n bits of security if there exist no methods that are more efficient than an exhaustive key search. Of course, this nonexistence is usually impossible to prove, so one is often faced with the option of accepting a long term cipher as secure if no significant cryptanalytical advances have been made for some time.

Designers often incorporate an iterated procedure that can provide a security margin. For example, in stream ciphers it is common to define an initialization round, which parametrizes the entire cipher. Attacks can then be carried out on a reduced version of the cipher that is initialized using only a smaller number of initial rounds. We will see more of reduced-round primitives in Chapter 5.

1.4 STREAM CIPHERS

This dissertation will for the most part consider stream ciphers, so let us explain what these are.

There is a symmetric cryptosystem called the one-time pad (OTP), which was proven unbreakable[2] by Claude Shannon [Sha49] in 1949 using information theoretical arguments. Given a plaintext message p and a secret random key string z of the same length, the ciphertext c is given by

$$c = p \oplus z,$$

where $\oplus$ denotes bitwise xor. Here, we think of all plaintext, ciphertext and key entities as strings of bits—strings of zeros and ones. The bitwise xor operation is a very simple addition operation between two bits as shown in Table 1.1.

[2] Although proven unbreakable, it was broken in the movie Swordfish from 2001. For this feat we honor script writer Skip Woods with a cryptographic epic fail award.

Table 1.1: Binary addition table.

$\oplus$	**0**	**1**
0	0	1
1	1	0

The receiver uses the same secret random key string z to recover the plaintext by calculating

$$c \oplus z = p.$$

Why is it not used all the time if it is unbreakable? Because it requires the key length to be as large as the message itself. The problem of transferring a secret l-bit message to a given recipient is transformed into transferring an l-bit secret key, which is not much easier.

Stream ciphers provide a solution to this problem by mimicking the OTP. A stream cipher takes a secret n-bit key as input and generates a very long random-looking keystream sequence z. The keystream is then used as in the OTP to mask and unmask the message.

Stream ciphers have a fixed key length, typically in the range of 80–256 bits. However, since using the same key twice will output the same keystream, one must be careful with the usage. It is therefore common for a stream cipher to take a second parameter, an initialization value (IV). The secret key can then be reused if one chooses new randomly chosen IVs for every new encryption task, and these IVs do not need to be secret. The keystream generator paradigm is illustrated in Figure 1.3.

Stream ciphers are stream oriented, which means that they treat plaintext and ciphertext like a continuous stream of characters. For many applications, this approach is not only reasonable, but also desirable as it allows for very fast and efficient processing of large amounts of data in a very short time.

Also, stream ciphers are the main concern in this dissertation, but we will briefly acknowledge the existence of other cryptographic primitives in Section 2.1.

1.5 A CHALLENGING GAME OF STRUCTURE

Cryptanalysis often amounts to finding and exploiting structural relationships. Ciphers are designed to behave in a random-like fashion, so even if the structure is there, it is usually not a simple task to identify and analyze it.

Cryptanalysts search for any irregularity (bias) that they can find, and if

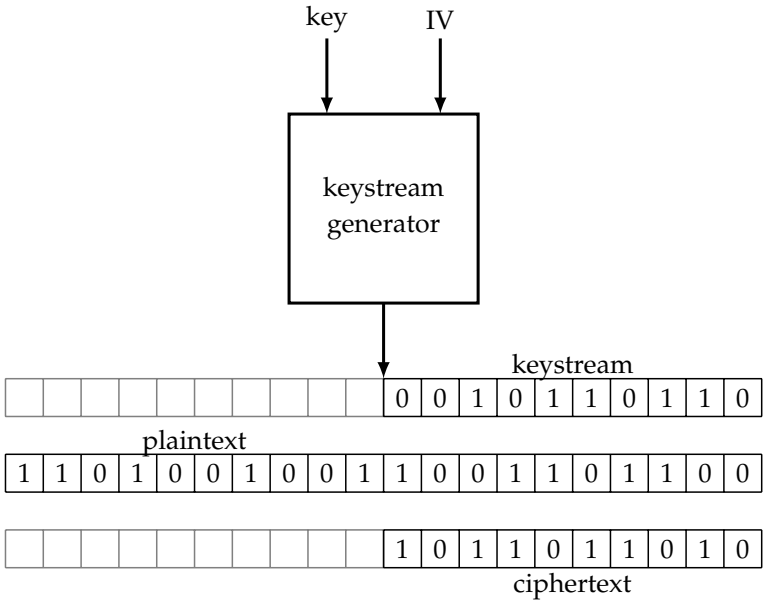


Figure 1.3: A keystream generator mimics the OTP.

the irregularity is significant enough, it may be possible to exploit it for an attack. In practice, most of the investigative time is spent staring at a design specification or possibly testing some code, but more often than not, you end up finding nothing at all. Many initial ideas that look good at first turn to dust in the wind. Once in a while, you do find a promising bias that turns out to be too small to be useful. Even rarer is finding a useful bias that holds up to produce an actual attack, but the holy grail of cryptanalysis is to find an exploit that completely breaks an entire encryption scheme in a fundamental way.

Imagine the moment in time when you go from having nothing and having had nothing for some time, to discovering a serious flaw in the design—a flaw that will break the system. Imagine the precise time instance at which you suddenly realize that the chaos you are observing suddenly displays very distinct and clear structural patterns, patterns that swiftly transform chaos into order.

Some of this feeling is beautifully illustrated by the following problem, which Prof. Daniel J. Costello Jr. challenged us with at ISIT 2012.

A prison has n inmates, which have been labeled 1 to n for our convenience. Once a day, the warden plays the following card game with the inmates. Using

a thoroughly shuffled deck of n cards labeled 1 through n, each inmate is, in turn, challenged to find the card with his own number on it by turning over at most half of the cards in the deck. If *all* inmates find their own card, then they are all released. If at least one prisoner fails to find his card, then they all stay in jail but get a new chance to play the following day.

Every inmate faces the exact same conditions; new random shuffle every day, the same shuffle (card order) for all prisoners, and no information may be passed along to the fellow inmates. The card selection process may be thought of as being performed in parallel, so that all inmates walk into one of n identically configured rooms. The prisoners may agree on a strategy that will maximize their success probability.

At first, it may seem like there cannot be an efficient strategy for solving this puzzle. It seems evident that the first prisoner stands a 50–50 chance to find his card. No information about the cards can pass from one inmate to the next, so it seems that the second inmate also has a 50–50 chance. And the third, and fourth, and so on. If each inmate independently at random chooses half of the cards in the deck, n inmates will achieve a success probability of $\left(\frac{1}{2}\right)^n$.

However, there is a better strategy, a much better one. We have already given the reader a hint, that this is a game of structure. A second hint is that the optimal success probability probably cannot be expressed as a nice and tidy formula for an exact analytical expression. But the success probability is huge, it is almost 30%!

Can you solve the riddle and find that strategy?

1.6 DISSERTATION OUTLINE

In Chapter 2, we introduce some very general and basic concepts that are common to all parts of the dissertation. Attack concepts, some information theory basics and hypothesis testing is described here.

The main theory and results are divided into three parts. Chapter 3 focuses on the FCSR building block in general, and on the FCSR-based constructions X-FCSR and F-FCSR-H v3 in particular. For the X-FCSR family of stream ciphers we perform a severe state recovery attack. This attack works for the two stream ciphers X-FCSR-128 and X-FCSR-256.

We then develop a generalized birthday algorithm for finding linear relations in FCSRs. This algorithm applies to the most recent and general FCSR architecture, the ring FCSR, so it can be used for analyzing the FCSR of any FCSR-based design. We apply the algorithm to produce an efficient distinguisher for F-FCSR-H v3, which was previously unbroken.

Chapter 4 covers topics related to the HC family of stream ciphers. First,

a very general treatment of sampling methods is presented. Surprisingly, perhaps, a positive result is given. We prove that an efficient sampling method based on sampling vector weights is optimal in a given context. This sampling technique is employed to produce the best known distinguisher for HC-128.

We go on to show a few theoretical results on functions that use word rotation and xor. These results are applied to a modified variant of HC-128, and this application shows how the theory could be used in a cryptanalytical scenario. It also shows the important role of the addition operator in HC-128, without which the cipher would be much less secure.

In Chapter 5 we analyze stream ciphers, and block ciphers to a lesser extent, using algebraic methods. We develop a simple and intuitive greedy algorithm for automatic security testing of cryptographic primitives. This is done in a black box fashion, without using any information on the internal structure of the primitives. Despite this, it is shown how structural information is revealed very clearly under certain circumstances. The main features here are very good results for the three well known stream ciphers Trivium, Grain-128 and Grain v1.

2

Technical Introduction

In this section we very briefly go through some basic concepts in cryptography and cryptanalysis. In Section 2.1 we categorize different cryptographic primitives. We explain what attacks are in Section 2.2 and attempt to provide something of an attack taxonomy, explaining which assumptions that one commonly makes for an attacker that is to break a given cryptosystem. Information theory basics is covered in Section 2.3 and, finally, in Section 2.4 we learn how to compare probability distributions by using the optimal hypothesis test.

2.1 CRYPTOGRAPHIC PRIMITIVES

We have already been acquainted with a few different classes of cryptographic primitives, but here is a very short rundown for future reference. It is not intended to be complete in any way, so we refer to »Handbook of Applied Cryptography« [MOV97] for further details.

2.1.1 UNKEYED PRIMITIVES

Unkeyed primitives are primitives that do not require a key. Cryptographic hash functions belong in this category. A hash function is a mapping that takes an arbitrary length string as input and produces a fixed length output. A reasonable hash function should strive to be preimage resistant, second preimage resistant, and collision resistant.

2.1.2 SYMMETRIC PRIMITIVES

Symmetric ciphers use the same secret key at both ends. There are three main classes of symmetric primitives; block ciphers, stream ciphers and message authentication codes (MACs). Block and stream ciphers are used for encryption/decryption, while MACs are used for integrity checks—to verify that a message has not been altered, accidentally or intentionally, during transport.

Block ciphers divide the plaintext into blocks of a fixed size and encrypt one block at a time. Padding is employed to handle plaintexts with lengths, and several different methods of chaining the blocks exist, but these technicalities are beyond the scope of this rudimentary summary. For a block cipher, the ciphertext must be available when decrypting.

Stream ciphers are stream oriented and do not need to divide the plaintext into fixed-size blocks, deal with padding or other such preprocessing duties. For some applications, these properties give significant performance advantages over block ciphers.

Also, stream ciphers differ from block ciphers in that the ciphertext must not be available[1] when decrypting. As stream ciphers generate keystream that is easily added with a simple bitwise xor during encryption and decryption, a decryption device could pregenerate very large amounts of keystream for the purpose of achieving a fantastic encryption speed when the application requires this.

To mention only two examples, stream ciphers are used in the background when we are using the GSM network, or when we look at encoded cable channels.

MACs can be thought of as keyed hash functions. If a sender sends a plaintext message (without encryption) together with a MAC value of this message, then the receiver can verify that the message indeed originates from the intended party, and that the message has not been tampered with. Note that this particular scheme provides authenticity and integrity, but not confidentiality.

2.1.3 ASYMMETRIC PRIMITIVES

The best known asymmetric primitive is probably RSA encryption, which is a provably secure scheme.

As mentioned before, an asymmetric scheme is used with a pair of matching keys, a public and a private one. The public key can be used to encrypt data, and decryption can only be performed using the matching private key. As always, it is crucial to store the private key securely so that it is not com-

[1] This is true for synchronous stream ciphers, for which the keystream depends only on the key and the IV. For asynchronous stream ciphers, the future keystream also depends on the previous keystream/ciphertexts.

promised by an attacker.

Digital signatures are another application of asymmetric primitives. Digital signatures can provide nonrepudiation, which is to say that a signer cannot later deny that she has produced that signature. For example, online banking services may use digital signatures when making transactions.

The provable security of RSA comes from assuming hardness of the so called RSA problem, which challenges a problem solver to recover a plaintext p from a ciphertext $c = p^e \bmod n$, where she is given the public RSA key (n, e) such that $n = pq$ where p and q are primes, $2 < e < n$, $\gcd(e, \varphi(n)) = 1$ and $0 \leq c < n$. This tautological security statement is related but not equivalent to the factoring problem—the problem of factoring very large integers into prime factors, which is also assumed to be a hard problem. It is easy to realize that the existence of an efficient factoring algorithm will break RSA encryption and solve the RSA problem. However, the implication in the other direction has not been proven by anyone. Whether an efficient algorithm for breaking RSA can be translated into an efficient factoring algorithm is an open question.

2.2 WHAT IS AN ATTACK?

We tend to be very generous with allowing the term attack to be used for most attempts at breaking a cryptographic primitive. If any nontrivial information is revealed in the attempted break, then we allow the term attack, even if the attack itself is more expensive than exhaustive search.

The cost of executing an attack can be measured in terms of the resources it requires. Several variants are common here. Time and storage are typical examples of such requirements, but some attacks also require availability of several plaintext/ciphertext pairs (typically for block ciphers), or availability of a certain amount of keystream (for stream ciphers). The fewer resources an attack requires, the better it is. Attacks may or may not be practical depending on their resource requirements.

We list four different types of attacks below; key recovery, state recovery, distinguishers and nonrandomness detectors. They are listed in decreasing order of strength—a key recovery attack is generally considered better than a state recovery attack, and so on.

2.2.1 KEY RECOVERY

A key recovery attack is a very strong type of attack, in which the attacker reconstructs the secret key using only keystream, or possibly several pieces of keystream generated by altering the public input variables (such as IV). Several rule variations are common here. In some of these, the attacker may

use several ciphertexts/plaintexts pairs, in others the IV values may be altered to produce different ciphertexts that can be analyzed.

One type of key recovery attack is the brute-force attack. If we are given one plaintext/ciphertext pair or some length of keystream, then we can iterate through all the possible keys to see which key was used during encryption. For a cipher with a k-bit key, we will find the correct key after we have tried about 2^{k-1} different keys. This time complexity is usually considered a baseline. Efforts that that take longer time than a brute-force attack are usually not dignified with the attack epitaph, while faster ones are.

2.2.2 STATE RECOVERY

A state recovery attack is less severe than a key recovery attack, but it is still a strong attack if it is performed faster than an exhaustive key search. Given some amount of keystream or other public information, it is the task of the attacker to reconstruct the internal state of the cryptographic primitive. If such a reconstruction is possible, the succeeding keystream sequence will be known from that point on.

2.2.3 DISTINGUISHERS

A simplified view of a distinguisher is that it is a decision mechanism that takes a sequence of n samples as input and outputs either CIPHER or RANDOM. The decision mechanism uses the n input samples, and possibly some public input parameters, to perform a hypothesis test to determine which of the two known probability distributions that is the most likely sample source, the cipher distribution or the uniform one. Lemma 2.1, the Neyman-Pearson lemma, provides the optimal hypothesis test.

However, apart from a given stretch of keystream, a distinguisher may also alter the public input variables of the given stream cipher and generate several related keystream sequences that can be analyzed together. In Section 5.2.5 we detail two different distinguishing models, taking advantage of this approach. The standard distinguishing model is described in Definition 5.7, and Section 5.8 describes the multiple choice distinguishing model.

Distinguishers are practical in the sense that it is possible to use a distinguisher in a real-life scenario to extract nontrivial information from the cipher. Also, distinguishers can sometimes be practical in the sense that they can be transformed into a full or partial key recovery attack.

2.2.4 NONRANDOMNESS DETECTORS

A nonrandomness detector is, like the distinguisher, a decision mechanism that outputs either CIPHER or RANDOM. However, the inputs of the nonrandomness detector are not limited to public inputs. It is also possible to

use, say, parts of the secret key as input. The general idea is that a cipher should in some sense behave randomly, specifically by outputting data that cannot be distinguished from a uniformly random source. The purpose here is to show nonrandomness. This will show that the cipher is not ideal, that it contains some structural flaw.

Nonrandomness detectors form the weakest kind of cryptanalytical result, but they operate at the edge of early weakness detection, which can be invaluable to an algorithmic designer. We will go into more details about this in Definition 5.4 in Section 5.2.2.

While it is certainly reasonable to group all related-key attacks into this category, our intention here is rather to introduce some continuity. A distinguisher allows modification of public input variables only, but we want to gradually loosen these conditions into including hidden input variables (such as the key) as well. Generalizing the distinguisher concept in this way means that we sacrifice practicality for earlier nonrandomness detection, but this is a reasonable tradeoff in our view.

2.3 INFORMATION THEORY BASICS

While we will be using the relative entropy measure in Definition 2.4 (below) for the most part, there are times when we will refer to the classical definition of entropy and conditional entropy. The following definitions are taken from [CT91].

Entropy is the information content (amount of randomness or uncertainty) of a single random variable.

Definition 2.1 (Entropy) Let X be a discrete random variable that takes values in $\mathcal{X}$ according to probability distribution $P_X(x) = \Pr\{X = x\}$. The entropy $H(X)$ of X is then given by

$$H(X) = -\sum_{x \in \mathcal{X}} P_X(x) \log P_X(x).$$

In the following, let X and Y be discrete random variables that take values in $\mathcal{X}$ and $\mathcal{Y}$ according to probability distributions $P_X(x) = \Pr\{X = x\}$ and $P_Y(y) = \Pr\{Y = y\}$, respectively.

The information content of a pair of random variables is given by their joint entropy, which we define as follows.

Definition 2.2 (Joint entropy) The joint entropy $H(X, Y)$ of a pair of discrete random variables (X, Y) with joint probability distribution $P_{X,Y}$ is given by

$$H(X, Y) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P_{X,Y}(x, y) \log P_{X,Y}(x, y).$$

We further define conditional entropy as follows.

Definition 2.3 (Conditional entropy) The conditional entropy $H(X|Y)$ of a discrete random variable X with probability distribution P_X is given by

$$\begin{aligned} H(X|Y) &= \sum_{y\in\mathcal{Y}} P_Y(y) H(X|Y=y) \\ &= -\sum_{y\in\mathcal{Y}} P_Y(y) \sum_{x\in\mathcal{X}} P_X(x|y) \log P_X(x|y) \\ &= -\sum_{y\in\mathcal{Y}} \sum_{x\in\mathcal{X}} P_{X,Y}(x,y) \log P_X(x|y). \end{aligned}$$

In loose terms, $H(X|Y)$ means the amount of information that the random variable X contains if one disregards the information overlap with Y.

Relative entropy captures the notion of distance between probability distributions.

Definition 2.4 (Relative entropy) The relative entropy between two probability distributions P_0 and P_1 over the same domain $\mathcal{X}$ is defined as

$$D\left(P_0 \| P_1\right) = \sum_{x\in\mathcal{X}} P_0(x) \log \frac{P_0(x)}{P_1(x)}. \tag{2.1}$$

Relative entropy has a couple of aliases in literature; information divergence, Kullback-Leibler divergence, information gain and redundancy.

Strictly speaking, the relative entropy measure does not qualify as a distance measure, since it is neither symmetric nor satisfies the triangle inequality. However, one reason for thinking of it as a distance between probability distributions will be given in Section 2.4, where we will see how we can use the divergence measure in practice to determine how many samples a distinguisher requires in order to correctly classify the samples that it analyzes. After all, the job of a distinguisher is to tell two probability distributions apart, to decide which of two probability distribution that the samples most likely are drawn from.

In the sequel, we will sometimes say 'the divergence of P' meaning $D(P\|U)$, where U denotes the corresponding uniform distribution. Also, please note that the symbol $\|$ is used for two different purposes in this dissertation. For the divergence operator D it is used as a parameter separator, which is classical usage in this regard, but otherwise it is to be interpreted as a concatenation operator. This ambiguity is resolved by the context.

Mutual information is the amount of information that two random variables share.

Definition 2.5 (Mutual information) The mutual information $I(X;Y)$ of two discrete random variables X and Y with probability distributions P_X and P_Y, respectively, and joint probability distribution $P_{X,Y}$ is given by

$$\begin{aligned} I(X;Y) &= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} P_{X,Y}(x,y)\log\frac{P_{X,Y}(x,y)}{P_X(x)P_X(y)} \\ &= D(P_{X,Y}\|P_XP_Y). \end{aligned}$$

Notice that it is possible to express mutual information in terms of divergence.

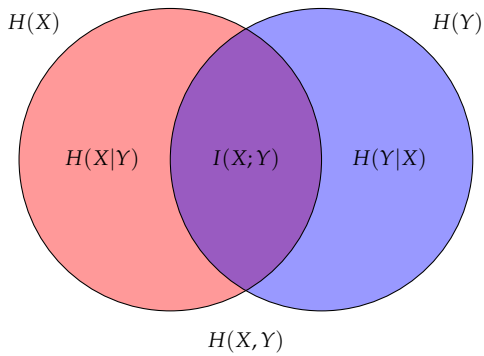


Figure 2.1: Entropy diagram.

The relationships between $H(X), H(Y), H(X|Y), H(X|Y)$ and $H(X,Y)$ are illustrated in Figure 2.1. Although we do not prove the following here, Figure 2.1 is meant to imply that

$$\begin{aligned} H(X) &= H(X|Y) + I(X;Y), \\ H(Y) &= \qquad\quad I(X;Y) + H(Y|X), \\ H(X,Y) &= H(X|Y) + I(X;Y) + H(Y|X). \end{aligned}$$

2.4 HYPOTHESIS TESTING

Hypothesis testing is central in cryptanalysis. In particular, we will need to perform hypothesis tests when we construct distinguishers. Neyman-Pearson provides the optimal hypothesis test.

Lemma 2.1 (Neyman-Pearson) Let the random variables $X_1, X_2, \ldots, X_n$ be independent and identically distributed according to mass function P_X. Consider the decision problem corresponding to the hypotheses $P_X = P_0$ vs.

$P_X = P_1$. For $T \geq 0$ define a region

$$\mathcal{A}_n(T) = \left\{ \frac{P_0(x_1, x_2, \ldots, x_n)}{P_1(x_1, x_2, \ldots, x_n)} > T \right\}. \tag{2.2}$$

Let $\alpha = P_0^n[\mathcal{A}_n^c(T)]$, where c denotes complement, and $\beta = P_1^n[\mathcal{A}_n(T)]$ be the error probabilities corresponding to the decision region $\mathcal{A}_n$. Let $\mathcal{B}_n$ be any other decision region with associated error probabilities α^* and β^*. If $\alpha^* \leq \alpha$, then $\beta^* \geq \beta$.

If all samples are independent, Equation (2.2) is equivalent to

$$\mathcal{A}_n(T) = \left\{ \sum_{i=1}^{n} \log \left(\frac{P_0(x_i)}{P_1(x_i)} \right) > \log T \right\}, \tag{2.3}$$

and this is the form that the Neyman-Pearson test takes in most applications. It is common to set $T = 1$, which indicates an unweighted comparison between the probability distributions P_0 and P_1.

The efficiency of a distinguisher is commonly expressed in terms of how much keystream it requires to make a correct decision with probability significantly larger than $\frac{1}{2}$, and it is in some sense here that the relative entropy measure enters the picture.

This type of hypothesis test models independent and identically distributed samples drawn from a probability distribution P_X. We have two possible hypotheses, the null hypothesis H_0 and the alternate hypothesis H_1;

$$H_0 : P_X = P_0,$$
$$H_1 : P_X = P_1.$$

The hypothesis test allows for two types of errors.

Type I error:	Reject H_0 when it is true (probability α).
Type II error:	Accept H_0 when H_1 is true (probability β).

As no universal expressions for α and β exist, we do not know the performance of the test in the general case. However, there are *asymptotic* expressions for the error probabilities. The asymptotic error probabilities are linked to the relative entropy through Stein's lemma, which roughly[2] states that β decreases so that

$$\lim_{n \to \infty} \frac{\log \beta}{n} = -D(P_0 \| P_1), \tag{2.4}$$

[2]The full story is actually a bit more involved, requiring double limits and the like, see [CT91].

if we fix the error probability α. Note that the value of α does not affect the exponential rate at which β decreases. Asymptotically we therefore have

$$\beta \approx 2^{-nD(P_0\|P_1)}, \tag{2.5}$$

so that we reach the point at which the error probabilities for the corresponding hypothesis test start to decrease exponentially when the number of samples n approaches

$$n \approx \frac{1}{D(P_0\|P_1)}. \tag{2.6}$$

We will use Equation (2.6) as a measure of the number of samples needed for our distinguishers in Chapter 4. This is the same measure that was used in [SRHJ12]. In particular, the divergence measure serves the purpose of comparing the performances of our distinguishers to previous ones. If we use Equation (2.6) for all cases the comparison is indeed fair.

We do emphasize, however, that a practical scenario would actually require us to use a small multiple of the number n as the number of samples needed by a distinguisher. The reason for this is that a practical application requires an error probability β that, in some sense, is small. Exactly how small β should be depends on the application, but there is a tradeoff between the error probability and the number of samples. The number n as defined in Equation (2.6) can be seen as a baseline requirement for the number of samples that a distinguisher needs.

There are other ways of estimating the number of required samples, and we refer to [BJV04, CT91, HJB09] for a more detailed treatment of this topic.

A practical application of Equation (2.6) is shown in Example 2.1.

Example 2.1 (A common rule of thumb) Consider the problem of distinguishing a probability distribution that is *almost* uniform. Let P be the binary probability distribution with bias ε according to

$$\begin{cases} P(0) = \frac{1}{2}(1+\varepsilon), \\ P(1) = \frac{1}{2}(1-\varepsilon), \end{cases}$$

for some small ε, and let U be the corresponding uniform distribution,

$$\begin{cases} U(0) = \frac{1}{2}, \\ U(1) = \frac{1}{2}. \end{cases}$$

A common rule of thumb used by cryptanalysts is that distinguishing a probability distribution with bias ε requires about

$$\frac{1}{\varepsilon^2}$$

samples. Using the probability distributions P and U above together with the divergence measure D, we will now show how one can motivate this rule of thumb.

The Taylor approximation

$$\ln(1+x) \approx x - \frac{x^2}{2}$$

is valid for small values of x, so according to Definition 2.4 we have

$$\begin{aligned} D(P\|U) &= P(0)\log\frac{P(0)}{U(0)} + P(1)\log\frac{P(1)}{U(1)} \\ &= \frac{1}{2}(1+\varepsilon)\log\frac{\frac{1}{2}(1+\varepsilon)}{\frac{1}{2}} + \frac{1}{2}(1-\varepsilon)\log\frac{\frac{1}{2}(1-\varepsilon)}{\frac{1}{2}} \\ &\approx \frac{1}{2\ln 2}\left((1+\varepsilon)\left(\varepsilon - \frac{\varepsilon^2}{2}\right) - (1-\varepsilon)\left(\varepsilon + \frac{\varepsilon^2}{2}\right)\right) \\ &\approx \frac{\varepsilon^2}{2\ln 2} \approx \varepsilon^2. \end{aligned}$$

Applying Equation (2.6) shows that the rule of thumb is reasonable.

3

Attacks on FCSRs and FCSR Based Constructions

This chapter deals with Feedback with Carry Shift Registers (FCSRs) and may be considered to present the main results in this dissertation. This part is derived from two articles.

The paper »An efficient state recovery attack on X-FCSR-256« [SHJ09], coauthored with Martin Hell and Thomas Johansson, was selected as one of the top three papers of FSE 2009. Based on the recommendation of the FSE program committee, it earned an invitation to Journal of Cryptology, where the extended version of the article was published as »An Efficient State Recovery Attack on the X-FCSR Family of Stream Ciphers« [SHJ12a] in 2012.

The second article is »A Generalized Birthday Approach for Efficiently Finding Linear Relations in ℓ-sequences« [WSJ13], which was coauthored with Hui Wang and Thomas Johansson. This article has been accepted for publication in the journal Designs, Codes and Cryptography.

3.1 INTRODUCTION

A common building block in stream ciphers is the Linear Feedback Shift Register (LFSR)[1]. The bit sequence produced by an LFSR has distributional properties (see [LN97, GK12]) that are very interesting for cryptographic applications. However, LFSRs are inherently linear, so additional building blocks are needed in order to introduce nonlinearity. For this reason, Nonlinear

[1] We assume that the reader is familiar with LFSRs, but if this is not the case, she may consult [WikiLFSR, LN97].

Feedback Shift Registers (NFSRs) are common components in modern stream ciphers. A special class of NFSRs are the FCSRs, which were introduced by Klapper and Goresky in [KG94]. FCSRs are very similar to LFSRs in many respects, and their intrinsic nonlinearity is due to the fact that FCSRs use integer addition with carries instead of the modular addition (xor) operator used in LFSRs. This property somewhat relaxes the nonlinearity requirements on other building blocks of cryptographic designs. The structural similarities between LFSRs and FCSRs imply that many of the good properties of LFSRs are inherited, but unlike most other NFSRs, FCSRs have a well-studied algebraic structure, making analysis much easier.

Two different flavors of FCSRs have been available in the past; the Fibonacci and Galois architectures [GK02, KG97, Kla05, GK12]. A new approach for FCSRs, ring FCSRs (or diversified FCSRs or ring representation), was proposed in [Arn+09], with an extended description of ring FCSRs being available in the journal version [ABP11]. The new ring FCSR automaton and its associated theory truly do unify the previous architectures, but more interestingly, it also generalizes the entire FCSR concept by adding flexibility. While Fibonacci and Galois FCSRs use many-to-one and one-to-many feedbacks, respectively, ring FCSRs allow many-to-many feedbacks, resulting in a generalized and unified theory.

The remainder of this chapter is divided into three parts. In Section 3.2 we briefly cover the basics of FCSR architectures, FCSR sequences and FCSR analysis. We also give a short description of the FCSR based stream ciphers that are our primary consideration here; X-FCSR-128, X-FCSR-256 and F-FCSR-H v3.

In Section 3.3 we go on to show how the X-FCSR family of stream ciphers can be severely broken with a state recovery attack. This part uses a technique—LFSRization—that was introduced in [HJ08] for breaking the eSTREAM [ECRb] final portfolio cipher F-FCSR-H v2 [ABL06]. After the publication of [HJ08], the F-FCSR-H v2 stream cipher was removed from the eSTREAM final portfolio. We show how to exploit LFSRization together with a few other tricks and techniques to break the X-FCSR family.

The creators of F-FCSR-H v2 soon updated the stream cipher to F-FCSR-H v3 [Arn+09], and the new cipher is now immune to the LFSRization technique. The update was quite nontrivial as the prior Galois FCSR was replaced with a ring FCSR. In Section 3.4, we show how to attack F-FCSR-H v3 as well, this time with a *completely* different approach. It is a generalized birthday[2] approach in which we aim at finding linear relations in FCSR sequences in a very efficient way. These linear relations are then used to build a distinguisher

[2]It is presented as a regular birthday approach [WikiBP] for simplicity, but it will become clear that a generalized birthday approach [Wag02] is applicable as well.

that breaks the exhaustive search time complexity limit. The new technique is actually very general. Even though it is applied to F-FCSR-H v3 here, the technique can be applied to any FCSR automaton, so linearly filtered FCSRs and FCSR combiners may be particularly interesting targets for future cryptanalysis.

We begin with some preliminaries on FCSRs.

3.2 FCSR PRELIMINARIES

An FCSR is a device that produces a binary expansion of a rational number p/q for some integers p and q, where q is odd. Traditionally, this device has been available in two different configurations; Fibonacci and Galois representation, named analogously to the corresponding LFSR representations that are very similar in structure. Both will be briefly reviewed here, along with the more general ring FCSR configuration, which generalizes the two previous architectures.

In FCSR based stream ciphers, p usually depends on the secret key and the initialization vector (IV), and q is a public parameter. The choice of q induces a number of FCSR properties of which the arguably most important property is that it completely determines the length of the period of the binary sequence that the FCSR outputs.

We now give a very brief overview of Fibonacci, Galois and ring FCSRs. We then introduce some relevant background information on FCSR sequences, and show the FCSR based stream ciphers X-FCSR-128, X-FCSR-256 and F-FCSR-H v3 that we will analyze later on.

3.2.1 FIBONACCI FCSR ARCHITECTURE

The state of a Fibonacci FCSR of size n consists of two principal parts, the main register $\mathbf{m} = (m_0, m_1, ..., m_{n-1})$, with $m_i \in \{0, 1\}$, and a memory register $b \in \mathbb{N}$. Let $\mathbf{m}(t)$ and $b(t)$ denote the state of the registers $\mathbf{m}$ and b at time t. State updates are performed according to

$$\begin{cases} m_i(t+1) = m_{i+1}(t), & \text{for } 0 \leq i \leq n-2, \\ \sigma(t+1) = b(t) + \sum_{i=1}^{n} q_i m_{n-i}(t), & \\ m_{n-1}(t+1) = \sigma(t+1) \bmod 2, & \\ b(t+1) = \sigma(t+1) \operatorname{div} 2. & \end{cases}$$

The auxiliary values $\sigma(\cdot)$ are used for clarity only.

Each FCSR is associated with a connection integer q, and for Fibonacci

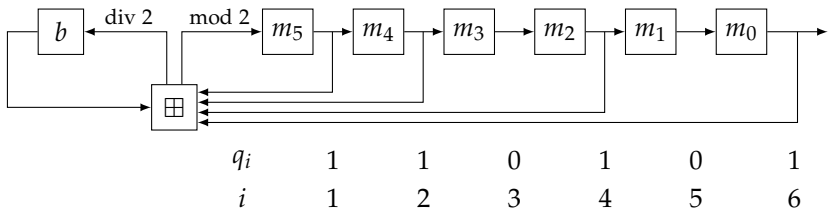


Figure 3.1: The Fibonacci FCSR with connection integer $q = 85$.

FCSRs this is defined as

$$q = q_n 2^n + q_{n-1} 2^{n-1} + \cdots + q_2 2^2 + q_1 2 - 1.$$

An example of a Fibonacci FCSR with $q = 85$ is given in Figure 3.1. It should be noted that the box labeled $\boxplus$ is a full adder that adds all inputs into an integer-valued sum σ. The value $\sigma \bmod 2$ is then fed into m_{n-1}, while σ div 2 is stored in the memory register b. The memory register b is a $(\lfloor \log(w_H(q+1)) \rfloor + 1)$-bit register, where w is the weight function that reports the number of ones in the binary expansion of a number (Hamming weight).

3.2.2 GALOIS FCSR ARCHITECTURE

The state of a Galois FCSR of size n also consists of two principal parts, the main register $\mathbf{m} = (m_0, m_1, ..., m_{n-1})$, as before, together with a carries register $\mathbf{c} = (c_0, c_1, ..., c_{n-2})$. We additionally let $\mathbf{c}(t)$ denote the state of the register $\mathbf{c}$ at time t.

As for Fibonacci FCSRs, define the odd connection integer as

$$q = q_n 2^n + q_{n-1} 2^{n-1} + \cdots + q_2 2^2 + q_1 2 - 1,$$

and let $d = \frac{q+1}{2} = (d_{n-1} \dots d_0)_{binary}$ according to

$$d = d_{n-1} 2^{n-1} + \cdots + d_2 2^2 + d_1 2 + d_0.$$

The carries register contains l active cells, where $l+1$ is the number of nonzero binary digits d_i in d. Disregarding d_{n-1}, which must always be one, the active carry cells correspond to the $d_i = 1$ in the interval $0 \leq i \leq n-2$.

We continue to let $\boxplus$ denote (regular) addition with carry, but in the Galois case we will only need one memory bit per carry, so all c_i's contain single bits. The $\boxplus$ operator now takes three inputs in total, two external inputs and the carry bit. For every clocking it first computes the (regular) sum σ of all three

input bits. It then feeds σ mod 2 into the succeeding main memory cell and stores the bit σ div 2 in the carry register. The state update mechanism can be summarized as follows.

$$\begin{cases} \sigma_i(t+1) = m_{i+1}(t) + c_i(t) + d_i m_0(t), & \text{for } 0 \le i \le n-2, \\ m_i(t+1) = \sigma_i(t+1) \bmod 2, & \text{for } 0 \le i \le n-2, \\ c_i(t+1) = \sigma_i(t+1) \text{ div } 2, & \text{for } 0 \le i \le n-2, \\ m_{n-1}(t+1) = m_0(t). \end{cases}$$

Following [ABL06], we specifically illustrate the case $q = 347$, which gives us $d = 174 = (10101110)_{binary}$, in Figure 3.2.

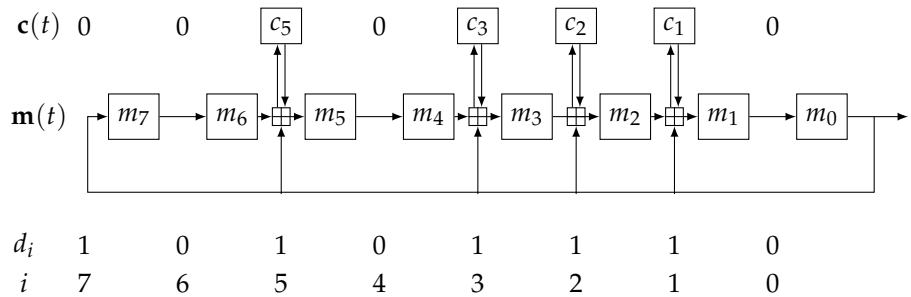


Figure 3.2: The Galois FCSR with connection integer $q = 347$.

3.2.3 RING FCSR ARCHITECTURE

Contrary to the Fibonacci and Galois representations, ring FCSRs allow full freedom in tap placement and can be seen as a generalization of the two previous architectures. Using main and carries registers $\mathbf{m} = (m_0, m_1, ..., m_{n-1})$ and $\mathbf{c} = (c_0, c_1, ..., c_{n-1})$ to describe the state of a ring FCSR of size n, updates are performed according to

$$\begin{cases} \boldsymbol{\sigma}(t+1) = \mathbf{T}\mathbf{m}(t) + \mathbf{c}(t), \\ \mathbf{m}(t+1) = \boldsymbol{\sigma}(t+1) \bmod 2, \\ \mathbf{c}(t+1) = \boldsymbol{\sigma}(t+1) \text{ div } 2, \end{cases}$$

where $\mathbf{T} = (t_{i,j})_{0 \le i,j < n}$ is the $n \times n$ transition matrix of the corresponding automaton graph. We have

$$t_{i,j} = \begin{cases} 1 & \text{if } m_j \text{ is used to update } m_i, \\ 0 & \text{otherwise.} \end{cases}$$

Definition 3.1 (Connection integer of a ring FCSR) Let $\mathbf{T}$ be the transition matrix of a ring FCSR. The connection integer q of that FCSR is then given by

$$q = -\det(\mathbf{I} - 2\mathbf{T}),$$

where $\mathbf{I}$ is the identity matrix of matching size.

Note that Definition 3.1, in contrast to the corresponding one in [Arn+09], defines q as a positive number.

As we have seen in Figures 3.1 and 3.2, there is an immediate structural interpretation of the value q (via d) for Fibonacci and Galois FCSRs. This is not obviously the case for ring FCSRs, as many different transition matrices produce the same connection integer. This means that there are many different ring FCSR configurations that can produce the same binary output sequence.

To obtain a ring FCSR with desirable properties, $\mathbf{T}$ should be chosen according to the guidelines listed in Section 5.1 in [Arn+09]. Following [ZD11], it is also possible to construct a transition matrix with a given connection integer.

A ring FCSR with transition matrix

$$\mathbf{T} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

and connection integer $q = 243$ is illustrated in Figure 3.3 with an equivalent alternative view in Figure 3.4.

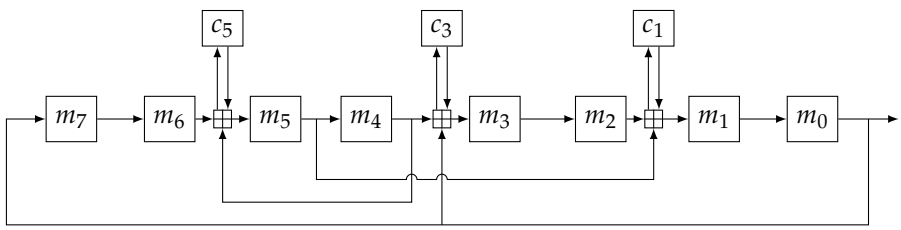


Figure 3.3: A ring FCSR with connection integer $q = 243$.

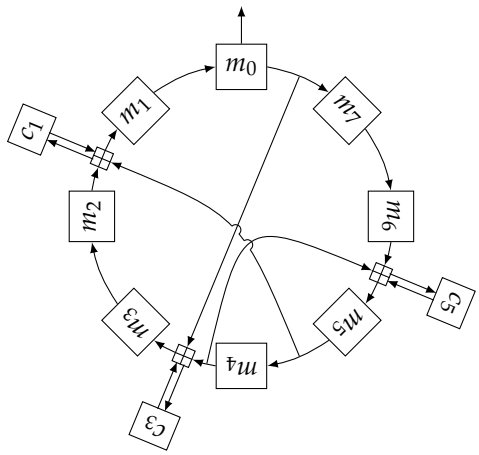


Figure 3.4: Alternative view of a ring FCSR with connection integer $q = 243$.

3.2.4 ℓ-SEQUENCES AND THE EXPONENTIAL REPRESENTATION

The definitive reference for FCSR theory is [GK12], which we refer to for further information if our simple and minimal introduction below should fall short in the eyes of our reader.

The output sequence $\underline{a} = \{a_t\}_{t=0}^{\infty}$ of an FCSR automaton is called an FCSR sequence. For all FCSR architectures we let the output bits be defined by the contents of the main register cell m_0 according to

$$a_{t+1} = m_0(t).$$

Let us first say something about the periodicity of FCSR sequences.

Definition 3.2 (FCSR sequence periodicity) An FCSR sequence $\underline{a} = \{a_t\}_{t=0}^{\infty}$ is *strictly periodic* if there exists some $L > 0$ such that $a_t = a_{t+L}$ for all $t \geq 0$. The sequence $\underline{a}$ is *eventually periodic* if there exists some $L > 0$ and $T > 0$ with $a_t = a_{t+L}$ for $t \geq T$. In both cases, the smallest such L is called the *period*. The sequence $\underline{a}$ is *periodic* if it is either strictly periodic or eventually periodic.

FCSR analysis relies to a great extent on properties of 2-adic integers.

Definition 3.3 (2-adic integer) A 2-adic integer is a formal power series over $\mathbb{Z}/2\mathbb{Z}$,

$$\sum_{i=0}^{\infty} a_i 2^i,$$

where the symbol 2 may be seen as a placeholder.

Note that all $a_i \in \mathbb{Z}/2\mathbb{Z}$, and that the series typically does not converge in the usual sense. Also, it is more convenient to use the symbol 2 as a placeholder rather than an indeterminate x, say, because of the way that operations on 2-adic integers behave.

We define operations to take carries into account. Addition is defined as follows. Let two 2-adic integers u and v be given. Then

$$u + v = \sum_{i=0}^{\infty} u_i 2^i + \sum_{i=0}^{\infty} v_i 2^i = \sum_{i=0}^{\infty} w_i 2^i = w$$

is to be interpreted via the unique solution to the equation system

$$\begin{cases} u_0 + v_0 = w_0 + c_0 2, \\ u_i + v_i + c_{i-1} = w_i + c_i 2, \quad i > 0, \end{cases} \tag{3.1}$$

where $w_i, c_i \in \mathbb{Z}/2\mathbb{Z}$ for all $i \geq 0$, and the c_i's may be regarded as carries.

Multiplication is defined correspondingly via the unique solution to the equation system

$$\begin{cases} u_0 v_0 = w_0, \\ \sum_{k=0}^{i} u_i v_{i-k} + c_{i-1} = w_i + c_i 2, \quad i > 0, \end{cases} \tag{3.2}$$

where $w_i \in \mathbb{Z}/2\mathbb{Z}$ and $c_i \in \mathbb{N}$ for all $i \geq 0$. The introduction of the two operators motivates the following definition.

Definition 3.4 (The ring $\mathbb{Z}_2$ of 2-adic integers) The 2-adic integers over $\mathbb{Z}/2\mathbb{Z}$ form a ring $\mathbb{Z}_2$ under addition and multiplication as defined by equation systems (3.1) and (3.2).

We ask the casual reader to note the difference between $\mathbb{Z}/2\mathbb{Z}$ and $\mathbb{Z}_2$. It may be noted that $\mathbb{Z} \subset \mathbb{Z}_2$, since the equality

$$-1 = 1 + 2 + 2^2 + 2^3 + \cdots$$

can be seen to hold in $\mathbb{Z}_2$ by adding one to both sides and propagating the carries to cancel all the terms in the sum. Furthermore, not all elements of $\mathbb{Z}_2$ are invertible, but the connection integers q that we will use are invertible when interpreted as 2-adic integers. With the preceding definitions and observations in mind, we hope that the reader is ready to accept that there is a correspondence between FCSR structure and 2-adic integers.

Theorem 3.1 (FCSR structure correspondence) Let $\underline{a} = \{a_t\}_{t=0}^{\infty}$ denote the output sequence of an FCSR with connection integer q. Then, $\underline{a}$ corresponds to a rational number $\beta = \frac{p}{q} \in \mathbb{Z}_2$. Specifically, $\sum_{t=0}^{\infty} a_t 2^t = \frac{p}{q}$ in $\mathbb{Z}_2$, which means that $\underline{a}$ can be interpreted as the 2-adic representation of β. The sequence $\underline{a}$ is strictly periodic if and only if $-q < p \leq 0$.

The same FCSR sequence can be generated by many different FCSRs with different connection integers, but we will abuse notation somewhat by letting the connection integer q of an FCSR sequence refer to the smallest possible such integer.

Sequences of maximal period are of special interest when constructing cryptographic primitives. For LFSRs, the maximal period sequences are called m-sequences (m for maximal). For FCSRs, they are called ℓ-sequences (ℓ for long). Definition 3.6 provides the specifics, but we first need to define primitive roots in Definition 3.5.

Definition 3.5 (Primitive root modulo q) Let $q = p^e$, where p is an odd prime and $e \geq 1$. A primitive root modulo q is a number $0 < g < q$ such that g generates the multiplicative group $(\mathbb{Z}/q\mathbb{Z})^*$.

That is, if $g = 2$ is a primitive root modulo $q = p^e$, then the sequence of numbers $2, 2^2, 2^3, \ldots, 2^{\varphi(q^e)}$ taken modulo q, where φ is the Euler totient function, runs through all invertible elements of $\mathbb{Z}/q\mathbb{Z}$. We can now define ℓ-sequences.

Definition 3.6 (ℓ-sequence) An ℓ-sequence $\underline{a}$ is an FCSR sequence with connection integer q for which $q = p^e$ ($e \geq 1$), p is an odd prime and 2 is a primitive root modulo q. The period of $\underline{a}$ is $\varphi(q)$.

The extreme case for an FCSR is therefore when q is an odd prime, for which the period is $\varphi(q) = q - 1$.

Last but not least, there is also an exponential representation of ℓ-sequences that is useful for FCSR analysis.

Theorem 3.2 (Exponential representation of ℓ-sequences) Given an ℓ-sequence $\underline{a} = \{a_t\}_{t=0}^{\infty}$ with connection integer q, there exists a unique integer $A \in \mathbb{Z}/q\mathbb{Z}$, such that

$$a_t = \alpha_t \bmod 2, \quad t \geq 0,$$

with $\alpha_t = (A2^{-t}) \bmod q$, where 2^{-1} denotes the multiplicative inverse of 2 in $\mathbb{Z}/q\mathbb{Z}$.

We will use this tool for proving specific properties of the linear relations that we encounter in Section 3.4.

3.2.5 X-FCSR-128 AND X-FCSR-256

The X-FCSR family of stream ciphers is specified in [ABLM07] and provides two stream cipher instances; X-FCSR-128 and X-FCSR-256. The designs are—as the name suggests—based on FCSRs, and they are targeted for software applications.

Both X-FCSR-128 and X-FCSR-256 admit a secret key of 128-bit length and a public IV of bitlength ranging from 64 to 128 as input. The core of the X-FCSR stream ciphers consists of two 256-bit Galois FCSRs with main registers Y and Z which are clocked in opposite directions.

$$Y(t) = (y_{t-255}, \ldots, y_{t-2}, y_{t-1}, y_t), \text{ clocked } \leftarrow$$
$$Z(t) = (z_{t+255}, \ldots, z_{t+2}, z_{t+1}, z_t), \text{ clocked } \rightarrow$$

At each discrete time instance t, Y and Z are used to form a 256-bit block $X(t)$ according to

$$X(t) = Y(t) \oplus Z(t),$$

where $\oplus$ denotes bitwise XOR, so that

$$\begin{aligned} X(0) &= (y_{-255} \oplus z_{255}, \ldots, y_{-2} \oplus z_2, y_{-1} \oplus z_1, y_0 \oplus z_0), \\ X(1) &= (y_{-254} \oplus z_{256}, \ldots, y_{-1} \oplus z_3, y_0 \oplus z_2, y_1 \oplus z_1), \\ X(2) &= (y_{-253} \oplus z_{257}, \ldots, y_0 \oplus z_4, y_1 \oplus z_3, y_2 \oplus z_2), \\ &\vdots \end{aligned}$$

X-FCSR-128 and X-FCSR-256 are identical up to this point, but they differ in the extraction function. Let us concentrate on X-FCSR-256 for a while. $X(t)$ is used as input to the extraction function. We define

$$W(t) = round_{256}(X(t)) = mix_{256}\left(sr_{256}\left(sl_{256}\left(X(t)\right)\right)\right), \tag{3.3}$$

where sl_{256}, sr_{256} and mix_{256} mimic the general structure of the AES round function;

sl is an S-box applied at the byte level,
sr is a row shifting function operating on bytes,
mix is a column mixing function operating on bytes.

The X-FCSR-256 round function operates on a 256-bit input, as defined in Equation (3.3). The general idea behind the round function operations becomes apparent if one considers how the functions operate on the 256-bit input when it is viewed as a 4×8 matrix $\mathbf{H}$ at the byte level. Let the byte entries of $\mathbf{H}$ be denoted $h_{i,j}$ with $0 \leq i \leq 3$ and $0 \leq j \leq 7$.

The first transformation layer consists of an S-box sl applied at the byte level. The chosen S-box has a number of attractive properties that are described in [ABLM07].

The second operation shifts the rows of $\mathbf{H}$, and sr is identical to the row shifting operation of Rijndael. That is, sr shifts (i.e., rotates) each row of $\mathbf{H}$ to

the left at the byte level, shifting the first, second, third and fourth rows 0, 1, 3 and 4 bytes respectively.

The purpose of the third operation, mix, is to mix the columns of $\mathbf{H}$. This is also done at the byte level according to

$$mix_{256}\begin{pmatrix} h_{0,j} \\ h_{1,j} \\ h_{2,j} \\ h_{3,j} \end{pmatrix} = \begin{pmatrix} h_{3,j} \oplus h_{0,j} \oplus h_{1,j} \\ h_{0,j} \oplus h_{1,j} \oplus h_{2,j} \\ h_{1,j} \oplus h_{2,j} \oplus h_{3,j} \\ h_{2,j} \oplus h_{3,j} \oplus h_{0,j} \end{pmatrix}$$

for every column j of $\mathbf{H}$.

Note that sl, sr and mix are all both invertible and byte oriented. Finally, the 256 bits of keystream that are output at time t are given by

$$out(t) = X(t) \oplus W(t-16). \tag{3.4}$$

This last equation introduces a time delay of 16 time units. The first block of keystream is produced at $t = 0$ and the key schedule takes care of defining $W(t)$ for $t < 0$.

X-FCSR-128 has a very similar extraction function, but it operates on a 128-bit block. If we by $X_L(t)$ and $X_R(t)$ denote the left and right parts of $X(t)$ according to $X(t) = X_L(t) \| X_R(t)$ where $\|$ denotes concatenation, form $\widetilde{X}(t) = X_L(t) \oplus X_R(t)$ and similarly define

$$W(t) = round_{128}\left(\widetilde{X}(t)\right) = mix_{128}\left(sr_{128}\left(sl_{128}\left(\widetilde{X}(t)\right)\right)\right), \tag{3.5}$$

and

$$out(t) = \widetilde{X}(t) \oplus W(t-16) \tag{3.6}$$

for X-FCSR-128. Now view the 128-bit block as a 4×4 matrix at the byte level. The row shifting function sr_{128} shifts the first, second, third and fourth rows by 0, 1, 2 and 3 bytes respectively, and the corresponding mix function uses the same matrix as above, but now with only four columns.

This description is sufficient for our purposes, but more information can be found in [ABLM07].

3.2.6 F-FCSR-H V3

The stream cipher F-FCSR-H v3 is a completely different stream cipher. Contrary to the X-FCSR family of stream ciphers, F-FCSR-H v3 is designed for hardware applications.

Table 3.1: The set S of nontrivial connections in the transition matrix of F-FCSR-H v3.

$(1,121)$	$(2,133)$	$(4,44)$	$(5,82)$	$(9,38)$	$(11,40)$
$(12,54)$	$(14,105)$	$(15,42)$	$(16,63)$	$(18,80)$	$(19,136)$
$(20,2)$	$(21,35)$	$(23,28)$	$(25,137)$	$(28,131)$	$(31,102)$
$(36,41)$	$(39,138)$	$(40,31)$	$(42,126)$	$(44,127)$	$(45,77)$
$(46,110)$	$(47,86)$	$(48,93)$	$(49,45)$	$(51,17)$	$(54,8)$
$(56,7)$	$(57,150)$	$(59,25)$	$(62,51)$	$(63,129)$	$(65,130)$
$(67,122)$	$(73,148)$	$(75,18)$	$(77,46)$	$(79,26)$	$(80,117)$
$(81,1)$	$(84,72)$	$(86,60)$	$(89,15)$	$(90,89)$	$(91,73)$
$(93,12)$	$(94,84)$	$(102,141)$	$(104,142)$	$(107,71)$	$(108,152)$
$(112,92)$	$(113,83)$	$(115,23)$	$(116,32)$	$(118,50)$	$(119,43)$
$(121,34)$	$(124,13)$	$(125,74)$	$(127,149)$	$(128,90)$	$(129,57)$
$(130,103)$	$(131,134)$	$(132,155)$	$(134,98)$	$(139,24)$	$(140,61)$
$(141,104)$	$(144,48)$	$(145,14)$	$(148,112)$	$(150,59)$	$(153,39)$
$(156,22)$	$(157,107)$	$(158,30)$	$(159,78)$		

F-FCSR-H v3 uses keys and *IV*s that are 80 bits long. The main component is a 160-bit ring FCSR with connection integer

$$q = 1741618736723237862812353996255699689552526450883$$

and transition matrix $\mathbf{T} = (t_{i,j})_{0 \le i,j < 160}$ with

$$t_{i,j} = \begin{cases} 1, & (i,j) \in S \text{ or } j \equiv i+1 \bmod 160, \\ 0, & \text{otherwise,} \end{cases}$$

where S is the set of 82 pairs given in Table 3.1.

Eight linear filters $F_0, \ldots, F_7$ are used. For every FCSR clocking, eight output bits are produced, one by each filter. Each filter produces the xor of the main memory cells whose indices are listed in Table 3.2. For example, filter F_0 produces output bit

$$m_1 \oplus m_{15} \oplus m_{28} \oplus m_{46} \oplus m_{59} \oplus m_{79} \oplus m_{93} \oplus m_{115} \oplus m_{128} \oplus m_{141} \oplus m_{158}.$$

Note that F_0 and F_1 xor 11 main register cells, while $F_2, \ldots, F_7$ only xor 10. The information given here is not complete, but it is sufficient for understanding what follows. Further details can be found in the specification [Arn+09].

Table 3.2: Linear filters used in F-FCSR-H v3.

F_0	1, 15, 28, 46, 59, 79, 93, 115, 128, 141, 158
F_1	2, 16, 31, 47, 62, 80, 94, 116, 129, 144, 159
F_2	4, 18, 36, 48, 63, 81, 102, 118, 130, 145
F_3	5, 19, 39, 49, 65, 84, 104, 119, 131, 148
F_4	9, 20, 40, 51, 67, 86, 107, 121, 132, 150
F_5	11, 21, 42, 54, 73, 89, 108, 124, 134, 153
F_6	12, 23, 44, 56, 75, 90, 112, 125, 139, 156
F_7	14, 25, 45, 57, 77, 91, 113, 127, 140, 157

3.3 AN EFFICIENT STATE RECOVERY ATTACK ON THE X-FCSR FAMILY OF STREAM CIPHERS

Consider the following problem for a stream cipher. Given the keystream that has been output so far, how can you efficiently and accurately predict what the future keystream is?

Solving this problem is considered a serious cryptanalytical break, as the encryption scheme is then compromised—all encrypted messages from that point on will now easily be decrypted.

A state recovery attack is one way of accomplishing this. The internal state of the stream cipher fully describes its operation and output, so we need to find a mapping from keystream to cipher states able of producing the given keystream. We will now show how this can be done for the X-FCSR stream cipher family.

3.3.1 BACKGROUND

Using FCSRs to generate sequences for cryptographic applications was initially proposed by Klapper and Goresky in [KG94]. Recently, several new constructions based on the concept of FCSRs have been proposed. The class of F-FCSRs, Filtered FCSRs, was proposed by Arnault and Berger [AB05, KG97]. These constructions were cryptanalyzed in [JM06], using a weakness in the initialization function. Also a time/memory tradeoff attack was demonstrated in the same paper.

Another similar construction targeting hardware environments is F-FCSR-H, which was submitted to the eSTREAM project [ECRb]. F-FCSR-H was later updated to F-FCSR-H v2 because of a weakness demonstrated in [JM05]. F-FCSR-H v2 was one of the four ciphers targeting hardware that were se-

lected for the final portfolio at the end of the eSTREAM project. Inspired by the success, Arnault, Berger, Lauradoux and Minier presented the X-FCSR construction described in Section 3.2.5, now targeting software implementations. The main idea was to use two Galois FCSRs instead of one, and to also include an additional nonlinear extraction function inspired by the Rijndael round function. Adding this would allow more output bits per register update and thus increase throughput significantly. According to the designers, X-FCSR-256 runs at 6.5 cycles/byte and X-FCSR-128 runs at 8.2 cycles/byte. As this is comparable to the fastest known stream ciphers, it makes them very interesting in software environments.

Here, we will describe the attack in [SHJ12a]. For the security of X-FCSR-256 and X-FCSR-128, we note that no other attacks faster than exhaustive key search have been published.

In [HJ08, HJ11] a new property inside the Galois FCSR was explored, namely that the update was sometimes temporarily linear for a number of clocks. This resulted in a very efficient attack on F-FCSR-H v2 and led to its removal from the eSTREAM portfolio.

The state recovery attack that we present here uses the observation in [HJ08, HJ11]. The fact that two registers are used, together with the extraction function, makes it impossible to immediately use this observation to break the ciphers. However, several additional nontrivial observations will allow a successful cryptanalysis. The keystream is produced using state variables 16 time instances apart. By considering consecutive output blocks, and assuming that the update is linear, we are able to partly remove the dependency of several state variables. A careful analysis of the extraction function then allows us to treat parts of the state independently and brute force these parts separately, leading to an efficient state recovery attack. It is shown that the X-FCSR-256 state can be recovered using $2^{44.3}$ output keystream blocks and a computational complexity of $2^{4.7}$ table lookups per output block on average. Note that table lookup operations are *much* cheaper than testing a single key. The corresponding figures for X-FCSR-128 are $2^{55.2}$ for the number of output keystream blocks with a computational effort of $2^{16.3}$ table lookups per block.

The remainder of this section is organized as follows. In Section 3.3.2, we describe the different parts of the attack. Each part of the attack is described in a separate subsection and in order to simplify the description we will deliberately base the attack on assumptions and methods that are not optimal for the cryptanalyst. Then, additional observations and more efficient algorithms are discussed in Sections 3.3.3 and 3.3.4, leading to a more efficient attack. We will use X-FCSR-256 as our basic case to show how the attack works in full detail. In Section 3.3.5 we show how to adapt the attack to X-FCSR-128. The attack is summarized with an algorithmic description in Section 3.3.6.

We will continue to use the notation that was introduced in Section 3.2.5. As

described there, the X-FCSR family of stream ciphers uses two Galois FCSRs at the core of their construction.

For the purposes of this section it is sufficient to recall the Galois FCSR automaton as described in Section 3.2.2. For ease of reading, we repeat Figure 3.2 in Figure 3.5, where the case $q = 347$, which gives us $d = 174 = (10101110)_{binary}$, is illustrated (following [ABL06]). We will rehash this example a few times to highlight some key points.

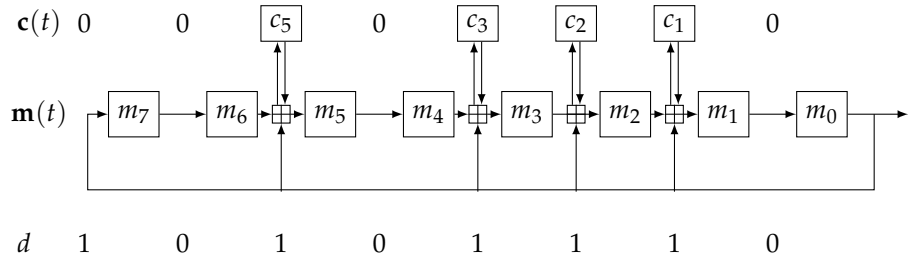


Figure 3.5: The Galois FCSR with connection integer $q = 347$.

3.3.2 ATTACK DESCRIPTION

Consider an FCSR automaton with n bits of memory in the main register and l bits in the carries register for a total of $n + l$ bits. If $(\mathbf{m}, \mathbf{c})$ is our *state*, then many states are equivalent in the sense that starting in equivalent states will produce the same output. As the period is at most $q - 1 \approx 2^n$, the number of states equivalent to a given state is in the order of 2^l.

A conceptual basis for understanding the attack is obtained by dividing it into the four parts listed below. Each part has been attributed its own section.

- LFSRization of FCSRs
- Combining Output Blocks
- Analytical Unwinding
- Solving for the State

In Section 3.3.2.1 we describe a trick we call *LFSRization* of FCSRs. We explain how the observation in [HJ08, HJ11] can be used to allow treating FCSRs as LFSRs. There is a price to pay for introducing this simplification, of course, but the penalty is not as severe as one may expect.

We observe that we can combine a number of consecutive output blocks to effectively remove most of the dependency on $X(t)$ introduced in Equation (3.4). The LFSRization process works in our favor here as it provides

a linear relationship between FCSR variables. Output block combination is explored in Section 3.3.2.2.

Once a suitable combination Q of output blocks is defined, state recovery is the next step. This is done in two parts. In Section 3.3.2.3 we explain how to work with Q analytically to transform its constituent parts into something that will get us closer to the state representation. As it turns out, we can do quite a bit here. Finally, we find that the state can be divided into several almost independent parts that can be treated separately. This is described in Section 3.3.2.4.

3.3.2.1 LFSRIZATION OF FCSRS

As mentioned above, an observation in [HJ08, HJ11] provides a way of justifying the validity in treating FCSRs as LFSRs, and does so at a very reasonable cost. We call this process LFSRization of FCSRs, or simply LFSRization[3] when there is no confusion as to what is being treated as an LFSR. There are two parts to the process, a flush phase and a linearity phase.

The observation is simply that a zero feedback bit in the Galois implementation of an FCSR, see Figure 3.5, causes the contents of the carry registers to change in a very predictable way. Adopting a statistical view and assuming independent events is helpful here. Assuming a zero feedback bit, carry registers containing zeros will not change, they will remain zero. The carry registers containing ones are a different matter, though. A one bit will change to a zero bit with probability $\frac{1}{2}$. In essence this means that one single zero feedback bit will cut the number of ones in the carry registers roughly in half.

The natural continuation of this observation is that a sufficient amount of consecutive zero feedback bits will eventually flush the carry registers so that they contain only zeros. On average, roughly half of the carry registers contain ones to start with, so an FCSR with N active carry registers requires roughly $\log \frac{N}{2} + 1$ zero feedback bits to flush the ones away with probability $\frac{1}{2}$. By expected value we therefore require roughly $\log \frac{N}{2} + 2$ zero feedback bits to flush a register completely. For the X-FCSR family we have $N = 210$, indicating that we need no more than nine zero feedback bits to flush a register.

After the flush phase, a register is ready to act as an LFSR. In order to take advantage of this state we need to maintain a linearity phase in which we keep having zero feedback bits fed for a sufficiently long duration of time. As we will see from upcoming arguments, we will in principle require the linearity property for two separate sets of five consecutive zero feedback bits,

[3]The term LFSRization can be seen as a gross understatement, as the resulting behavior is that of a simple shift of the main memory cells *without* any influence from a feedback function.

with the two sets being sixteen time units apart. We will need the FCSRs to act as LFSRs during this time, so our base requirement consists of two smaller LFSRizations. One LFSRization event requires roughly $9 + 5$ bits for flush and linearity phase, respectively, for an event probability of about $2^{-(9+5)}$, so two LFSRization events in the same register happen once in about $2^{2(9+5)} = 2^{28}$ clockings. The probability of two smaller LFSRizations occurring in *both* registers Y and Z simultaneously is therefore about $2^{-4(9+5)} = 2^{-56}$. In other words, our particular LFSRization condition appears once in about 2^{56} output blocks.

A real-life deviation from this theoretical flush reasoning was observed in [HJ08, HJ11]. We cannot flush the carry register entirely as the last active carry bit will tend to one instead of zero. As further noted in [HJ08, HJ11], flushing all but the last carry bit does not cause a problem in practice. Consider the maximally linearized FCSR in Figure 3.6, the state transitions of which are detailed in Table 3.3. It produces a maximal number of zero feedback bits for an FCSR of its size.

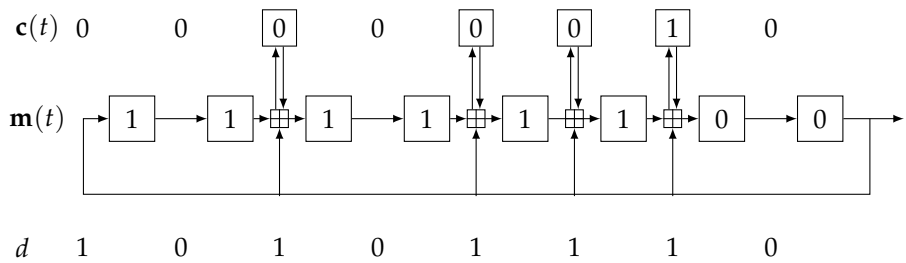


Figure 3.6: Maximally linearized FCSR outputting zero feedback bits.

In simulations and analytical work we must compensate for this effect, of course, but the theoretical reasoning to follow remains valid as we allow ourselves to treat FCSRs as simple LFSRs. The interested reader is referred to [HJ08, HJ11] for details on this part.

Furthermore, assumptions of independence are not entirely realistic. Although the theoretical reasoning above is included mainly for reasons of completeness, simulations show that we are not far from the truth, effectively providing some degree of validation for the theory. Our simulations show that the expected number of output blocks that we need to observe before the LFSRization effect appears twice is about $2^{28.0}$ for the Y register and $2^{26.0}$ for the Z register, for a total of about $2^{54.0}$ observed output blocks before LFSRization takes place in X-FCSR in the basic setting made explicit below.

Our requirements for the basic attack are summarized as follows. At some

Table 3.3: Consecutive states of the maximally linearized FCSR in Figure 3.6.

t	state	output bit
0	1 $1^0$1 $1^0$$1^0$$1^1$0 0	-
1	0 $1^0$1 $1^0$$1^0$$1^1$0 0	0
2	0 $0^0$1 $1^0$$1^0$$1^1$0 0	0
3	0 $0^0$0 $1^0$$1^0$$1^1$0 0	0
4	0 $0^0$0 $0^0$$1^0$$1^1$0 0	0
5	0 $0^0$0 $0^0$$0^0$$1^1$0 0	0
6	0 $0^0$0 $0^0$$0^0$$0^1$0 0	0
7	0 $0^0$0 $0^0$$0^0$$0^0$1 0	0
8	0 $0^0$0 $0^0$$0^0$$0^0$0 1	0
9	1 $0^0$0 $0^0$$0^0$$0^0$0 0	1

specific time instance we require the carry registers of Y and Z to be completely flushed except for the last bit. Here we also require the tails of the main registers to end with 11100 as in Figure 3.6 to guarantee at least five consecutive zero feedback bits for the five upcoming time instances. Sixteen time instances later we require this set-up to appear once again. In each flush-set, the five upcoming zero feedback bits ensure that the main registers remain linear.

In Table 3.4 we explicitly list the requirements for the Y register, with the requirements for the Z register defined correspondingly.

The X-FCSR family members output a block of keystream at each clocking, 128 and 256 bits for X-FCSR-128 and X-FCSR-256, respectively. Let the expected number of such output blocks (or FCSR clockings) for an attack to come through be denoted $COST_{keystream}$. To be fair and accurate we will use the simulation values, which puts us at

$$COST_{keystream} < 2^{54.0}$$

for the basic attack scenario with requirements **R1**–**R4** detailed in Table 3.4.

Later, in Sections 3.3.4.1 and 3.3.4.2, we will minimize keystream by relaxing requirements **R2** and **R4** to only three consecutive zero feedback bits, and in Section 3.3.4.3 we use a symmetry observation for a reduced keystream complexity of

$$COST_{keystream} < 2^{44.3}.$$

Table 3.4: Requirements for the Y register.

R1	At time $t-16$, the carry registers of Y are completely flushed except for the last bit.
R2	At least 5 consecutive zero feedback bits are output starting from time $t-16$.
R3	At time t, the carry registers of Y are completely flushed except for the last bit.
R4	At least 5 consecutive zero feedback bits are output starting from time t.

3.3.2.2 COMBINING OUTPUT BLOCKS

The principal reason for combining consecutive output blocks is to obtain a set of data that is easier to analyze and work with, ultimately leading to a less complicated way to reconstruct the cipher state. Remember that we now treat the two FCSRs as LFSRs with the properties given in Table 3.4.

The main observation is that the modest and regular clocking of the two main registers provides the following equality:

$$X(t) \oplus [X(t+1) \ll 1] \oplus [X(t+1) \gg 1] \oplus X(t+2) = (\star, 0, 0, \ldots, 0, \star) \quad (3.7)$$

The shifting operations $\ll$ and $\gg$ on the left hand side denote shifting of the corresponding 256-bit block left and right, respectively. From this point onward we discard bits that fall over the edge of the 256 bit blocks, and we do so without loss of generality or other such severe penalties. The right hand side is then the zero vector[4], with the possible exception of the first and last bits which are undetermined (and denoted $\star$). Define

$$OUT(t) = out(t) \oplus [out(t+1) \ll 1] \oplus [out(t+1) \gg 1] \oplus out(t+2) \quad (3.8)$$

in the corresponding way. Recalling from Equation (3.4) that

$$out(t) = X(t) \oplus W(t-16),$$

[4]Recall that we ignore the effects of the last carry bit being one instead of zero, as explained in Section 3.3.2.1. The arguments below are valid as long as adjustments are made accordingly.

we have

$$
\begin{aligned}
OUT(t) = & \\
& X(t) \oplus [X(t+1) \ll 1] \oplus [X(t+1) \gg 1] \oplus X(t+2) \oplus \\
& W(t-16) \oplus [W(t-15) \ll 1] \oplus [W(t-15) \gg 1] \oplus W(t-14) \\
& = \\
& (\star, 0, 0, \ldots, 0, \star) \oplus \\
& W(t-16) \oplus [W(t-15) \ll 1] \oplus [W(t-15) \gg 1] \oplus W(t-14) \\
& \approx \\
& W(t-16) \oplus [W(t-15) \ll 1] \oplus [W(t-15) \gg 1] \oplus W(t-14), \quad (3.9)
\end{aligned}
$$

where $\approx$ denotes bitwise equality except for the peripheral bits. This expression allows us to relate keystream bits to bits inside the generator that are just a few time instances apart. This will turn out to be very useful when recovering the state of the FCSRs. In order to further unwind Equation (3.9) we need to take a closer look at the constituent parts of W, namely the round function operations sl, sr and mix.

3.3.2.3 ANALYTICAL UNWINDING

Reviewing the round function operations from Section 3.2.5, recall that all of the operations are invertible and byte oriented. We can also see that the operations mix, sr and their inverses are linear over $\oplus$, such that

$$
\begin{aligned}
mix(A \oplus B) &= mix(A) \oplus mix(B), \\
sr(A \oplus B) &= sr(A) \oplus sr(B).
\end{aligned}
$$

Obviously, sl does not harbor the linear property. So, in order to unwind Equation (3.9) as much as possible, we would now ideally like to apply mix^{-1} and sr^{-1} in that order. Let us begin with focusing on the mix operation.

The linearity of mix over $\oplus$ is the first ingredient we need as it allows us to apply mix^{-1} to each of the W terms separately. The shifting does, however, cause us some problems since

$$
mix^{-1}\left(W(t) \ll 1\right) \neq mix^{-1}\left(W(t)\right) \ll 1.
$$

Therefore mix^{-1} cannot be applied directly in this way, but realizing that mix^{-1} is a byte-oriented operation, it is clear that the equality holds if one restricts comparison to every bit position except the first and last bit of every

byte. This is easy to realize if one considers the origin and destination byte of the six middlemost bits as mix^{-1} is applied. One single bit shift does not affect the destination byte of these bits. Furthermore, a peripheral bit that is shifted out of its byte position is mapped to another peripheral bit position. We therefore have

$$\begin{aligned} mix^{-1}\left(OUT(t)\right) \cong\ & sr\left(sl\left(X(t-16)\right)\right) \oplus \\ & \left[\ sr\left(sl\left(X(t-15)\right)\right) \ll 1\ \right] \oplus \\ & \left[\ sr\left(sl\left(X(t-15)\right)\right) \gg 1\ \right] \oplus \\ & sr\left(sl\left(X(t-14)\right)\right), \end{aligned}$$

where $\cong$ denotes equality with respect to the six middlemost bits of each byte. The same arguments apply to sr^{-1}, so we define

$$Q(t) = sr^{-1}\left(mix^{-1}\left(OUT(t)\right)\right) \tag{3.10}$$

to obtain

$$\begin{aligned} Q(t) \cong\ & sl\left(X(t-16)\right) \oplus \\ & \left[\ sl\left(X(t-15)\right) \ll 1\ \right] \oplus \\ & \left[\ sl\left(X(t-15)\right) \gg 1\ \right] \oplus \\ & sl\left(X(t-14)\right). \end{aligned}$$

Loosely put, we can essentially bypass the effects of the *mix* and *sr* operations by ignoring the peripheral bits of each byte.

We have combined consecutive keystream blocks $out(t)$ into Q in hope of Q being easier to analyze than $out(t)$. As our expression for Q involves only X and sl, let's see how and at what cost we can brute-force Q and solve for Y and Z.

3.3.2.4 SOLVING FOR THE STATE

In this section we outline the state recovery step. We proceed in a divide-and-conquer fashion by dividing the state into several almost independent parts and treat each part separately by solving a related equation system.

State solving can most easily be understood by focusing on one specific byte position in $Q(t)$. Given the, say, seventh byte in $Q(t)$, how can we *uniquely* reconstruct the corresponding parts of Y and Z? Let us first figure out which bits one needs from $Y(t-16)$ and $Z(t-16)$ in order to be able to calculate the given byte in $Q(t)$. Note that this step is possible only because of the LFSRization described in Section 3.3.2.1.

Consider the first part of Equation (3.10): $sl\,(X(t-16))$. Since sl is an S-box that operates on bytes, we need to know the full corresponding byte from $X(t-16)$. Those eight bits are derived from eight bits in each of Y and Z, totaling 16 bits, as shown in the left column of Figure 3.7.

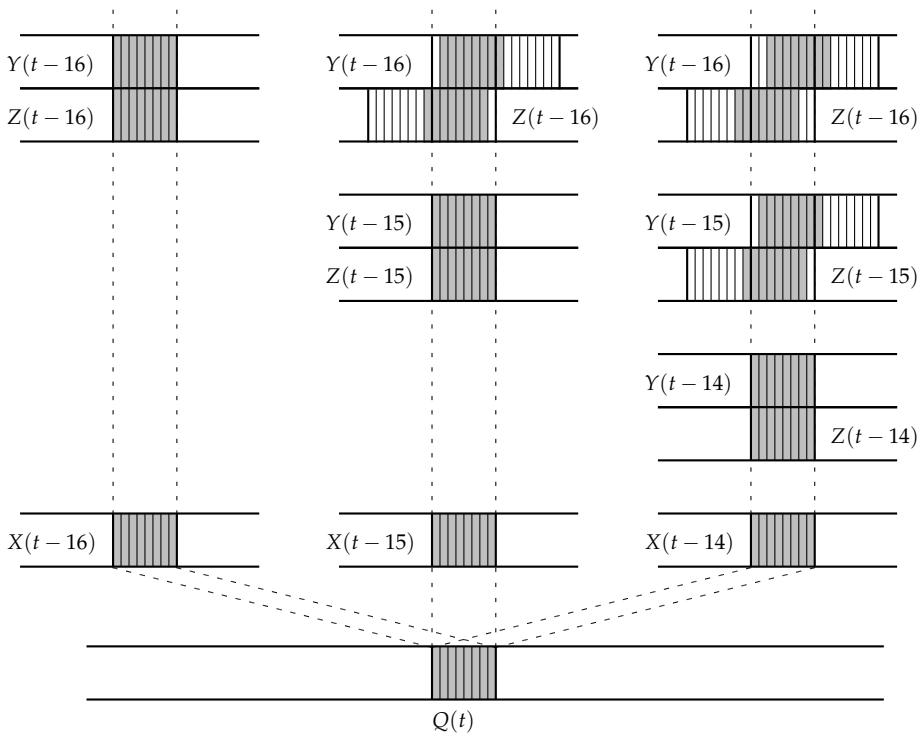


Figure 3.7: Bit usage for one byte in $Q(t)$.

The next parts of Equation (3.10) involves $sl(X(t-15))$. The same reasoning applies here. We need to know the full corresponding byte of $X(t-15)$ in order to be able to calculate this S-box value. But, since the main registers act like LFSRs, most of the bits we need from Y and Z for $X(t-15)$ have already been employed for $X(t-16)$ previously. Since the two main registers are clocked only one step at each time instance, only two more bits are needed, one from Y and one from Z. This is illustrated by the middle column of Figure 3.7. We count 18 bits in Y and Z so far.

In the same vein, two more bits are needed from Y and Z to calculate

$sl(X(t-14))$, illustrated in the remaining part of Figure 3.7. This brings the total up to 20 bits. All in all, for one byte position in $Q(t)$ we have total bit usage as shown in Figure 3.8.

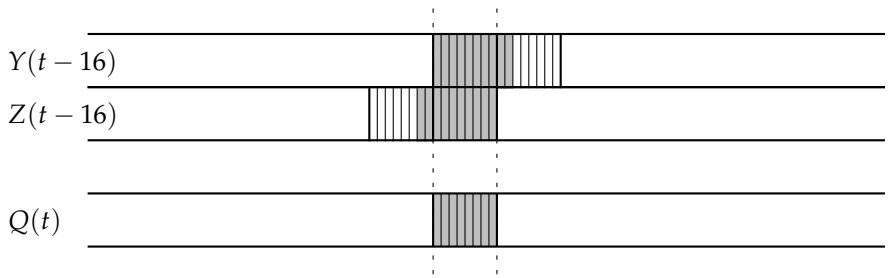


Figure 3.8: Bit usage in $Q(t)$.

So, 10 bits in $Y(t-16)$ and 10 bits in $Z(t-16)$ is what we require to be able to calculate one specific byte position in $Q(t)$. By restricting our attention to the six middlemost bits of each byte in Q we accomplish two objectives; we effectively reduce the number of unknown bits we are dealing with in Y and Z, and we simplify the expression for calculating the byte in Q by safely reducing the effects of the shifting operation. Specifically, shifting one bit left or right does not bring neighboring bytes into play.

Focusing on one single byte position gives us six equations, one for each of the six middlemost bits, and 20 unsolved variables, one for each bit position in Y and Z. This amounts to an underdetermined system, but we can easily add more equations by having a look at the same byte position in $Q(t+1)$. The six middle bits of that byte give us six new equations at the cost of introducing a few new variables. To see how many, we must perform the analysis for $Q(t+1)$ corresponding to Figure 3.7. The total bit usage for one byte position in $Q(t+1)$ in terms of bits in $Y(t-16)$ and $Z(t-16)$ is given in Figure 3.9.

From this we see that the six new equations have the downside of introducing two new variables. In total we therefore have 12 equations and 22 variables after including $Q(t+1)$, and 18 equations and 24 variables after including $Q(t+2)$. The corresponding bit usage for our three consecutive Q's in terms of bits in $Y(t-16)$ and $Z(t-16)$ is illustrated in Figure 3.10.

When solving one byte position in Q we essentially recover 24 bits. If we scan Q from left to right, solving the corresponding system for each byte, we can reuse quite many of these bits. Instead of solving for 24, we need only solve for 16 as the remaining 8 have already been determined. Thus, we actually have an overdetermined system with 18 equations and 16 variables. This is illustrated in Figure 3.11.

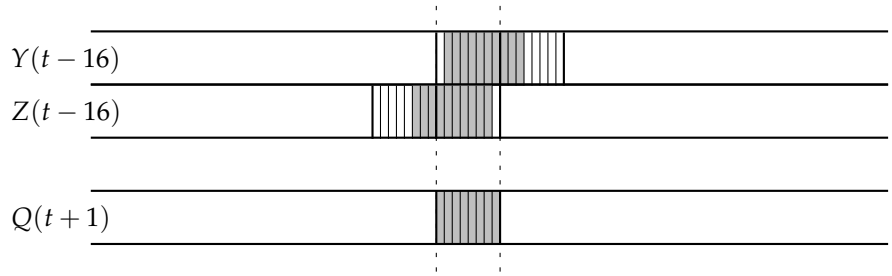


Figure 3.9: Bit usage in $Q(t+1)$.

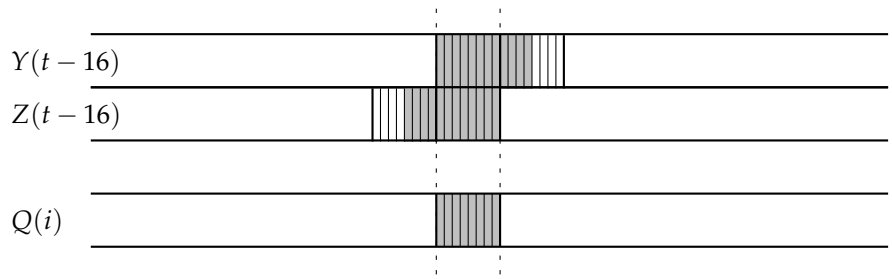


Figure 3.10: Total bit usage for $Q(i)$, $t \leq i \leq t+2$.

Reusing bits in this way works fine for all byte positions except the first one. For the first byte position we don't have any prior solution to lean back on, so at first glance it seems that this system is larger and thus more expensive to solve. In Section 3.3.3 we will explain what the first and last byte position systems look like in more detail, and we will see how to use the LFSRization assumption to reduce the system complexity in these cases.

As it turns out, the middle byte position systems are largest in terms of unsolved bits, which will dominate the worst-case cost of the equation solving part. Let $COST_{solver}$ denote the required number of variable assignments that must be tested for an attack to come through. Employing bit reuse, the *worst-case* cost for the solving part becomes

$$COST_{solver} < 32 \times 2^{16} = 2^{21}.$$

This concludes the principles of the basic attack, in which we have assumed availability of four separate sets of five consecutive zero feedback bits as described in Section 3.3.2.1. The only thing that remains is to calculate the solving complexity more rigorously. Using precomputed lookup tables and con-

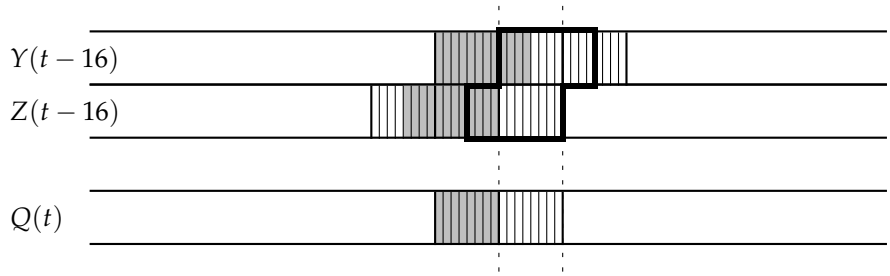


Figure 3.11: Reusing bits when solving for $Q(t)$.

sidering the expected-case complexity, we can significantly lower the cost for equation solving. This is what we will do in the following sections.

3.3.3 THE ANATOMY OF EQUATION SOLVING

In our attack scenario we wait for the first opportunity in which our keystream fulfills the requirements given in Table 3.4. For every block of keystream that is output, we try to solve for the state. Most times we fail, but our solver will find a solution when the requirements **R1**–**R4** have been met for registers Y and Z. Therefore, the average cost is more interesting from a practical perspective, so this is what we will compute next.

In Section 3.3.3.1 we warm up by finding the cost when precomputation is disallowed. In Section 3.3.3.2 we analyze the precomputation case, which concludes the basic attack on X-FCSR-256. We start by taking a closer look at the equation systems at different byte positions.

3.3.3.1 EQUATION SOLVING

Restating Equation (3.11), one may view the equation solving game as solving for the state at time $t-16$ given output at time t.

$$\begin{aligned} OUT(t) = & \\ & X(t) \oplus [X(t+1) \ll 1] \oplus [X(t+1) \gg 1] \oplus X(t+2) \oplus \\ & W(t-16) \oplus [W(t-15) \ll 1] \oplus [W(t-15) \gg 1] \oplus W(t-14) \\ & = \\ & (\star, 0, 0, \ldots, 0, \star) \oplus \\ & W(t-16) \oplus [W(t-15) \ll 1] \oplus [W(t-15) \gg 1] \oplus W(t-14). \end{aligned} \tag{3.11}$$

When solving for the state without precomputation, what we do in practice is to run through all unknown bits in $Y(t-16)$ and $Z(t-16)$ to see if we can find a configuration that produces the expected output. We do this byte by byte from left to right in the Y and Z registers for efficiency. Three byte position cases need to be considered; first, middle and last. The simplest case, the middle byte position case, is depicted in Figure 3.12.

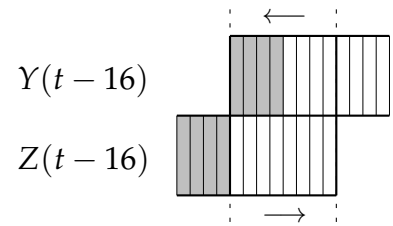


Figure 3.12: Equation system at middle byte positions.

We saw this system in Figure 3.11 before. The grayed bits are the ones we can reuse from solving the equation system from the preceding byte position, leaving 16 bits unsolved. Feedback bits do not affect the equation systems at middle byte positions.

The equation system for the first byte position is shown in Figure 3.13.

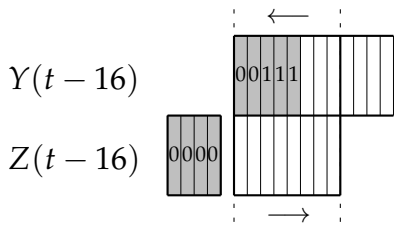


Figure 3.13: Equation system at first byte position.

As before, we have 24 variables and 18 equations. One difference is that 4 of the variables are new, having just entered the Z register. Another difference is that we cannot reuse variables from a prior solution. On the other hand we can use requirement **R2**. The last 5 bits of Y are known (00111), and the 4 bits entering Z are all zero.

Thus, for the first byte position system, 9 of the 24 bits are predetermined, leaving 15 bits unsolved.

The equation system at the last byte position mirrors that of the first, except that the bits from the previous byte position system are also given.

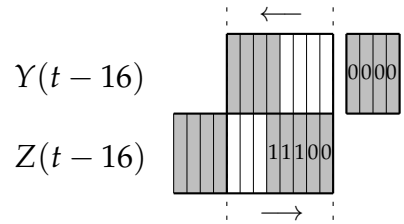


Figure 3.14: Equation system at last byte position.

Thus, for the last byte position system, 17 of the 24 bits are predetermined, leaving only 7 bits unsolved (Figure 3.14).

The amortized cost for attempting to solve for the entire state is then given by considering the relative frequencies of solving attempts per byte position. We process the byte positions from left to right in the natural way.

Using verification of Equation (3.11) as unit, the *expected*[5] cost for recovering the state is given by

$$COST_{solver} = 2^{15} + \frac{1}{8}\left(2^{16} + \frac{2^{16}}{4} + \frac{2^{16}}{4^2} + \cdots + \frac{2^{16}}{4^{29}} + \frac{2^7}{4^{30}}\right) < 2^{15.5}.$$

The factor $\frac{1}{8}$ is derived from the fact that we have 15 variables and 18 equations for the first byte position system. For the middle byte position systems we have 16 variables and 18 equations, producing the factor $\frac{1}{4}$ above.

3.3.3.2 EQUATION SOLVING WITH PRECOMPUTATION

The amortized cost for attempting to solve for the entire state using precomputation is, similarly, given by considering the relative frequencies of lookups per byte position. Instead of solving an equation system at each step, we look up the answer in a table. Letting $COST_{solver}$ denote the average number of required table lookups for a keystream block to be fully analyzed, we have

$$COST_{solver} = 1 + \frac{1}{8}\left(1 + \frac{1}{4} + \frac{1}{4^2} + \cdots + \frac{1}{4^{30}}\right) < 2^{0.3}.$$

Total computational complexity, i.e., the total number of table lookups, is given by

$$COST = COST_{keystream} \times COST_{solver}.$$

[5]It is also possible to reduce the size of the first equation system even further by using the zero feedback bits from the flush phase. That approach does produce a significant saving to, for example, $COST_{solver} < 2^8$ for eight flush-bits. As the number of bits needed to flush the carry register is unknown, this assumption may be false, leading to more keystream before the state can be recovered.

To see how the corresponding tables are constructed, once again consider Equation (3.10). We have 24 Y and Z variables that are combined into 18 Q values. As a conceptual starting point, make an auxiliary table A containing the corresponding Q values for all 2^{24} variable configurations. That is, table A has 2^{24} entries, each containing an 18-bit value.

The equation system for the first byte position has only 15 of the 24 Y and Z variables undetermined. Filtering out the corresponding 2^{15} entries from table A and making a reverse lookup hash table will do the trick. The hash table will be indexed on, at most, 2^{15} 18-bit Q values, and the entries will be the corresponding variable assignment (15 bits) for Y and Z.

For the middle byte position systems we correspondingly populate a hash table indexed on the 18-bit Q values *and* the eight known and reused variables. This table will contain 2^{24} entries, as we will use all of table A. Each entry will state the corresponding variable assignment (16 bits) for Y and Z.

Although it seems to be possible to use the table for the first byte position system for the last byte position by mirroring, this opportunity is destroyed by our upcoming minimizations of keystream requirements. Therefore, for the last byte position systems we construct a hash table indexed on the 18-bit Q values *and* the eight known and reused variables. This table will contain 2^{15} entries, where each entry represents the corresponding variable assignment (7 bits) for Y and Z.

In total, no more than 2^{25} table entries are needed, and each table entry fits well within a 32-bit word. The above numbers are possible to obtain in practice by employing, for example, cuckoo hashing (see [PR04]), which offers practical $O(1)$ lookups and amortized $O(1)$ insertions with $O(n)$ storage (all constants small).

This concludes the basic attack on X-FCSR-256 for which we have

$$COST_{keystream} < 2^{54.0}$$

and

$$COST_{solver} < 2^{0.3}$$

with 2^{25} precomputational storage. These numbers assume 5 feedback bits as described by the requirements stated in Table 3.4.

In the next section, our aim is to reduce the amount of required keystream.

3.3.4 REDUCING KEYSTREAM

We go on to reduce the keystream requirements by increasing the amount of equation solving. This is done in two steps. In Section 3.3.4.1 we see how the zero vector compensation from Equation (3.9) can be modified to allow for a

state recovery that is faster in terms of keystream complexity. The corresponding effort is then applied to the equation solving part, which will reduce the required keystream even further. This is examined in Section 3.3.4.2.

3.3.4.1 ZERO VECTOR COMPENSATION

We will now take a closer look at requirements **R3** and **R4** from Table 3.4. Referring to Equation (3.11) once more, one can see that the purpose of **R3** and **R4** is to make way for the X's to cancel out properly according to Equation (3.7). Requirement **R4** for the Y register dictates the behavior at one end of the vector, and that of the Z register controls the other.

If we relax **R4** from at least five consecutive zero feedback bits to precisely four, that fifth one feedback bit prohibits the X's from canceling out entirely. We can cope with this anomaly by compensating for such a non-null aggregate of the X's in Equation (3.11). The important issue is that we are in control of the resulting changes. As noted in Section 3.3.2.1, at least five consecutive zero feedback bits forces the tails of the main registers to contain the bit sequence $\ldots 11100$ as in Figure 3.6. To handle the case with *precisely* four consecutive zero feedback bits, one must compute the corresponding zero vector[6] for the five-bit tail $\ldots 01100$ and compensate accordingly. Solving for the state in the case with precisely four consecutive zero feedback bits amounts to solving a very similar equation system for the first and last byte position.

It is the tail of the Y register that determines the left end of the zero vector. The tail of the Z register determines the right end. It seems at first that we need to quadruple the computation to solve for all four variants. Taking the relative frequencies into account, the last byte position system is very cheap to solve. In fact, it comes almost for free. The modified cost for the case when we relax **R4** to at least four consecutive zero feedback bits is

$$COST_{solver} = 2\left(1 + \frac{1}{8}\left(1 + \frac{1}{4} + \frac{1}{4^2} + \cdots + \frac{1}{4^{29}} + \frac{2}{4^{30}}\right)\right) < 2^{1.3}.$$

To support our new solver we also need two new tables. These are of the same size as the previous ones for the first and last byte positions. The total space requirement therefore remains unchanged (at most 2^{25} table entries), since the storage requirement for the middle byte position systems dominates.

We take the procedure one step further and relax **R4** to only three consecutive zero feedback bits. This time we run into a complication. The bad news is that we get a one feedback bit for the last of the five output blocks. This triggers an additional summation at all carry cell positions, effectively

[6]The term zero vector may seem a little out of place as the vector is not all-zeros, but we appeal to the readers' idealizingly Platoesque nature.

pushing several ones into the carry vector. The problem with this is that the LFSRization effect is ruined, so we cannot hope to push the process even further to relax **R4** to only two consecutive zero feedback bits. For the three-case, however, we can still calculate a zero vector compensation and proceed as above.

Another positive note is that we do not need to consider both tail cases when we consider three consecutive feedback bits. We have covered the four-or-more case above with five-bit tails, and it remains to treat the precisely-three case. The tails of the registers must contain the bit sequence $\ldots 00100$ or $\ldots 10100$, when compared to Figure 3.6. Both tail sequences lead to the exact same zero vector compensation, so we only need to consider the 4-bit tail $\ldots 0100$. In terms of equation solving, this means that we have one less known variable for the first and last byte position systems. But the storage requirements are, as before, dominated by the middle byte position systems, so we may disregard the sizes of the first and last byte position systems. We cannot, however, disregard the equation system differences at the different middle byte positions, the differences imposed by the last feedback bit setting the many carries. The sizes of the systems remain the same, but we now have 30 different middle byte position systems, which increases memory usage for precomputation by a factor 2^5.

To summarize, we can recover the state also when **R4** is relaxed to three or more consecutive zero feedback bits. The three different tails and the possibility of the one feedback bit setting the many carries together generate six different equation systems for the first and last byte position. For the middle bytes there are four different systems. It is possible to recover the entire state with an expected

$$COST_{solver} < 6\left(1+\frac{1}{8}\left(1+\frac{1}{4}+\frac{1}{4^2}+\cdots+\frac{1}{4^{29}}+\frac{6}{4^{30}}\right)\right) < 2^{2.9}$$

table lookups into a precomputational storage of at most 2^{30} table entries. The interested reader is referred to [HJ08, HJ11], in which a similar situation is discussed.

Table lookups for the first byte position are most expensive as they occur most frequently. We may optimize the solver by merging all first byte position system tables. The cost of recovering the entire state is then reduced to

$$COST_{solver} < 1+\frac{6}{8}\left(1+\frac{1}{4}+\frac{1}{4^2}+\cdots+\frac{1}{4^{29}}+\frac{6}{4^{30}}\right) < 2^{1.1}$$

without increasing the storage requirements.

3.3.4.2 A SECOND REQUIREMENT RELAXATION

Having relaxed requirement **R4** to three consecutive zero feedback bits, we now turn the attention to requirement **R2**. Can we use the corresponding technique to relax **R2** to at least three consecutive zero feedback bits? This question is answered in the affirmative, and the corresponding cost of recovering the entire state is then

$$COST_{solver} < 1 + \frac{6^2}{8}\left(1 + \frac{1}{4} + \frac{1}{4^2} + \cdots + \frac{1}{4^{29}} + \frac{6^2}{4^{30}}\right) < 2^{2.9},$$

when both **R2** and **R4** are simultaneously relaxed. As before, the possible Z register tails at the last byte position are solved for at virtually no cost. The formula above indicates that we can treat the possible endings in each register as a separate system and create a separate table for each. For the middle byte positions there are, as before, four different systems. The size of the systems at middle byte positions dominates the storage requirements. We double our previous storage estimate to at most 2^{31} table entries.

3.3.4.3 FEEDBACK ONES—A SYMMETRY CASE

It is also possible to shorten the keystream requirement further by considering the symmetry case of several consecutive one feedback bits. Analogously to Figure 3.6, a maximally linear FCSR outputting consecutive ones for feedback bits is given in Figure 3.15.

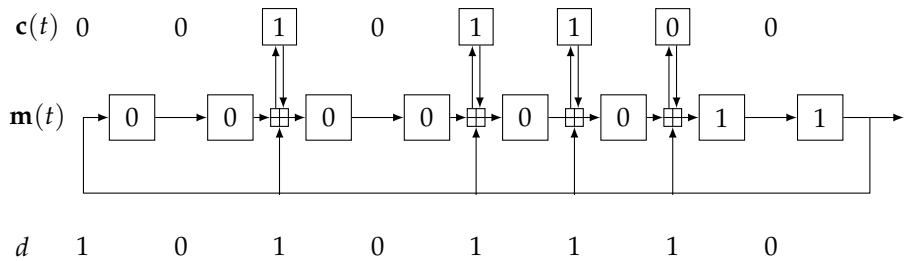


Figure 3.15: Maximally linearized FCSR outputting consecutive ones for feedback bits.

In the original case with zero feedbacks, we wait for the carries to be flushed in order for the FCSR to act linearly. In the conjugate case with one feedbacks, the same linear behavior appears when we have accumulated ones in the carries. Reviewing the entire methodology for the zero feedback case, one can see that the corresponding arguments and techniques hold when we are

Table 3.5: Costs for the X-FCSR-256 attack.

	Keystream	Solver	Storage
Basic attack without tables	$2^{54.0}$	$2^{15.5}$	–
Basic attack with tables	$2^{54.0}$	$2^{0.3}$	2^{25}
Reduced keystream attack according to Sections 3.3.4.1–3.3.4.3	$2^{44.3}$	$2^{4.7}$	2^{33}

facing one feedback bits as well. The only practical difference is that we alter the constants in the equation systems we are solving.

Instead of requiring simultaneous LFSRization with zero feedbacks in both Y and Z registers, we can relax our requirement to simultaneous LFSRization with zero *or* one feedbacks in each of Y and Z. Thus, by quadrupling the precomputational storage requirements and increasing the computational effort, we may reduce the amount of required keystream to one quarter using this additional observation.

To summarize again, with requirements **R2** and **R4** relaxed to at least three zero feedback bits and exploiting the symmetry ones case, we obtain

$$COST_{keystream} < 2^{44.3}$$

with

$$COST_{solver} < 1 + \frac{4 \cdot 6^2}{8}\left(1 + \frac{1}{4} + \frac{1}{4^2} + \cdots + \frac{1}{4^{29}} + \frac{4 \cdot 6^2}{4^{30}}\right) < 2^{4.7}$$

using precomputational storage of size 2^{33}.

This is our best result, both for minimizing the keystream requirement of the attack and for minimizing the total number of table lookups for recovering the state. The various costs are shown in Table 3.5.

3.3.5 X-FCSR-128

The LFSRization process is identical for both variants of X-FCSR, as is the analytical unwinding, leaving only the equation solving parts to be considered. In Equation (3.5) we can see that the 256-bit entity $X(t)$ is "folded" to produce a 128-bit result for X-FCSR-128. In effect, more state bits are condensed into one byte position of Q as analyzed in Section 3.3.2.4. This affects cost in a negative way, actually making the attack more expensive for X-FCSR-128. We are forced to solve larger equation systems to recover the state, so we need more Qs to increase the number of equations. The equation system for the

first byte position is illustrated in Figure 3.16 for the case when six Qs are used.

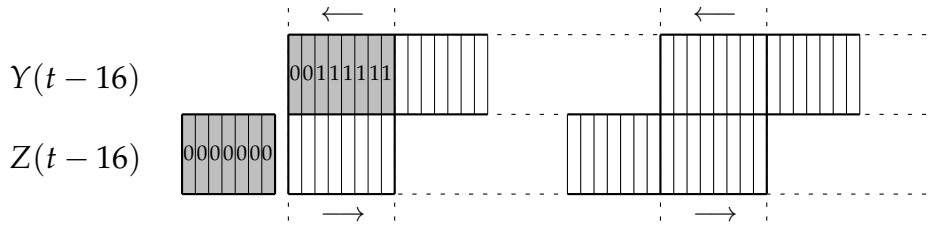


Figure 3.16: Equation system at first byte position (6 Qs).

This system is the largest with its 45 unknown variables. As before, the time complexity of state recovery is largely determined by the size of the middle byte position system. Regardless of how many Qs we use, this system has 32 unknown variables as depicted in Figure 3.17.

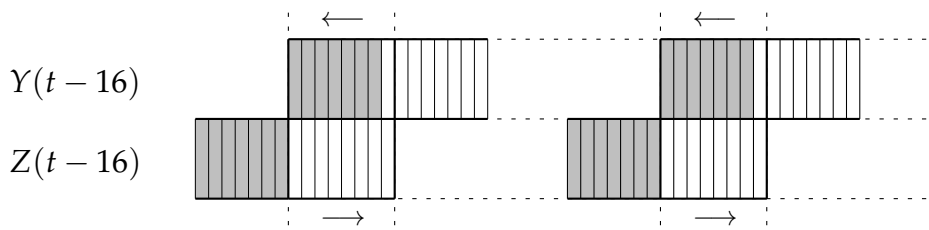


Figure 3.17: Equation system at middle byte positions (6 Qs).

Note that the six Qs induce 36 equations, leaving the first byte position system underdetermined by a factor 2^9, and the middle byte position systems overdetermined by a factor 16. The corresponding third byte position system is not illustrated, but it has 17 unsolved variables. Solving for the state without precomputation (compare to Equation (3.3.3.1)) therefore costs

$$COST_{solver} = 2^{45} + 2^9 \left(2^{32} + \frac{2^{32}}{16} + \frac{2^{32}}{16^2} + \cdots + \frac{2^{32}}{16^{13}} + \frac{2^{17}}{16^{14}} \right) < 2^{45.1},$$

where the factors 2^9 and $\frac{1}{16}$ are derived from the over- and underdeterminedness of the respective systems.

The time complexity of recovering the state using six Qs *with* precomputation is given by

$$COST_{solver} = 1 + 2^9 \left(1 + \frac{1}{16} + \frac{1}{16^2} + \cdots + \frac{1}{16^{13}} + \frac{1}{16^{14}} \right) < 2^{9.1}$$

Table 3.6: Costs for the X-FCSR-128 attack.

	Keystream	Solver	Storage
Basic attack without tables	$2^{64.0}$	$2^{45.1}$	–
Basic attack with tables	$2^{64.0}$	$2^{9.1}$	2^{60}
Reduced keystream attack according to Sections 3.3.4.1–3.3.4.3	$2^{55.2}$	$2^{16.3}$	2^{67}

in the basic setting with no relaxation of requirements **R2** and **R4**. The precomputational storage is now 2^{60}, again dominated by the middle byte position system.

We minimize $COST_{keystream}$ by using six Qs. With requirements **R2** and **R4** relaxed to at least three zero feedback bits, and exploiting the symmetry ones case, we obtain

$$COST_{keystream} < 2^{55.2}$$

with

$$COST_{solver} = 1 + 4 \cdot 6^2 \cdot 2^9 \left(1 + \frac{1}{16} + \frac{1}{16^2} + \cdots + \frac{1}{16^{13}} + \frac{4 \cdot 6^2}{16^{14}}\right) < 2^{16.3},$$

using precomputational storage of size 2^{67}. The corresponding cost table for X-FCSR-128 is given in Table 3.6.

3.3.6 SUMMING UP THE ATTACK

The results have been verified with simulations. Specifically, for X-FCSR-256 we have successfully recovered the entire state for all variations on the requirement set {**R1**,**R2**,**R3**,**R4**} discussed above.

The total cost for state recovery in terms of table lookups is given by

$$COST = COST_{keystream} \times COST_{solver}.$$

To summarize, we have $COST < 2^{44.3+4.7} = 2^{49.0}$ for X-FCSR-256 using at most 2^{33} table entries of precomputational storage. This attack variant minimizes both keystream and total complexity. The corresponding cost for X-FCSR-128 is $COST < 2^{55.2+16.3} = 2^{71.5}$ using at most 2^{67} storage.

Judging by the authors' performance figures in the specification, one X-FCSR initialization corresponds to outputting about $13.1 = 2^{3.7}$ or $20.9 = 2^{4.4}$ output blocks for X-FCSR-256 and X-FCSR-128, respectively. That is, if we count the cost in terms of initializations we have

$$COST_{initializations} = COST_{keystream} \times 2^{\alpha},$$

Algorithm 1 – X-FCSR Precomputation

Output: Lookup table sets $T_1, \ldots, T_n$, one for each byte position $i = 1, 2, \ldots, n$ of $Q(\cdot)$.

```
for (i = 0; i < n; i++) {
   T_i = ∅; /* initialize lookup table set T_i */
   for (all possible requirement variations of Y and Z) {
      /* requirement variations as in Sections 3.3.3.2 and 3.3.4 */
      /* tables within a table set may be merged for efficient lookup */
      T_i = T_i ⋃{table for this requirement variation};
   }
}
return T_1, ..., T_n;
```

with $\alpha = -3.7$ for X-FCSR-256 and $\alpha = -4.4$ for X-FCSR-128. We therefore have $COST_{initializations} = 2^{45.3}$ and $COST_{initializations} = 2^{67.1}$ for X-FCSR-256 and X-FCSR-128, respectively.

A high-level description of the algorithm may be specified as follows. Recall $Q(t)$ from Equation (3.10). The online part of the attack begins by calculating k consecutive such $Q(i), t-k+1 \leq i \leq t$, collectively denoted $Q(\cdot)$ below. For X-FCSR-256 and X-FCSR-128 we have $k = 3$ and 6, respectively. $Q(\cdot)$ is then analyzed byte by byte from left to right. A lookup table set T_1 is queried for plausible state configurations corresponding to the first byte position of $Q(\cdot)$. If solutions exist, we go on and query table set T_2 for matching state configurations corresponding to the second byte position of $Q(\cdot)$, and so on. Two neighboring state configurations are said to be matching if they have identical variable assignments for their common variables.

The algorithm is easily described in terms of depth-first search, if one views the plausible state configurations as vertices in a tree in which two vertices are adjacent if and only if they represent matching solutions at neighboring byte positions. $Q(\cdot)$ corresponds to a forest, the solution space, in which each solution to the first byte position system generates a separate tree. A path of length $n-1$ in this tree represents a permissible configuration for the entire state.

Algorithm 2 – X-FCSR State Recovery at Time t

Input: Keystream bits for computing $Q(\cdot)$, see Equation (3.10).
Output: Recovered state or phrase "No solution".

```
/* offline */
Use Algorithm 1 to compute lookup tables T_1, ..., T_n;
/* online */
Compute Q(·) according to Equation (3.10);
while (performing depth-first search into solution space of Q(·)) {
   if (vertex at depth n is reached) {
      /* found matching state variable assignments
       * for all byte positions */
      return recovered state;
   }
}
return "No solution";
```

3.4 A GENERALIZED BIRTHDAY APPROACH FOR EFFICIENTLY FINDING LINEAR RELATIONS IN ℓ-SEQUENCES

We now shift our focus to the hardware oriented design F-FCSR-H v3. In this section we show how to exploit a particular set of linear relations in ring FCSR sequences. We show what biases can be expected, and we also present a generalized birthday algorithm for actually realizing these relations. As all prerequisites of a distinguishing attack are present, we explicitly show a new such attack on F-FCSR-H v3 with an online time complexity of only $2^{37.2}$. The offline time complexity (for finding a linear relation) is $2^{56.2}$. This is the first successful attack on F-FCSR-H v3, the first attack to breach the exhaustive search complexity limit of 2^{80} (initializations). As mentioned before, this attack is *completely* different from that of F-FCSR-H v2, and despite our focus on F-FCSR-H v3, the presented algorithm is very general and can be applied to any FCSR automaton, making this technique potentially very useful for the future cryptanalysis of linearly filtered FCSRs and FCSR combiners in particular.

In Section 3.2.4 we covered the basics of FCSR analysis, ℓ-sequences and their exponential representation. The remainder of this section is organized as follows. In Section 3.4.1 we recall some known linearity properties of ℓ-sequences, which we expand upon in Section 3.4.2. In Section 3.4.3 we show our generalized birthday algorithm, and Section 3.4.4 shows some simulation results that verify our theory. We discuss applications in Section 3.4.5, where we detail the attack on F-FCSR-H v3.

3.4.1 LINEAR PROPERTIES OF ℓ-SEQUENCES

We now define a particular set of linear relations.

Definition 3.7 ($(i+j)$-relations $\mathcal{R}_q$) Let a connection integer q and distinct positive integers $u_1, \ldots, u_i$ and $v_1, \ldots, v_j$ be given. The linear relation

$$2^{-u_1} + \cdots + 2^{-u_i} \equiv 2^{-v_1} + \cdots + 2^{-v_j} \bmod q$$

is called an $(i+j)$-*relation* and is denoted $\mathcal{R}_q(u_1, \ldots, u_i; v_1, \ldots, v_j)$.

We now take some definitions and results on ℓ-sequences from Tian and Qi [TQ09] that we either state directly or expand upon. Define the r-tuple set

$$\Omega_r(q) = \{(i_1, \ldots, i_r) \,|\, i_1, \ldots, i_r \in \mathbb{Z}/q\mathbb{Z}\backslash\{0\}, i_1 + \cdots + i_r \not\equiv 0 \bmod q\}.$$

The available keystream is always finite in practice, so from now on we consider finite ℓ-sequences only. A linear property of ℓ-sequences can now be described as follows.

Theorem 3.3 ($(3+1)$-relation bias) Let $\underline{a} = \{a_t\}_{t=s+d}^{s+e+k-1}$ be an ℓ-sequence with connection integer q, and let a $(3+1)$-relation $\mathcal{R}_q(w, x, y; z)$, where $d = \min(w, x, y, z)$, $e = \max(w, x, y, z)$, $s \geq 0$ and $\alpha_t = (A2^{-t}) \bmod q$ as in Theorem 3.2 be given. If the triplet sequence

$$\left\{\left(\alpha_{t+w}, \alpha_{t+x}, \alpha_{t+y}\right)\right\}_{t=s}^{s+k-1}$$

cannot be distinguished from a triplet sequence drawn uniformly at random from $\Omega_3(q)$, then the k events

$$a_{t+w} \oplus a_{t+x} \oplus a_{t+y} \oplus a_{t+z} = 0, \qquad 0 \leq t < k,$$

can be seen as k independent Bernoulli trials with success probability $\frac{1}{3}$.

In short, Theorem 3.3 shows that $(3+1)$-relations have bias $\frac{1}{3}$, and Theorem 3.3 can easily be generalized to $(m+1)$-relations. Specifically, the bias is zero when m is even. That is, relations of odd weight do not exhibit a bias that we can detect. Furthermore, the bias is nonzero when m is odd, decreasing as m grows. These nonzero biases stem from properties of modular addition.

Although [TQ09] considers Galois and Fibonacci FCSRs, the proof of Theorem 3.3 only assumes ℓ-sequences, so the result immediately carries over to ring FCSRs.

It should also be noted that Theorem 3.3 above is stated as in [TQ09]. The condition on the triplet sequence over $\Omega_3(q)$ is of particular interest. This condition surely does not hold for a known value q since $\alpha_{t+1} = (\alpha_t 2^{-1}) \bmod q$.

However, the situation is not as bad as it appears. Loosely put, it is sufficient if the corresponding $a_t = LSB(\alpha_t)$ behave randomly in a statistical sense. While a sequence $\{\alpha_t\}$ is fully determined after the first term has been observed, the corresponding binary sequence $\{a_t\}$ is much more well-behaved, even for a known q. The requirement stated for Theorem 3.3 is stronger than it needs to be — it is possible to relax the requirement somewhat.

As Theorem 3.3 is stated it appears that one would need to find a suitable triplet sequence as described above in order to realize an attack, but this is not necessary in practice. In [TQ09], simulations were used to verify that the triplet condition is reasonable for practical applications.

The possibility to use $(3+1)$-relations for cryptanalytic attacks was identified in [TQ09], but the briefly outlined algorithm has a complexity that exceeds 2^{80}, which is the exhaustive search complexity for F-FCSR-H v3. Also, the estimated data complexity of the resulting distinguisher is underestimated by a factor of about 32, which is pointed out in Section 3.4.5.

3.4.2 THE BIAS OF $(2+2)$-RELATIONS AND XORED ℓ-SEQUENCES

Armed with the bias for $(3+1)$-relations, we are now encouraged to find corresponding results for $(2+2)$-relations. A motivational factor here is that balanced equations, with two terms on either side in this case, are suitable for birthday attack approaches.

It turns out that transforming Theorem 3.3 from $(3+1)$-relations to $(2+2)$-relations is not all that hard. All we need is two simple lemmas.

Lemma 3.4 If 2 is a primitive root modulo q, then $2^{\frac{\varphi(q)}{2}} \equiv -1 \bmod q$. We thus have $2^{i+\frac{\varphi(q)}{2}} \equiv -2^i \bmod q$.

The following lemma is a direct consequence of Proposition 1 in [GK97].

Lemma 3.5 Using the notation in Theorem 3.2, we have $a_t = a_{t+\frac{\varphi(q)}{2}} \oplus 1$ and $\alpha_t = q - \alpha_{t+\frac{\varphi(q)}{2}}$.

Using Lemma 3.5, we can now relate a triplet sequence to a $(2+2)$-relation.

Theorem 3.6 ($(2+2)$-relations have bias $\frac{1}{3}$) Let $\underline{a} = \{a_t\}_{t=s+d}^{s+e+k-1}$ be an ℓ-sequence with connection integer q, and let a $(2+2)$-relation $\mathcal{R}_q(w,x;y,z)$ where $d = \min(w,x,y,z)$, $e = \max(w,x,y,z)$, $s \geq 0$ and $\alpha_t = (A2^{-t}) \bmod q$ as in Theorem 3.2 be given. If the triplet sequence

$$\left\{ \left(\alpha_{t+w}, \alpha_{t+x}, q - \alpha_{t+y} \right) \right\}_{t=s}^{s+k-1}$$

cannot be distinguished from a triplet sequence drawn uniformly at random from $\Omega_3(q)$, then the k events

$$a_{t+w} \oplus a_{t+x} \oplus a_{t+y} \oplus a_{t+z} = 0, \qquad 0 \leq t < k, \tag{3.12}$$

can be seen as k independent Bernoulli trials, each with a success probability of $\frac{2}{3} = \frac{1}{2}\left(1 + \frac{1}{3}\right)$.

Proof. Given $2^w + 2^x \equiv 2^y + 2^z \bmod q$, we have $2^w + 2^x - 2^y \equiv 2^z \bmod q$. Using Lemma 3.4, since 2 is a primitive root modulo q, we have $2^w + 2^x + 2^{y+\frac{\varphi(q)}{2}} \equiv 2^z \bmod q$. Thus $\mathcal{R}_q(w, x, y + \frac{\varphi(q)}{2}; z)$ is a $(3+1)$-relation. If we use Lemma 3.5, we can write the triplet sequence given above as

$$\left\{ \left(\alpha_{t+w}, \alpha_{t+x}, \alpha_{t+y+\frac{\varphi(q)}{2}} \right) \right\}_{t=s}^{s+k-1}.$$

From Theorem 3.3 we know that the k events

$$a_{t+w} \oplus a_{t+x} \oplus a_{t+y+\frac{\varphi(q)}{2}} \oplus a_{t+z} = 1, \qquad 0 \le t < k, \tag{3.13}$$

can be seen as k independent Bernoulli trials with success probability $\frac{2}{3}$. Using Lemma 3.5, if we replace $a_{t+y+\frac{\varphi(q)}{2}}$ in Equation (3.13) by $a_{t+y} \oplus 1$, then Equation (3.12) can be obtained. ■

The point of Theorem 3.6 is that the bias of $(2+2)$-relations is $\frac{1}{3}$, which is very large in this context. Theorem 3.6 can also easily be generalized to $(n+n)$-relations.

For the sake of our upcoming F-FCSR-H v3 analysis, we also consider what happens when several ℓ-sequences are combined by bitwise xor. Theorem 3.7 provides some answers.

Theorem 3.7 (Bias of xored ℓ-sequences) Let a $(2+2)$-relation $\mathcal{R}_q(w, x; y, z)$ be given, and let $\underline{a}^i = \{a_t^i\}_{t=s+d}^{s+e+k-1}, 1 \le i \le m$, be m *independent* ℓ-sequences with connection integer q where $d = \min(w, x, y, z)$ and $e = \max(w, x, y, z)$, with α_t^i defined correspondingly for each i. If the m triplet sequences

$$\left\{ \left(\alpha_{t+w}^i, \alpha_{t+x}^i, q - \alpha_{t+y}^i \right) \right\}_{t=s}^{s+k-1}, \quad 1 \le i \le m,$$

cannot be distinguished from uniformly random triplet sequences over $\Omega_3(q)$, then the k events

$$\bigoplus_{i=1}^{m} a_{t+w}^i \oplus a_{t+x}^i \oplus a_{t+y}^i \oplus a_{t+z}^i = 0, \quad s \le t < s + k,$$

can be seen as k independent Bernoulli trials with a total success probability of $\frac{1}{2}\left(1 + \left(\frac{1}{3}\right)^m\right)$.

Proof. The independence of the m ℓ-sequences motivates using the piling-up lemma, see [Mat94]. ■

Theorem 3.7 considers the expected value of the bias, but for practical applications we need to know what level of accuracy we may expect. Theorem 3.8 gives us a practical error-bounding formula for the bias of several xored ℓ-sequences.

Theorem 3.8 (Bounding formula for bias of xored ℓ-sequences) Using the notation from Theorem 3.7, let $\underline{b} = \{b_t\}_{t=s+d}^{s+e+k-1}$ be the bitwise xor of the sequences $\underline{a}^i, 1 \leq i \leq m$. For $\mathcal{R}_q(w,x;y,z)$, the bias of sequence $\underline{b}$ is defined as

$$\varepsilon_{\underline{b}}^q = \frac{1}{k}\sum_{t=0}^{k-1}(-1)^{b_{t+w}\oplus b_{t+x}\oplus b_{t+y}\oplus b_{t+z}}. \tag{3.14}$$

Then, $\varepsilon_{\underline{b}}^q$ satisfies

$$\Pr\left[\varepsilon_{\underline{b}}^q \geq \left(\frac{1}{3}\right)^m - \frac{4.6802}{\sqrt{k}}\right] \approx 0.9999. \tag{3.15}$$

Proof. According to Theorem 3.7, the k events

$$b_{t+w}^i \oplus b_{t+x}^i \oplus b_{t+y}^i \oplus b_{t+z}^i = 0$$

for $0 \leq t < k$ can be seen as independent Bernoulli trials with success probability $p = \frac{1}{2}\left(1 + \left(\frac{1}{3}\right)^m\right)$. If $X_1, X_2, \ldots, X_k$ are random variables associated with these Bernoulli trials, we have

$$\Pr[X_t = 1] = p \text{ and } \Pr[X_t = 0] = 1 - p,$$

with $E(X_t) = \mu = p$ and $Var(X_t) = \sigma^2 = p(1-p)$ for each $0 \leq t < k$. Defining $Y_j = \sum_{i=1}^{j} X_i$, the Central Limit Theorem states that the probability distribution of

$$W_k = \frac{\frac{Y_k}{k} - \mu}{\frac{\sigma}{\sqrt{k}}} = \frac{\frac{Y_k}{k} - p}{\sqrt{\frac{p(1-p)}{k}}}$$

goes to $N(0,1)$ in the limit as $k \to \infty$, see [HT93].

The value k can be regarded as sufficiently large when $k(1-p) \geq 5$ and $kp \geq 5$. According to the normal distribution table, we have

$$\Pr[W_k \geq -4.6802] \approx 0.9999,$$

that is

$$\Pr\left[\frac{Y_k}{k} \geq p - \frac{4.6802 \times \sqrt{p(1-p)}}{\sqrt{k}}\right] \approx 0.9999.$$

When m is large enough, say $m = 10$, $\sqrt{p(1-p)} \approx \frac{1}{2}$, so

$$\Pr\left[\frac{Y_k}{k} \geq p - \frac{2.3401}{\sqrt{k}}\right] \approx 0.9999.$$

Noting that $\varepsilon_{\underline{b}}^q = \frac{2Y_k}{k} - 1$, we finally deduce

$$\Pr\left[\varepsilon_{\underline{b}}^q \geq \left(\frac{1}{3}\right)^m - \frac{4.6802}{\sqrt{k}}\right] \approx 0.9999. \qquad \blacksquare$$

Simulations indicating that Theorem 3.8 holds for practical applications can be found in Section 3.4.4.

3.4.3 A GENERALIZED BIRTHDAY ALGORITHM

We now know the bias of $(2+2)$-relations, but we still need to be able to actually find them. In this section we introduce a generalized birthday algorithm that efficiently solves the problem. Our algorithm will find a $(2+2)$-relation $\mathcal{R}_q(w,x;y,z)$ with width $\max(w,x,y,z) - \min(w,x,y,z) < N$. The value N is tunable, chosen so that the running time of the algorithm is minimized while the success probability is sufficiently high. We need a definition for reduction purposes.

Definition 3.8 (k-small) Given an integer q and a real number k, the integer n-tuple $(i_1, \ldots, i_n)$ is k-small if $2^{-i_1} + \cdots + 2^{-i_n} \bmod q < \frac{q}{k}$.

The k-small 1- and 2-tuples will be referred to as k-small numbers and pairs, respectively. Neither q nor k should be assumed to be small (in the usual sense) in the general case.

We first generate all k-small numbers in a specific interval, storing them in a table T_1. Note that each such number is smaller than $\frac{q}{k}$. We then form all possible (unordered) pairs $(w,x), 0 \leq w,x < N$, of these k-small numbers, storing them in a hash table T_2 (cuckoo hashing with $O(1)$ insertion and lookups is appropriate, see [PR04]) that stores value pairs keyed on their sum modulo q. We keep adding such pairs to T_2 until we find a collision. That is, we look for a set $\{(w,x),(y,z)\}$ satisfying

$$2^{-w} + 2^{-x} \equiv 2^{-y} + 2^{-z} \bmod q.$$

Algorithm 3 specifies the details[7].

[7] From a notational point of view, the reader may think of both T_1 and T_2 as hash tables, where, e.g., $T_1[k] = v$ means insertion of value v keyed on k. While T_1 can be implemented as linear storage (an array), in practice T_2 needs to be implemented as a hash table. This should become clear in Section 3.4.3.1

Algorithm 3 – Generalized Birthday Approach to Finding a $(2+2)$-relation

Input: Integers N and q, real number k.
Output: $\mathcal{R}_q(w,x;y,z)$ or the phrase "No $(2+2)$-relation was found".

create empty tables T_1 and T_2;
$B =$ an integer in the interval $[0,q)$ chosen uniformly at random;
/* insert all k-small numbers in $[B, B+N)$ into T_1 */
for $(i = B; i < B+N; i{+}{+})$ {
  if (i is k-small) {
    $T_1[i] = 2^{-i} \bmod q$;
  }
}
/* insert pairs of k-small numbers into T_2 and look for a collision */
for (all pairs (a,b) of keys a,b from T_1) {
  $s = T_1[a] + T_1[b]$;
  if (key s is in T_2) {
    $(x,y) = T_2[s]$; /* get pair (x,y) from T_2 */
    $m = \min(a,b,x,y)$; /* normalize relation */
    return $\mathcal{R}_q(a-m, b-m; x-m, y-m)$;
  }
  $T_2[s] = (a,b)$; /* insert pair (a,b) into T_2 */
}
return "No $(2+2)$-relation was found";

Because of the reduction, tables T_1 and T_2 will contain approximately $\frac{N}{k}$ and $\binom{\frac{N}{k}}{2}$ entries, respectively. The time complexity of Algorithm 3 is $N + \binom{\frac{N}{k}}{2}$, since this is the time it takes to build tables T_1 and T_2. We expect to find a collision after about

$$\binom{\frac{N}{k}}{2} \approx \sqrt{\frac{2q\alpha}{k}}$$

insertions into T_2, where we use $\alpha = 9.22$ to set the collision probability to at least 99% (see [WikiBP]). Minimizing the time complexity we get the conditions

$$\begin{cases} N = \binom{\frac{N}{k}}{2}, \\ \binom{\frac{N}{k}}{2} = \sqrt{\frac{2q\alpha}{k}}, \end{cases} \iff \begin{cases} N = 2k^2, \\ N^4 = 8k^3\alpha q, \end{cases}$$

which for F-FCSR-H v3 (using $\log_2 q < 160.26$) gives us $k = \left(\frac{\alpha q}{2}\right)^{1/5}$ and $N = 2^{66.0}$ for a total time complexity of $2N = 2^{67.0}$.

A few observations may and should be made at this point. First of all, Algorithm 3 is trivially generalized to $(n+n)$-relations, $n \geq 2$, and the corresponding minimization conditions are given by

$$\begin{cases} N = \binom{\frac{N}{k}}{n}, \\ \binom{\frac{N}{k}}{n} = \sqrt{\frac{nq\alpha}{k}}, \end{cases} \iff \begin{cases} N^{n-1} = n!k^n, \\ N^{2n} = n(n!)^2k^{2n-1}\alpha q. \end{cases}$$

For $n = 3$ for F-FCSR-H v3 we get $k = \left(\frac{\alpha q}{2}\right)^{1/4}$ and $N = 2^{62.3}$ for a total time complexity of $2N = 2^{63.3}$. The bias in this case is $\frac{2}{15}$.

Also, for $n \geq 4$, we can reduce the time complexity further by applying additional smallness reductions in the lower layers according to the generalized birthday approach [Wag02]. In general, the time complexity decreases as n grows.

Secondly, we have tuned the parameters to make it sufficiently probable for us to find *one* linear relation. When we have reached the point of finding the first relation, it quickly becomes quite cheap to find many more collisions by increasing N slightly. This turns the algorithm into a cornucopia of $(n+n)$-relations. One possible usage for this is for a fast correlation attack. This would work very well in distinguishing situations where it is cheap to find many different relations compared to the amount of keystream needed for the distinguisher. One particular observation is important in this case. If we have

$$2^{-u_1} + \cdots + 2^{-u_n} \equiv 2^{-v_1} + \cdots + 2^{-v_n} \bmod q$$

for some numbers $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$, then we also have

$$2^{-u_1+i} + \cdots + 2^{-u_n+i} \equiv 2^{-v_1+i} + \cdots + 2^{-v_n+i} \bmod q$$

for all i. This shows that we need to take dependency into account when we search for multiple linear relations. However, since we employ k-smallness, this effect will only be valid for a limited number of i in our case. Furthermore, the above observation could also be used for normalizing the parameters in an $(n+n)$-relation, as we do in Algorithm 3.

This normalization is performed at the end, just before the $(2+2)$-relation is output. Performing it any sooner would ruin the generalized birthday approach, as we then would be required to find a collision between *different* sets of pairs, ultimately degrading the complexity. Some further comments on dependency can be found in Section 3.4.3.2.

Third, the offline complexity measure used here involves mostly table insertions and lookups. A better comparison to exhaustive key search would compare these complexities to those of initializations. For F-FCSR-H v3, one initialization involves 48 FCSR updates. This should be compared to the operations in Algorithm 3, one modular exponentiation (that can be translated into a shift and a conditional subtraction) and insertion into tables T_1 and T_2. Since these operations do not need to be too complicated, we conclude that our offline complexity measure constitutes a conservative measure compared to initializations.

The online complexity measure, xoring $2n$ bits, is *much* cheaper than 48 FCSR updates (for n of reasonable size).

Fourth, the purpose of the randomly selected value B is to prevent algorithm designers from choosing some suitable q that will extend the running time of Algorithm 3. By choosing the starting point B at random, the expected time complexity does not depend on q.

And last but not least, Algorithm 3 is specified with a fixed N for clarity. In practice one can simply keep extending T_1 and T_2 until sufficiently many collisions have been found.

3.4.3.1 IMPROVEMENT BY MODULAR INTERVAL SUMMATION

We now describe an improvement that makes it even cheaper to find $(2+2)$-relations, a technique that is also generalizable to $(n+n)$-relations. The general idea here is to improve Algorithm 3 by employing a more efficient utilization of the numbers that we generate for storage in table T_1. In the original setting described above, all numbers that are not k-small are simply discarded, and this is a waste of resources.

In the second part of Algorithm 3 we form all possible pairs of k-small numbers from table T_1 and store all such pairs in table T_2. Note that these pairs are $\frac{k}{2}$-small. The improvement is made possible by the fact that we do not necessarily need to sum two k-small numbers to make $\frac{k}{2}$-small pairs. The previously discarded numbers can be used to this end in the following way.

Instead of using one table T_1 we now use k tables $T_{1,0}, \ldots, T_{1,k-1}$ for the first step to store *all* numbers $2^{-i} \bmod q, B \leq i < B + N$. Divide the interval $[0, q)$ into k equally wide subintervals $\left[\frac{jq}{k}, \frac{(j+1)q}{k}\right), 0 \leq j < k$. Then, a number $2^{-i} \bmod q$ is put in table $T_{1,j}$ if it is in the j^{th} subinterval.

As before, we use all entries in table $T_{1,0}$ to form all possible $\frac{k}{2}$-small pairs and store these in table T_2. But we can also form $\frac{k}{2}$-small pairs by pairing any two number from tables $T_{1,1}$ and $T_{1,k-1}$, and any two numbers from tables $T_{1,2}$ and $T_{1,k-2}$, and so on, as shown in Figure 3.18. In this way we can increase the number of pairs we can generate by a factor of about k.

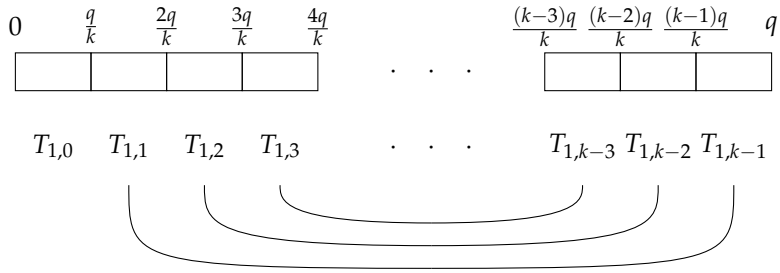


Figure 3.18: Interval subdivision and pairing of tables $T_{1,0}, \ldots, T_{1,k-1}$.

Revising the previous complexity analysis, tables $T_{1,0}, \ldots, T_{1,k-1}$ now contain about $\frac{N}{k}$ elements each for a total of N numbers and a total build time of N insertions.

T_2 will contain about $\binom{\frac{N}{k}}{2} + \frac{k-1}{2}\left(\frac{N}{k}\right)^2 \approx \frac{N^2}{2k}$ entries. The time complexity of the revised Algorithm 3, the time taken to build tables T_1 and T_2, is $N + \frac{N^2}{2k}$. We now expect to find a collision after about

$$\frac{N^2}{2k} \approx \sqrt{\frac{2q\alpha}{k}} \tag{3.16}$$

insertions into T_2, where we continue to use $\alpha = 9.22$ as before. Minimizing the time complexity, we get the revised conditions

$$\begin{cases} N = \frac{N^2}{2k}, \\ \frac{N^2}{2k} = \sqrt{\frac{2q\alpha}{k}}, \end{cases} \quad \Longleftrightarrow \quad \begin{cases} N = 2k, \\ k = \left(\frac{q\alpha}{2}\right)^{\frac{1}{3}}, \end{cases}$$

which gives us $k = 2^{54.2}$ and $N = 2^{55.2}$ for a total time complexity of $2N = 2^{56.2}$ for finding a $(2+2)$-relation for the FCSR used in F-FCSR-H v3.

Table 3.7: Time complexities for finding an $(n+n)$-relation for F-FCSR-H v3.

relation type	k	N	time
$(2+2)$	$2^{54.2}$	$2^{55.2}$	$2^{56.2}$
$(3+3)$	$2^{81.3}$	$2^{42.0}$	$2^{43.0}$
$(4+4)$	$2^{97.5}$	$2^{34.1}$	$2^{35.1}$

Table 3.8: F-FCSR-H v3 distinguisher complexities.

relation type	offline	online
$(2+2)$	$2^{56.2}$	$2^{37.2}$
$(3+3)$	$2^{43.0}$	$2^{63.6}$
$(4+4)$	$2^{35.1}$	$2^{89.7}$

It is possible to double the number of pairs by using, say, $T_{1,0}$ and $T_{1,1}$, and $T_{1,0}$ and $T_{1,k-1}$ together, and so on. Half of the pairs will be $\frac{k}{2}$-small in these cases. However, there is a small additional cost involved and we did not find that these additional pairs affect the time complexity in a positive way.

Generalizing this to $(n+n)$-relations, we need to use the entries in tables $T_{1,0},\ldots,T_{1,k-1}$ to form $\frac{k}{n}$-small n-tuples. This can be done by first choosing entries from any $n-1$ tables $T_{1,0},\ldots,T_{1,k-1}$, and then choosing the last table so that the sum ends up in $\left[0,\frac{nq}{k}\right)$ (modular interval summation). The new conditions become

$$\begin{cases} N = \frac{k^{n-1}}{n!}\left(\frac{N}{k}\right)^n, \\ \frac{k^{n-1}}{n!}\left(\frac{N}{k}\right)^n = \sqrt{\frac{nq\alpha}{k}}, \end{cases} \iff \begin{cases} N = (n!k)^{\frac{1}{n-1}}, \\ k = \left(\frac{(nq\alpha)^{n-1}}{(n!)^2}\right)^{\frac{1}{n+1}}. \end{cases}$$

For F-FCSR-H v3, the corresponding complexities are given in Table 3.7.

In Table 3.8 we also summarize the off- and online complexities for the F-FCSR-H v3 distinguishers for various relation types developed this far.

3.4.3.2 DEPENDENCY BETWEEN LINEAR RELATIONS

Consider once more the linear dependency condition of linear relations that was noted in Section 3.4.3.

Definition 3.9 (Linearly dependent relations) The two linear relations $\mathcal{R}_q(w,x;y,z)$ and $\mathcal{R}_q(w',x';y',z')$ are linearly dependent if and only if

$$w - w' = x - x' = y - y' = z - z'.$$

In Algorithm 3, table T_2 is used to store k-small pairs (w,x), and pairs are continuously added to this table until a collision is found. When two pairs (w,x) and (y,z) finally do collide, when their sums are equal, we have obtained a linear relation $\mathcal{R}_q(w,x;y,z)$. For a relation $\mathcal{R}_q(w,x;y,z)$ we have $0 \leq w,x,y,z < N$, so any pair of linear relations $\mathcal{R}_q(w,x;y,z)$ and $\mathcal{R}_q(w',x';y',z')$ are linearly dependent with a probability of about $\frac{1}{N^3}$. Thus, if we use Algorithm 3 to produce a set of L linear relations, by expected value we should find that about $\frac{L^2}{2N^3}$ of these linear relations are redundant. For the values of N that we consider in our applications, it is hard to see how this dependency could come into play, even for a fast correlation attack where one would need to produce many relations.

However, in order to ensure that this dependency does not deplete the search space of Algorithm 3, we also consider the search procedure itself. Table T_2 contains about $\frac{N^2}{2k}$ pairs, so the number of quadruples (pairs of pairs) that we test is about $\binom{\frac{N^2}{2k}}{2} \approx \frac{N^4}{8k^2}$. The linear dependency condition in Definition 3.9 applies to quadruples as well, in the sense that testing a quadruple that is linearly dependent on a previously tested quadruple is redundant work (search space redundancy). Therefore, among $\frac{N^4}{8k^2}$ quadruples we will find that at most

$$\binom{\frac{N^4}{8k^2}}{2} \frac{1}{N^3} \approx \frac{N^5}{2^7 k^4}$$

of them are linearly dependent, so that the fraction of linearly dependent quadruples is no more than $\frac{N}{16k^2}$. For the values of N and k that we encounter in our applications (see Table 3.7), we can again and finally conclude that the dependency issue is a nonissue.

3.4.4 SIMULATIONS

We have run a set of simulations for verifying the validity of Theorem 3.8. The guidelines in [Arn+09] were followed for random generation of F-FCSRs for these simulations. For each test, a new ring FCSRs of size n and a linear filter with m input bits was used. Algorithm 3 and the modular interval summation technique presented in Section 3.4.3.1 were employed to exhibit a $(2+2)$-relation. We randomly selected 100 keystream subsequences $\underline{b}$ of sufficient length and calculated their bias according to Equation (3.14).

To verify Theorem 3.8, we ran 10000 tests with parameters $20 \leq n \leq 30$ and $2 \leq m \leq 4$. We found Equation (3.15) to hold for all subsequences $\underline{b}$ in

Table 3.9: The set S of nontrivial connections in the transition matrix of a 40-bit F-FCSR.

$(0,6)$	$(1,29)$	$(2,4)$	$(3,15)$	$(5,20)$	$(7,21)$	$(9,37)$
$(10,19)$	$(12,36)$	$(13,34)$	$(15,7)$	$(17,26)$	$(18,17)$	$(20,13)$
$(21,18)$	$(23,39)$	$(25,31)$	$(26,10)$	$(34,11)$	$(35,12)$	$(36,30)$

all tests. The results of these tests can also be seen as an indication that the triplet sequence condition over $\Omega_3(q)$ in Theorems 3.6 and 3.7 is reasonable.

We also ran 100 tests for bigger F-FCSRs, $n = 40$ and $m = 4$, with the same result. Example 3.1 shows one of the F-FCSR ciphers in detail.

Example 3.1 A filtered FCSR cipher (an F-FCSR) based on a 40-bit ring FCSR with connection integer $q = 1111855758899$ and transition matrix $T = (t_{i,j})_{0 \le i,j < 40}$ with

$$t_{i,j} = \begin{cases} 1, & (i,j) \in S \text{ or } j \equiv i+1 \bmod 40, \\ 0, & \text{otherwise}, \end{cases}$$

where S is the set of pairs given in Table 3.9. The linear filter xors the values of the four main register cells m_i with $i \in \{0,5,9,17\}$. The $(2+2)$-relation $\mathcal{R}_q(0,22802;2166,27778)$ with bias $\frac{1}{3}$ was found. Initializing the main memory and carry registers with random bits, we randomly chose 100 subsequences of length 200000 for bias calculation.

We also performed several negative tests using the trials above, but randomly altered one or more of the parameters in the $(2+2)$-relation obtained from Algorithm 3. As expected, in this case, only a subset of the selected keystream subsequences satisfied Equation (3.15).

Beyond this, we also performed simulations to verify our complexity estimation of Algorithm 3 with modular interval summation. More than 10000 experiments with prime parameters $2^{35} \le q \le 2^{40}$ were conducted, and the results suggest that the average value of N is in fact somewhat lower than the estimation.

3.4.5 APPLICATIONS TO RING FCSRS, F-FCSRS, FCSR COMBINERS AND F-FCSR-H V3

Algorithm 3 is applicable to *all* ring FCSRs, so the underlying FCSRs of all linearly filtered FCSRs and FCSR combiners may be targeted. We now focus specifically on distinguishing attacks on F-FCSR-H v3.

From Theorem 3.8 we have

$$\Pr\left[\varepsilon_{\underline{b}}^{q} \geq \left(\frac{1}{3}\right)^{m} - \frac{4.680}{\sqrt{k}}\right] \approx 0.9999.$$

Using the common rule of thumb (see [HJB09]) that approximately $(\varepsilon_{\underline{b}}^{q})^{-2}$ samples are needed to distinguish a given sequence from a uniform distribution, we obtain the condition

$$k \geq \left(\left(\frac{1}{3}\right)^{m} - \frac{4.680}{\sqrt{k}}\right)^{-2},$$

implying that

$$k \geq 32.26 \times 3^{2m},$$

where k denotes the required number of keystream bits for a distinguisher with an accuracy level of 99.99%. In [TQ09] it is claimed that the required length of keystream for the $(3+3)$-relation case is simply 3^{2m}, which is not entirely accurate since that disregards the factor of 32.26. However, the size of this factor depends on the accuracy level of the distinguisher.

Getting back to F-FCSR-H v3, making use of the six smaller subfilters only (out of eight), we can apply the above formula with $m = 10$ to get

$$k \geq \frac{8}{6} \times 32.26 \times 3^{2m} = 2^{37.2}.$$

As stated before, the offline time complexity is $2^{56.2}$. This is the time required for finding the $(2+2)$-relation. As for keystream requirements, we need four separate chunks of $2^{37.2}$ keystream bits each for a total of $2^{39.2}$ keystream bits to perform the attack in time $2^{37.2}$. However, these four chunks can be quite far apart due to the width

$$\max(w, x, y, z) - \min(w, x, y, z)$$

of the $(2+2)$-relation $\mathcal{R}_q(w, x; y, z)$. Thus, we can perform the attack in time $2^{37.2}$, but we still need to observe $2^{56.2}$ keystream bits. The keystream requirement is therefore $2^{56.2}$ bits, while we can manage with only $2^{39.2}$ bits of storage.

The relation width can be made smaller by using higher-order $(n+n)$-relations, but the bias quickly diminishes as n gets large. Also, the fact that we are employing a generalized birthday attack contributes to this width. One would expect to find shorter relations using a standard birthday attack, but the width is then improved only at the expense of an increased time complexity.

The F-FCSR-H v3 distinguisher is made explicit in Algorithm 4, where the last if-clause may use any efficient statistical decision mechanism, for example the Neyman-Pearson lemma in [CT91].

Algorithm 4 – F-FCSR-H v3 Distinguisher

Input: Keystream bits z_t for $t \geq 0$.
Output: The classification CIPHER or RANDOM.

```
/* offline */
Use Algorithm 3 to find a (2+2)-relation R_q(w, x; y, z);
/* online */
c = 0; /* relation counter */
for (i = 0; i < 2^37.2; i++) {
   if (i mod 8 ≥ 2) {/* if small subfilter was used */
      if (z_{i+w} ⊕ z_{i+x} ⊕ z_{i+y} ⊕ z_{i+z} = 0) {
         c++;
      }
   }
}
if (c significantly deviates from 1/2 × 6/8 × 2^37.2) {
   return CIPHER;
}
return RANDOM;
```

3.5 CONCLUDING REMARKS

From what we have seen in Section 3.3, it is clear that the design of the X-FCSR stream cipher family is not sufficiently secure. Depending on one's inclination, it is possible to attribute this insufficiency to the modest clocking of the two FCSRs, the size or number of FCSRs, how they are combined, the complexity of the round function or some other issue. All of these factors are parts of the whole, but the key insight, however, is that it is important not to rely on the nonlinear property of FCSRs too heavily. The LFSRization process shows that it is relatively cheap to linearize Galois FCSRs, the cost being roughly logarithmic in the size of active carry registers.

The attack presented here is not directly applicable to the newer ring FCSRs presented in [Arn+09]. The desired LFSRization effect is much less likely to appear in ring FCSRs since these allow multiple simultaneous feedbacks. After the publication of [SHJ09], new ring FCSR versions of the F-FCSR family and X-FCSR-128 were presented in [Arn+09] and [BMP09], respectively.

In Section 3.4 we presented a generalized birthday algorithm that can be used to find a specific set of linear relations in ring FCSRs. The algorithm was applied to produce different distinguishing attacks on F-FCSR-H v3. The one we advocate uses a $(2+2)$-relation and has off- and online time complexities $2^{56.2}$ and $2^{37.2}$, respectively. These are the first successful attacks on F-FCSR-H

v3, breaking the exhaustive search complexity bound. While this application was very specific, the presented algorithm itself is very general as it is applicable to all ring FCSRs. In particular, it can be applied to the underlying FCSRs of all linearly filtered FCSRs and FCSR combiners.

Future research in this direction may, for example, involve correlation attacks for key recovery.

4

Optimal Sampling and the Stream Cipher HC-128

This chapter is derived from three articles. The first one, »An Optimal Sampling Technique for Distinguishing Random S-boxes« [SH12], was coauthored with Martin Hell and presented at ISIT in 2012. The second paper, »Improved Distinguishers for HC-128« [SRHJ12], was coauthored with Sushmita Ruj, Martin Hell and Thomas Johansson, and was published in the journal Designs, Codes and Cryptography in 2012 (online in 2011). And finally, the paper »Analysis of Xorrotation with Application to an HC-128 Variant« [SHJ12b] was coauthored with Martin Hell and Thomas Johansson, and was presented at ACISP in 2012.

4.1 INTRODUCTION

HC-128 [Wu08] is a stream cipher selected for the eSTREAM [ECRb] final portfolio [ECRa], and is thus considered to be one of the most promising stream ciphers today. Being in the Profile 1 category it is suitable for fast encryption in software. In fact, in most reported results, HC-128 is the fastest stream cipher in the eSTREAM final portfolio. As one example, on a Pentium M processor, the speed of HC-128 reaches 3.05 cycles/byte [Wu08]. This alone makes HC-128 a very interesting target for cryptanalysis.

HC-128 was proposed in 2006 and there have been very few cryptanalytic results on the cipher. There are still no attacks (not relying on side-channel information) that are more efficient than exhaustive key search. Here, it is worth to note that Wu presented a distinguisher for HC-128 in the design paper. We will show how this distinguisher works in detail in Section 4.2.3,

but it uses the least significant bit of several consecutive 32-bit keystream words.

Maitra et al. [Mai+10] showed that Wu's distinguisher can be generalized to work for any one bit of the complete 32-bit word. However, the bias of the other bits is smaller than that of the least significant bit. Thus, they present several new distinguishers, all of which are weaker than Wu's original distinguisher.

Dunkelman [Dun] observed that keystream bits would leak information of the secret state. However, this observation has not yet been exploited in designing a distinguisher, and it was argued by Wu [Wu] that it cannot be used for cryptanalysis at all.

The initialization step was analyzed by Liu and Qui in [LQ09]. They showed that the key can be recovered if the internal state of the cipher is known.

As we will see in Section 4.2.1, two tables play a central role in HC-128. In [PMR10], Paul et al. showed that it was possible to construct one of these tables with knowledge of the other table together with 2048 keystream words. The time complexity for this reconstruction is 2^{42}. While not saying much about the security of HC-128, this result gives more insight into the algorithm itself, possibly providing a foundation for future attacks.

Although HC-128 is targeted at software environments, differential fault analysis of HC-128 was performed in [KY10]. It was shown that by injecting faults into the state, without control over the location or value of the fault, it is possible to recover the internal state.

The most efficient nongeneric attack to date is the distinguishing attack given in [SRHJ12]. We will build up to describing this distinguisher in Section 4.4 with a much improved analytical motivation compared to the original paper. Our distinguisher is based on the one Wu presented in the design paper. Wu used the least significant bit of the 32-bit keystream words to construct his distinguisher. According to our revised analysis, it has a keystream complexity of $2^{169.5}$ 32-bit words. This value is much higher than what Wu claims in [Wu08], but in Section 4.3 we prove that our revised figure is the efficiency level that can be analytically motivated. For comparison, using the same measure, our distinguisher in Section 4.4 will be shown to have both a computational and keystream complexity of $2^{152.6}$ 32-bit words, and subsequent results should be compared to these figures. The speed-up in our distinguisher actually comes from a more efficient sampling technique, and the discrepancy in the complexity figures is caused by sample dependencies.

Various methods of constructing samples from the outputs have been used in the literature. However, it has been unclear exactly how these methods differ and which method is optimal. In Section 4.3 we analyze four different sampling techniques. We prove that two of these sampling techniques—of

which Wu used one—are suboptimal as they utilize dependent samples. We further show one sampling technique that is optimal in terms of error probabilities in the resulting distinguisher. However, this sampling technique is quite impractical as it requires very large storage. We further show a fourth sampling technique that is much more practical, and we prove that it is equivalent to the optimal one. This is the sampling technique used in Section 4.4. We also show an improved algorithm for calculating the associated probability distributions that are required for using the optimal sampling technique. Our new algorithm saves about 80-85% in time and uses memory optimally in the sense that memory usage is in the order of the amount of memory required for storing the resulting probability distribution.

The analysis in Section 4.3 is inspired by HC-128, but the results are truly general. HC-128 is table-driven, and the tables are initialized and periodically reinitialized with (pseudo-)random entries. The tables may be regarded as random S-boxes that are rerandomized occasionally, and these can appear in cryptanalytical situations when observations, e.g., linear sums of keystream bits in stream ciphers, can be derived from outputs of a large table. A random S-box is an a-to-1-bit mapping in our analysis, and this mapping can be seen as a table containing $n = 2^a$ single bit entries, where the values of the bits are determined independently and uniformly at random by flipping a fair coin n times.

The optimal sampling technique can be applied to any cryptographic primitive that leaks a sufficiently biased probability distribution of the described sort.

One may note that we have restricted our S-box definition to an a-to-b-bit mapping with $b = 1$ instead of the more general case $b \geq 1$. Our results can be applied to the latter case as well, for example by considering only one of the output bits of the S-box. However, some additional work is required to obtain an optimal sampling technique for the general S-box setting.

It may also be noted that S-boxes quite often are referred to as *Boolean functions* in the literature. We will not be exploring this connection here, but we will get back to defining Boolean functions and some of their simple properties in Section 5.2.1 for the purpose of algebraic interpretation in Chapter 5.

In Section 4.5, we take a small step toward a different kind of analytical analysis. Consider the Xorrotation[1] function family

$$g_{w,r_1,\ldots,r_m}(x) = x \oplus (x \lll r_1) \oplus \cdots \oplus (x \lll r_m)$$

of functions for which x is a w-bit word, $\oplus$ denotes xor and $\lll$ denotes left rotation (cyclic shift) with respect to the word length w. For the rotation

[1] Named analogously to their Xorshift cousin defined and analyzed in [Mar03, PL05].

amounts r_i we have $0 < r_i < w$. These bit mixing functions are often used in cryptographic primitives to provide intraword diffusion.

While primitives that rely on modular addition (A), rotation (R) and xor (X) are commonly labeled ARX, $g_{w,r_1,\ldots,r_m}$ is RX. Pure AX and RX systems have been shown to be weak (see [KN10, PP05]), but we will show how our theory can be used in practice by applying it to a more complex system that includes RX operations *and* S-boxes. This system is composed of a *partly* linearized variant of HC-128 for which modular additions have been replaced by xor. We call this variant HC-128$^{\oplus}$ and define it in Section 4.2.2.

The Xorrotation function family has been studied by Thomsen [TK09] and Rivest [Riv11]. Thomsen showed that the mapping is invertible for all choices of distinct r_i with $0 \leq r_i < w$, and all word lengths $w = 2^k$ where $k \geq 2$ is an integer and m is even. Rivest gave a different and more general proof, a proof that in some sense reveals the true nature of the invertibility of the mapping. Many questions remain open, however. For example, some insight into the cases $w \neq 2^k$ and even m, separately and together, would be desirable.

While the main focus of Thomsen and Rivest was on invertibility, we are more interested in the probability distributions that $g_{w,r_1,\ldots,r_m}$ induce. That is, given an x chosen uniformly at random, what can we say about $g_{w,r_1,\ldots,r_m}(x)$ for different values of w and r_i, and how much do the resulting probability distributions differ from the uniform one? To answer this question we will need some more information about the function than an assessment of its invertibility.

From a cryptanalytic perspective, again, a primitive that exposes a heavily biased probability distribution is prone to distinguishing attacks. We present a general treatment of bit mixing using xor and word rotations. A number of theoretical results on related probability distributions are deduced, and these results can be used for cryptanalysis. After introducing probabilistic probability distributions in Section 4.6, we show how to apply the theory, together with some additional new observations, to produce a new distinguisher for HC-128$^{\oplus}$. This is described in Section 4.7. While the keystream complexity of this distinguisher is far from practical ($2^{90.9}$), we still feel that this part presents a significant contribution. These are the first results for HC-128$^{\oplus}$, and our hope is that the new ideas presented here can lead to even better analyses and attacks, perhaps reducing the keystream or total complexity of the untweaked HC-128 to a complexity below 2^{128}.

Background information on HC-128, HC-128$^{\oplus}$ and Wu's original distinguisher can be found in Section 4.2. The chapter is concluded in Section 4.8.

4.2 BACKGROUND

We will now briefly describe HC-128 and the variant HC-128$^{\oplus}$, which we will analyze later on in Section 4.7. As our analysis is independent of the initialization procedure, it will not be detailed here. The reader is referred to the specification for that part. Our focus now is on keystream generation and the original distinguisher that Wu presented in [Wu08].

4.2.1 BRIEF DESCRIPTION OF HC-128

In this section we give a very brief description of the original HC-128 keystream generation process. HC-128 is defined in [Wu08], from which we adopt and adapt the notation. HC-128 specifies both a key and IV size of 128 bits. Up to 2^{64} bits of keystream can be generated with each key/IV pair. Letting x and y be 32-bit integers, we have

$+$: $x + y$ means $(x + y) \mod 2^{32}$,
$\boxminus$: $x \boxminus y$ means $(x - y) \mod 512$,
$\oplus$: xor,
$\|$: concatenation,
$\ll$: left shift, $x \ll n$ means x shifted left n bits (zero-padded),
$\gg$: right shift, $x \gg n$ means x shifted right n bits (zero-padded),
$\lll$: left rotation, $x \lll n$ means x rotated left n bits,
$\ggg$: right rotation, $x \ggg n$ means x rotated right n bits.

Two tables, denoted P and Q, make up the internal state of HC-128. Each table contains 512 entries, where each entry is a 32-bit word. The keystream is denoted by s and the 32-bit keystream word generated at the i^{th} step is denoted s_i; $s = s_0\|s_1\|s_2\|\ldots$. The following six functions are used in HC-128:

$$
\begin{aligned}
f_1(x) &= (x \ggg 7) \oplus (x \ggg 18) \oplus (x \gg 3),\\
f_2(x) &= (x \ggg 17) \oplus (x \ggg 19) \oplus (x \gg 10),\\
g_1(x,y,z) &= ((x \ggg 10) \oplus (z \ggg 23)) + (y \ggg 8),\\
g_2(x,y,z) &= ((x \lll 10) \oplus (z \lll 23)) + (y \lll 8),\\
h_1(x) &= Q[x_0] + Q[256 + x_2],\\
h_2(x) &= P[x_0] + P[256 + x_2],
\end{aligned}
$$

where $x = x_3\|x_2\|x_1\|x_0$ is a 32-bit word and the x_i's represent a byte each, x_0 being the least significant byte of the word and x_3 being the most significant byte. The functions f_1 and f_2 are only used in the initialization of the cipher.

Algorithm 5 – HC-128 Keystream Generation

Input: initialized tables P and Q, each containing 512 32-bit words.
Output: 32-bit keystream words s_i for $i = 0, 1, \ldots$

```
i = 0;
repeat (until enough keystream bits are generated) {
    j = i mod 512;
    if ((i mod 1024) < 512) {
        P[j] += g1(P[j ⊟ 3], P[j ⊟ 10], P[j ⊟ 511]);
        s_i = h1(P[j ⊟ 12]) ⊕ P[j];
    } else {
        Q[j] += g2(Q[j ⊟ 3], Q[j ⊟ 10], Q[j ⊟ 511]);
        s_i = h2(Q[j ⊟ 12]) ⊕ Q[j];
    }
    i += 1;
}
```

Keystream generation proceeds as follows. One table entry is updated and one 32-bit keystream word is generated at each step. One full update of an entire table P or Q takes place during a *session* consisting of 512 consecutive steps. First, table P is updated and table Q is used to provide update values. The roles of tables P and Q are reversed every session. The keystream generation algorithm of HC-128 is given in Algorithm 5.

We will find it convenient to express $P[i]$ as P_i, $P[i \boxminus j]$ as P_{i-j}, and we will write P_{i-j}^{k} for $P_{i-j} \ggg k$.

4.2.2 AN HC-128 VARIANT: HC-128$^{\oplus}$

Let HC-128$^{\oplus}$ denote the HC-128 variant obtained by replacing $+$ with $\oplus$ in three principal places; in the g and h functions listed in Section 4.2.1, and in the table cell update function in Algorithm 5. This is the HC-128 variant that we will analyze in Section 4.7.

4.2.3 ORIGINAL DISTINGUISHING ATTACK BY WU

In the HC-128 design paper, a distinguishing attack was given based on the least significant bit of the keystream words s_i. Table P is updated at the i^{th} step, if $(i \bmod 1024) < 512$. The update function is given by

$$P[i \bmod 512] \mathrel{+}= g_1(P[i \boxminus 3], P[i \boxminus 10], P[i \boxminus 511]).$$

Also, $s_i = h_1(P[i \boxminus 12]) \oplus P[i \bmod 512]$. For $10 \leq (i \bmod 1024) < 511$, this feedback function can be alternatively written as

$$\begin{aligned} s_i \oplus h_1(z_i) =& (s_{i-1024} \oplus h'_1(z_{i-1024})) + \\ & g_1(s_{i-3} \oplus h_1(z_{i-3}), s_{i-10} \oplus h_1(z_{i-10}), s_{i-1023} \oplus h'_1(z_{i-1023})). \end{aligned} \tag{4.1}$$

Here, $h_1(x)$ and $h'_1(x)$ indicate two different functions because they refer to different S-boxes; z_j denotes the $P[j \boxminus 12]$ at the j^{th} step.

As shown in [Wu08], for the least significant bit, this equation reduces to

$$\begin{aligned} &[s_i]^0 \oplus [s_{i-3}]^{10} \oplus [s_{i-10}]^8 \oplus [s_{i-1023}]^{23} \oplus [s_{i-1024}]^0 = \\ &[h_1(z_i)]^0 \oplus [h_1(z_{i-3})]^{10} \oplus [h_1(z_{i-10})]^8 \oplus [h'_1(z_{i-1023})]^{23} \oplus [h'_1(z_{i-1024})]^0, \end{aligned}$$

where $[a]^i$ represents the i^{th} least significant bit of a. Looking at two *different* time instances i and j where $1024 \times \gamma + 10 \leq i, j < 1024 \times \gamma + 511$ we can write

$$\begin{aligned} &[s_i]^0 \oplus [s_{i-3}]^{10} \oplus [s_{i-10}]^8 \oplus [s_{i-1023}]^{23} \oplus [s_{i-1024}]^0 = \\ &[s_j]^0 \oplus [s_{j-3}]^{10} \oplus [s_{j-10}]^8 \oplus [s_{j-1023}]^{23} \oplus [s_{j-1024}]^0, \end{aligned} \tag{4.2}$$

which holds if and only if

$$\begin{aligned} &[h_1(z_i)]^0 \oplus [h_1(z_{i-3})]^{10} \oplus [h_1(z_{i-10})]^8 \oplus [h'_1(z_{i-1023})]^{23} \oplus [h'_1(z_{i-1024})]^0 = \\ &[h_1(z_j)]^0 \oplus [h_1(z_{j-3})]^{10} \oplus [h_1(z_{j-10})]^8 \oplus [h'_1(z_{j-1023})]^{23} \oplus [h'_1(z_{j-1024})]^0. \end{aligned} \tag{4.3}$$

We call the expressions in Equation (4.3) t_i and t_j, respectively, so that

$$\begin{aligned} t_i =& [h_1(z_i)]^0 \oplus [h_1(z_{i-3})]^{10} \oplus \\ & [h_1(z_{i-10})]^8 \oplus [h'_1(z_{i-1023})]^{23} \oplus [h'_1(z_{i-1024})]^0. \end{aligned} \tag{4.4}$$

Equation (4.3) can be approximated as

$$R(a_1) = R(a_2), \tag{4.5}$$

where R denotes a random secret 80-to-1-bit S-box, a_1 and a_2 are two 80-bit random inputs,

$$\begin{aligned} a_1 &= \overline{z}_i \| \overline{z}_{i-3} \| \overline{z}_{i-10} \| \overline{z}_{i-1023} \| \overline{z}_{i-1024}, \\ a_2 &= \overline{z}_j \| \overline{z}_{j-3} \| \overline{z}_{j-10} \| \overline{z}_{j-1023} \| \overline{z}_{j-1024}, \end{aligned}$$

where $\overline{z}$ indicates the concatenation of the least significant byte and the second most significant byte of z, i.e., $\overline{z} = x_0 \| x_2$. Theorem 4.1 from [Wu08], shows the bias of Equation (4.5).

Theorem 4.1 (Random S-box bias) Let R be an m-to-n-bit S-box and all those n-bit elements are randomly generated, where $m \geq n$. Let a_1 and a_2 be two m-bit random inputs to R. Then $R(a_1) = R(a_2)$ with probability $2^{-m} + 2^{-n} - 2^{-m-n}$.

Thus, Equation (4.5) holds with probability $\frac{1}{2} + \varepsilon = \frac{1}{2} + 2^{-81}$. Approximating the number of samples needed in a distinguisher by $4\varepsilon^{-2}$, Wu concludes that 2^{164} such equations are needed. Since it is possible to obtain $\binom{501}{2} \approx 2^{17}$ pairs for each 512-word keystream chunk, the number of keystream words needed are concluded to be 2^{156}, a factor of $\frac{\binom{501}{2}}{501} \approx 2^8$ less. This is however not correct. Wu runs into what seems to be two problems here—both spelled dependency.

The first problem is that the all pairs sampling (APS) technique that Wu uses, forming the 2^{17} samples from a 512-word keystream chunk, produces dependent samples. The hypothesis test used requires independent samples, so this approach is clearly a mismatch.

The second problem is that the dependency issue is actually even worse for Wu. As we will see in Theorem 4.2 in Section 4.3.2, it is not even possible to take more than one single sample from a 512-word keystream chunk. This induces a penalty factor of 2^9, so it seems more fair to conclude that Wu's distinguisher has a keystream complexity of 2^{173} words (32-bit). At least, this is the keystream complexity we obtain if we impose the condition of analytical correctness on Wu's proposed distinguisher. So, while Wu claims that it is possible to lower the keystream complexity of his original distinguisher from about 2^{164} to 2^{156} keystream words using APS, he should rather have concluded that the total keystream complexity of the distinguisher is about 2^{173}.

In a nutshell, two dependent samples carry less information than two independent ones. The consequence of using dependent samples is that the efficiency of the distinguisher is downgraded.

A number of interesting questions can be identified at this point. How efficient is Wu's APS technique? That is, if we erroneously but deliberately feed the standard hypothesis test with dependent samples and compensate for the relative inefficiency of the samples by taking more of them, how many samples does his distinguisher require? Wu assumes perfect efficiency, providing neither analytical justification nor simulation results. Also, using Wu's technique for combining keystream, Theorem 4.2 forces us to use only two out of 512 keystream words to construct one sample. But throwing the rest of them away seems like a waste of resources, so how can the keystream words be used more efficiently? What other reasonable sampling techniques are there? Is there an optimal sampling technique, and how good is it? As we will see, these questions can and will be answered in Section 4.3.

We will use the relative entropy measure for comparing probability distributions and building distinguishers, so we now give the corresponding complexity figure using this metric also for Wu's distinguisher. This allows us to make a fair comparison between our results and Wu's attack. With $p = \frac{1}{2} + 2^{-81}$ as before, we conclude that Wu's distinguisher requires $2^{160.5}$ samples. According to the arguments given above, including the penalty factor of 2^9, this corresponds to a keystream complexity of $2^{169.5}$ 32-bit words. For comparison, our best HC-128 distinguisher (in Section 4.4) will be shown to have both a computational and keystream complexity of $2^{152.6}$.

To see where we stand on the sampling issue, we now take a dive into analysis of various sampling techniques.

4.3 AN OPTIMAL SAMPLING TECHNIQUE FOR DISTINGUISHING RANDOM S-BOXES

At first glance it may appear that we have geared this section toward HC-128, but the results presented here are actually much more general than that. Our sampling technique analysis is potentially applicable to any cryptographic primitive that is table-driven or whose keystream is derived from S-box outputs. Let us introduce the general setting.

You are given an S-box with n random-valued entries, and these entries are unknown to you. At every time instance, one S-box entry is selected uniformly at random and the value contained in this entry is displayed. That is, you get to see the value, not its actual location within the S-box. This construction can be viewed as a very simple toy stream cipher—stream cipher T—that outputs one of its S-box entries at every clocking.

It is now our job to distinguish the given sequence of outputs from a truly random sequence of symbols, i.e., we are to build an efficient distinguisher. Also, to make the task a little more challenging, the S-box in stream cipher T is rerandomized after ℓ clockings, putting a time pressure on us. A set of ℓ such S-box observations will be referred to as a *chunk*, and the observations themselves are denoted $s_1, \ldots, s_\ell$. An illustration of the stream cipher T is provided in Figure 4.1.

One may note that the given problem has a simple solution if the S-box entries are very large in comparison to n. Assume that stream cipher T, like HC-128, has tables with $512 = 2^9$ elements, each of which are 32-bit words. The comparison in size here is between 2^9 and 2^{32}. We could then build a distinguisher based on the fact that repetitions of any 32-bit word are very common for stream cipher T, while they are very scarce in the random case. The easy version of the problem has an easy solution, so we go for the hardest version instead.

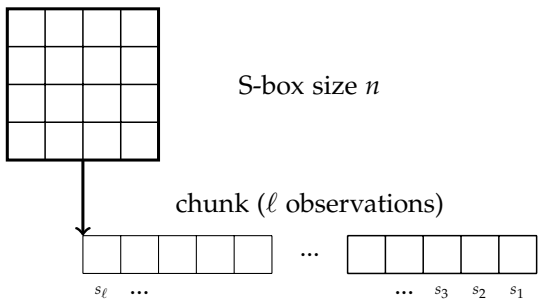


Figure 4.1: The toy stream cipher T.

Now assume that every S-box entry contains one single bit, so that the observations s_i denote single bits, and a chunk is an ordered sequence of ℓ single-bit observations. To summarize, we consider a $\log n$-to-1-bit random S-box that allows ℓ observations before it is rerandomized. The S-box may be regarded as a table with n entries of one bit each.

Also, our distinguisher may need more than one chunk to make a reasonable classification decision, so the number of ℓ-bit chunks it uses is denoted k, for a total of $k\ell$ observations.

Two outputs from an S-box of size n are equal with probability (at least) $\frac{1}{n}$, since the same entry may have been used twice. This simple observation can be used to construct a distinguisher for random S-boxes, also for the harder single-bit case.

As stated in Theorem 4.1, the xor sum of a pair of output bits is biased, and this bias stems from the fact that the same S-box entry may have been probed for both outputs. More specifically, for $i \neq j$,

$$\Pr(s_i = s_j) = \frac{1}{2}\frac{n-1}{n} + \frac{1}{n} = \frac{1}{2}(1 + \frac{1}{n}) = \frac{1}{2} + \frac{1}{2n}, \tag{4.6}$$

and the bias in Equation (4.6) can be used to construct a distinguisher.

The main problem we study now is exactly *how* to construct this distinguisher when the number of S-box observations is more than two. That is, how should a cryptanalyst use the observations to construct an optimal distinguisher?

The empirical probability distribution as defined by the sampling is denoted P^*. The corresponding (theoretical) probability distribution of the S-box is denoted P_1, while its uniform distribution is denoted P_2. The optimal hypothesis test is given by the Neyman-Pearson lemma, see Lemma 2.1. If we want the error probabilities in Neyman-Pearson to be equal, we set $T = 1$. In

other words, we decide $P^* = P_1$ if

$$\frac{P_1(x_1, \ldots, x_t)}{P_2(x_1, \ldots, x_t)} > 1 \underset{\text{indep.}}{\iff} \sum_{i=1}^{t} \log \frac{P_1(x_i)}{P_2(x_i)} > 0, \tag{4.7}$$

and $P^* = P_2$ otherwise. The equivalence in Equation (4.7) is valid when the samples $x_1, \ldots, x_t$ are independent.

In our case, the samples x_i will be constructed from the observations s_j. Note that the Neyman-Pearson lemma, which gives the optimal distinguisher, requires that the samples x_i are independent. By sampling technique we mean how to use the observations to build the samples used in the distinguisher.

If the samples are very easy to construct from the observations, we can say that the online computational complexity of the attack is given by the number of terms t in Equation (4.7). The offline complexity is the time needed to compute P_1.

We will consider the following four sampling techniques:

- **All Pairs Sampling (APS)** Take observation pairs $(s_i, s_j), 1 \leq i < j \leq \ell$ as samples. Let P_1 be the distribution corresponding to Equation (4.6), i.e., $\Pr(s_i = s_j) = \frac{1}{2}(1 + \frac{1}{n})$ and $\Pr(s_i \neq s_j) = \frac{1}{2}(1 - \frac{1}{n})$. P_2 is the uniform distribution, $\Pr(s_i = s_j) = \Pr(s_i \neq s_j) = \frac{1}{2}$.

- **Linear Pairs Sampling (LPS)** Take observation pairs $(s_i, s_{i+1}), 1 \leq i < \ell$ as samples and let P_1 and P_2 be as for APS above.

- **Vector Sampling (VS)** Take vectors $(s_1, s_2, \ldots, s_\ell)$ as samples and perform the hypothesis test with the corresponding vector probability distributions as P_1 and P_2.

- **Weight Sampling (WS)** Take vector weights $\| (s_1, s_2, \ldots, s_\ell) \|_1 = \sum_{i=1}^{\ell} s_i$ as samples and perform the hypothesis test with the corresponding vector weight probability distributions as P_1 and P_2.

It is clear that Vector Sampling (VS) is optimal since it preserves all information in the samples. The drawbacks are that the distribution is very large in storage (2^ℓ), and that it is demanding to compute. As we have previously seen, APS is precisely the sampling technique employed in Wu's original HC-128 distinguisher. It uses the easily computed bias in Equation (4.6) and produces many samples. For ℓ observations, $\binom{\ell}{2}$ samples are produced. Due to the dependency between samples, LPS was suggested in [SRHJ12], and WS was also applied as an improvement. However, it was an open question whether it was possible to improve over WS as it appears that not all sample information is retained in the vector weight samples.

In Sections 4.3.1 and 4.3.2 we prove that APS and LPS are faulty. In Sections 4.3.3 and 4.3.4 we give algorithms for computing the required distributions for VS and WS, respectively. We also prove that VS and WS are equivalent in terms of the performance of the resulting distinguisher. Section 4.3.5 explicitly compares APS, LPS and WS.

4.3.1 ALL PAIRS SAMPLING (APS)

The Neyman-Pearson lemma assumes that all samples are independent and identically distributed. In APS sampling, all possible bit pairs in an ℓ-bit chunk are taken as samples, producing in total $k\binom{\ell}{2}$ samples. It is very easy to prove that these samples are not independent. Consider a chunk with $\ell = 3$, where we take the samples $(s_1, s_2), (s_1, s_3)$ and (s_2, s_3). If we know the first two samples, then we also know the last sample. The last sample does not contain any new information. Stating this information theoretically, we have

$$H(S_2 \oplus S_3 | S_1 \oplus S_2, S_1 \oplus S_3) = 0,$$

where H is the entropy function from Definition 2.1, $\oplus$ denotes xor and S_1, S_2 and S_3 are random variables corresponding to the three observations. This argument is easily extended to the general case with arbitrary ℓ, which also serves as a direct motivation for defining and using LPS sampling.

Even though the samples are dependent, APS is very easy to apply. Computing and storing probability distribution P_1 requires negligible memory and can be trivially done by hand, see Equation (4.6). However, the large number of samples gives an online computational complexity of $k\binom{\ell}{2} = O(k\ell^2)$.

4.3.2 LINEAR PAIRS SAMPLING (LPS)

In LPS sampling we take (s_1, s_2) as the first sample and then only take one new sample for each new observation. This procedure produces $\ell - 1$ samples per chunk for a total of $k(\ell - 1)$ samples. In order to show that this sampling technique also gives dependent samples, for P_1 we calculate and compare $\Pr(s_3 = s_2 | s_2 = s_1)$ and $\Pr(s_3 = s_2 | s_2 \neq s_1)$ to see that the probability of pair equality in one sample depends on the pair equality of the previous one.

We regard the S-box as a table with n entries. The first time we read a specific entry in the table, we say that we "open" the entry. First consider $\Pr(s_3 = s_2 | s_2 \neq s_1)$. Given that $s_2 \neq s_1$, we must have opened precisely two entries in the table, one 0 and one 1. We can now have $s_3 = s_2$ in two different ways, by reading s_3 from either an "old" entry (same as s_2) or a "new" previously unopened one. Thus, we have

$$\Pr(s_3 = s_2 | s_2 \neq s_1) = 1 \cdot \frac{1}{n} + \frac{1}{2} \cdot \frac{n-2}{n} = \frac{1}{2}.$$

Calculating $\Pr(s_3 = s_2 | s_2 = s_1)$ divides into two cases.

Case A: s_1 and s_2 were read from the same entry.

Case B: s_1 and s_2 were read from different entries.

The probability of case A is $p = \frac{1}{n}$, while that of case B is $q = \frac{n-1}{2n}$. Given case A, the probability that $s_3 = s_2$ is

$$a = \frac{1}{n} + \frac{n-1}{2n} = \frac{n+1}{2n}.$$

Given case B, the probability that $s_3 = s_2$ is

$$b = \frac{2}{n} + \frac{n-2}{2n} = \frac{n+2}{2n}.$$

In total we get

$$\Pr(s_3 = s_2 | s_2 = s_1) = \frac{p}{p+q} \cdot a + \frac{q}{p+q} \cdot b = \frac{1}{2}(1 + \frac{2}{n+1}) > \frac{1}{2},$$

from which we conclude that

$$\Pr(s_3 = s_2 | s_2 \neq s_1) \neq \Pr(s_3 = s_2 | s_2 = s_1).$$

This proves that LPS is also erroneous in assuming independence between samples.

One may further note that the same probabilities are valid for any other overlapping pair, i.e., for $\Pr(s_k = s_j | s_j \neq s_i)$ and $\Pr(s_k = s_j | s_j = s_i)$ for all distinct indices i, j and k.

This dependency may seem natural since the two samples are overlapping in one of the observations. Collecting samples in a nonoverlapping fashion according to $(s_1, s_2), (s_3, s_4), (s_5, s_6)$, and so on, may at first glance seem better. However, by performing similar calculations we can also prove that

$$\Pr(s_4 = s_3 | s_2 \neq s_1) \neq \Pr(s_4 = s_3 | s_2 = s_1).$$

The corresponding calculations show that

$$\Pr(s_4 = s_3 | s_2 \neq s_1) = \frac{1}{2}\left(1 + \frac{n-2}{n^2}\right) \text{ and}$$
$$\Pr(s_4 = s_3 | s_2 = s_1) = \frac{1}{2}\left(1 + \frac{n^2+3n-2}{n^2(n+1)}\right).$$

This means that the probability of pair equality in one sample depends on the previous one in this case as well. This immediately generalizes to all nonoverlapping pairs, i.e., the same holds for $\Pr(s_j = s_i | s_v \neq s_u)$ and $\Pr(s_j = s_i | s_v = s_u)$ for all distinct indices i, j, u and v. Since the overlapping and nonoverlapping cases are exhaustive, we can conclude that any two samples will be dependent. An intuitive explanation for this is that a sample leaks information about the entries in the S-box. This information will affect the probability of the next sample since we may read the same entries as before. We summarize this result in Theorem 4.2.

Theorem 4.2 (Random S-box sampling theorem) It is not possible to extract more than one independent sample from a chunk $s_1, \ldots, s_\ell$ of observations from a random S-box.

Both APS and LPS are erroneous in assuming that their samples are independent, but that does not make the sampling techniques completely useless. It *does* mean that the efficiency of APS and LPS is lower than projected, which is an important factor to consider when employing these sampling techniques. Assuming full efficiency by ignoring dependencies is simply wrong. An estimation of the efficiencies of APS and LPS can be found in Section 4.3.5.

Computing and storing probability distribution P_1 for LPS is as simple and efficient as it is for APS. The advantage of LPS over APS is that fewer samples are used. The computational complexity of the online phase of LPS is $k(\ell - 1)$, or $O(k\ell)$, which is a significant improvement over APS if the chunks are very large (large ℓ).

4.3.3 VECTOR SAMPLING (VS)

In order to apply the Neyman-Pearson lemma correctly, we need to find the probability distribution of the complete chunk. Thus, we collect all observations in one vector $(s_1, s_2, \ldots, s_\ell)$. The vector probability distributions P_1 and P_2 both have a domain of size 2^ℓ, which determines the storage cost for handling P_1 and P_2 with VS.

For P_2, all vectors are equally likely, resulting in identical probability values $P_2(v) = 2^{-\ell}$ for all vectors v.

The S-box vector probability distribution P_1 can be calculated according to Algorithm 6, which is stated recursively for simplicity. We hope that this algorithm will clearly illustrate how the probabilities are derived. The main idea here is simply to use variables a_0 and a_1 to keep track of the number of probed table entries containing zeros and ones, respectively. These utility values will enable us to compute the associated probabilities at each stage.

The storage requirement for Algorithm 6 is precisely 2^ℓ (probability entries), and since this amount of memory is necessary to store the resulting

Algorithm 6 – Vector Distribution (vd)

Input: S-box size n, vector length ℓ, current depth d, current probability p, probability distribution container $dist$ of length 2^ℓ, vector v, number of opened table entries with zeros a_0, number of opened table entries with ones a_1.
Output: probability distribution $dist$.
Initial recursion parameters: $dist$ zeroized, $(d, p, v, a_0, a_1) = (0, 1, 0, 0, 0)$.

if (d==ℓ) { /* maximum depth reached */
  $dist[v]$ += p;
  return;
}
if ($a_0 > 0$) { /* old 0 reopened */
  $vd(dist, n, \ell, d+1, p \cdot \frac{a_0}{n}, v\|0, a_0, a_1)$;
}
if ($a_1 > 0$) { /* old 1 reopened */
  $vd(dist, n, \ell, d+1, p \cdot \frac{a_1}{n}, v\|1, a_0, a_1)$;
}
if ($a_0 + a_1 < n$) { /* table not exhausted */
  $vd(dist, n, \ell, d+1, p \cdot \frac{n-(a_0+a_1)}{2n}, v\|0, a_0+1, a_1)$; /* new 0 */
  $vd(dist, n, \ell, d+1, p \cdot \frac{n-(a_0+a_1)}{2n}, v\|1, a_0, a_1+1)$; /* new 1 */
}

probability distribution, no other algorithm can do better in terms of memory. The time complexity of Algorithm 6 is also exponential in ℓ, at most $4^\ell = 2^{2\ell}$, because every call at depth d results in at most 4 calls at depth $d+1$. By employing dynamic programming, see e.g., [CLRS09], it is possible to improve this time complexity to $O(n^2 2^\ell)$ at the expense of increased storage, $O(n^2 2^\ell)$, but the running time must still necessarily be exponential in ℓ.

For large ℓ, i.e., when many observations are generated before the S-box is reinitialized, the vector sampling technique is infeasible since the distribution P_1 is both too large to store and too demanding to compute.

4.3.4 WEIGHT SAMPLING (WS)

Now consider WS, for which we take vector weights $\|(s_1, s_2, \ldots, s_\ell)\|_1 = \sum_{i=1}^{\ell} s_i$ as samples. The corresponding vector weight probability distributions P_1 and P_2 have domains of size $\ell + 1$, which is much more manageable than those for VS.

For WS we begin with P_2. Every vector is equally likely in the ideal case, so the resulting vector weight probability distribution is combinatorially determined by

$$P_2(w) = \binom{\ell}{w} 2^{-\ell}$$

for all vector weights $0 \leq w \leq \ell$.

P_1 can be calculated according to Algorithm 7, which is just a simple modification of Algorithm 6. Instead of passing on a (partial) vector we now pass on the (accumulated) vector weight. The algorithm is, again, stated recursively for simplicity.

The recursiveness is an impediment as ℓ grows, of course, and for our application to HC-128 in Section 4.4 we will aim for $\ell = 501$. Deriving this probability distribution for such vector lengths is no walk in the park. We will need to perform our computations much more efficiently, so we show how to do this using a dynamic programming (DP) approach.

When translating Algorithm 7 into a DP variant, we use temporary storage for all intermediate probability values that are used in order to avoid unnecessary recalculations. The intermediate probability values for vectors of length $\ell - 1$ are used to deduce the corresponding probabilities for vectors of length ℓ. Consider the tuple (w, a_0, a_1), for which w indicates weight and the a's indicate how many table entries of each sort that have been opened, as before. The intermediate storage contains one probability entry for each such tuple, so it is necessary to be able to translate a tuple into an index in the temporary storage and vice versa. This translation is performed by the functions *getTuple* and *getIndex* in Algorithm 8.

Counting probability entries, one may note that the memory requirement

Algorithm 7 – Weight Distribution (wd)

Input: S-box size n, vector length ℓ, current depth d, current probability p, probability distribution container $dist$ of length $\ell + 1$, weight w, number of opened table entries with zeros a_0, number of opened table entries with ones a_1.
Output: probability distribution $dist$.
Initial recursion parameters: $dist$ zeroized, $(d, p, w, a_0, a_1) = (0, 1, 0, 0, 0)$.

if (d==ℓ) { /* maximum depth reached */
  $dist[w]$ += p;
  return;
}
if ($a_0 > 0$) { /* old 0 reopened */
  $wd(dist, n, \ell, d + 1, p \cdot \frac{a_0}{n}, w, a_0, a_1)$;
}
if ($a_1 > 0$) { /* old 1 reopened */
  $wd(dist, n, \ell, d + 1, p \cdot \frac{a_1}{n}, w + 1, a_0, a_1)$;
}
if ($a_0 + a_1 < n$) { /* table not exhausted */
  $wd(dist, n, \ell, d + 1, p \cdot \frac{n-(a_0+a_1)}{2n}, w, a_0 + 1, a_1)$; /* new 0 */
  $wd(dist, n, \ell, d + 1, p \cdot \frac{n-(a_0+a_1)}{2n}, w + 1, a_0, a_1 + 1)$; /* new 1 */
}

Algorithm 8 – Weight Distribution (DP version)

Input: S-box size n, vector length ℓ.
Output: probability distribution *dist* of length $\ell + 1$.

$m = \frac{1}{2}(n+1)^2(\ell+1)$; /* max num prob. entries at depth n */
$p = (1, 0, \ldots, 0)$; /* length m */
$q = (0, 0, \ldots, 0)$; /* length m */
for ($d = 0$; $d < \ell$; d++) { /* depth d */
  $m_d = \frac{1}{2}(n+1)^2(d+1)$; /* max num prob. entries at depth d */
  for ($e = 0$; $e < m_d$; e++) { /* entry index e */
    $(w, a_0, a_1) = getTuple(d, e)$;
    /* old 0 with probability $\frac{a_0}{n}$ */
    $q[getIndex(d+1, (w, a_0, a_1))]$ += $p[e] \cdot \frac{a_0}{n}$;
    /* old 1 with probability $\frac{a_1}{n}$ */
    $q[getIndex(d+1, (w+1, a_0, a_1))]$ += $p[e] \cdot \frac{a_1}{n}$;
    if ($a_0 + a_1 < n$) { /* table not exhausted */
      /* new 0 with probability $\frac{n-(a_0+a_1)}{2n}$ */
      $q[getIndex(d+1, (w, a_0+1, a_1))]$ += $p[e] \cdot \frac{n-(a_0+a_1)}{2n}$;
      /* new 1 with probability $\frac{n-(a_0+a_1)}{2n}$ */
      $q[getIndex(d+1, (w+1, a_0, a_1+1))]$ += $p[e] \cdot \frac{n-(a_0+a_1)}{2n}$;
    }
  }
  $p = q$; /* copy q to p (or swap buffers) */
  $q = (0, 0, \ldots, 0)$; /* clear q */
}
$dist = (0, 0, \ldots, 0)$; /* length $\ell + 1$ */
for ($e = 0$; $e < m$; e++) {
  $(w, a_0, a_1) = getTuple(m, e)$;
  $dist[w]$ += $p[e]$;
}
return *dist*;

of this implementation is $2m = (n+1)^2(\ell+1)$. However, this memory usage depends on how well the functions *getTuple* and *getIndex* are implemented. Better upper bound formulas for minimal time and memory complexity are given by

$$\frac{n^2\ell^2}{4} \quad \text{and} \quad \frac{n^2\ell}{2},$$

respectively. These formulas still leave some room for improvement. Depending on the accuracy requirements of the application[2], built-in floating point types or multiple precision data types may be employed.

The DP adaptation of Algorithm 7 to Algorithm 8 is rather straightforward, and we will soon see that it can be improved even further.

We now explicitly prove that sampling techniques VS and WS are equivalent in terms of keystream complexity of the resulting distinguisher. We first present Algorithm 9 which calculates the probability of an S-box outputting a specific vector—the vector probability. The correctness of Algorithm 9 follows from its relationship to Algorithm 6.

Theorem 4.3 (WS is optimal) WS is equivalent to VS in terms of the Neyman-Pearson test (Lemma 2.1).

Proof. The proof follows if we can show that all vectors of equal weight are equiprobable, because the probability distributions P_1 and P_2 for WS can then be derived from those of VS by grouping all probabilities for vectors of equal weight. In such a case the Neyman-Pearson test is equal for both sampling techniques, showing that no information is lost when considering WS over VS.

It is sufficient to show that the vector probability is invariant under pairwise bit flips. That is, we need to show that the vector probability does not change if a neighboring pair of bits in a vector are flipped from 10 to 01 (or from 01 to 10).

Let $v = (s_1, s_2, \ldots, s_\ell)$ be a vector for which $s_i = 0$ and $s_{i+1} = 1$ for some i, and let v' be the corresponding vector with $s'_i = 1$ and $s'_{i+1} = 0$. Let v_j denote the vector $(s_j, s_{j+1}, \ldots, s_\ell)$. We need to show that $vp(p, v, a_0, a_1) = vp(p, v', a_0, a_1)$ (we omit some of the less interesting parameters).

All recursive calls to $vp(p, v, a_0, a_1)$ and $vp(p, v', a_0, a_1)$ are identical up to depth i, so it is enough to consider any two such calls $vp(p, v_i, a_0, a_1)$ and $vp(p, v'_i, a_0, a_1)$ at depth i. We need to show that both these calls give rise to the same quadruple of function calls at depth $i+2$, two levels deeper.

[2]We found the precision of 64-bit data types insufficient for the calculations performed in Section 4.4.

Algorithm 9 – Vector Probability (vp)

Input: vector probability $prob$ (accumulated), current probability p, S-box size n, vector length t, vector v (s_1 at LSB), number of opened table entries with zeros a_0, number of opened table entries with ones a_1.
Output: vector probability $prob$.
Initial recursion parameters: $(prob, p, t, a_0, a_1) = (0, 1, \ell, 0, 0)$.

if $(t == 0)$ { /* maximum depth reached */
  $prob \mathrel{+}= p$;
  return;
}
if $(v \;\&\; 1)$ { /* next output bit is 1 */
  if $(a_1 > 0)$ { /* old 1 reopened */
    $vp(prob, p \cdot \frac{a_1}{n}, n, v \gg 1, t-1, a_0, a_1)$;
  }
  if $(a_0 + a_1 < n)$ { /* table not exhausted */
    $vp(prob, p \cdot \frac{n-(a_0+a_1)}{2n}, n, v \gg 1, t-1, a_0, a_1+1)$; /* new 1 */
  }
} **else** { /* next output bit is 0 */
  if $(a_0 > 0)$ { /* old 0 reopened */
    $vp(prob, p \cdot \frac{a_0}{n}, n, v \gg 1, t-1, a_0, a_1)$;
  }
  if $(a_0 + a_1 < n)$ { /* table not exhausted */
    $vp(prob, p \cdot \frac{n-(a_0+a_1)}{2n}, n, v \gg 1, t-1, a_0+1, a_1)$; /* new 0 */
  }
}

$vp(p, v_i, a_0, a_1)$ splits into

$$vp(p \cdot \frac{a_0}{n}, v_{i+1}, a_0, a_1) \text{ and}$$
$$vp(p \cdot \frac{n - a_0 - a_1}{2n}, v_{i+1}, a_0 + 1, a_1)$$

at level $i + 1$, and then into

$$vp(p \cdot \frac{a_0 a_1}{n^2}, v_{i+2}, a_0, a_1),$$
$$vp(p \cdot \frac{a_0(n - a_0 - a_1)}{2n^2}, v_{i+2}, a_0, a_1 + 1),$$
$$vp(p \cdot \frac{(n - a_0 - a_1)a_1}{2n^2}, v_{i+2}, a_0 + 1, a_1) \text{ and}$$
$$vp(p \cdot \frac{(n - a_0 - a_1)(n - (a_0 + 1) - a_1)}{4n^2}, v_{i+2}, a_0 + 1, a_1 + 1)$$

at level $i + 2$. Similarly, $vp(p, v'_i, a_0, a_1)$ splits into

$$vp(p \cdot \frac{a_1 a_0}{n^2}, v'_{i+2}, a_0, a_1),$$
$$vp(p \cdot \frac{a_1(n - a_0 - a_1)}{2n^2}, v'_{i+2}, a_0 + 1, a_1),$$
$$vp(p \cdot \frac{(n - a_0 - a_1)a_0}{2n^2}, v'_{i+2}, a_0, a_1 + 1) \text{ and}$$
$$vp(p \cdot \frac{(n - a_0 - a_1)(n - a_0 - (a_1 + 1))}{4n^2}, v'_{i+2}, a_0 + 1, a_1 + 1).$$

Here we have $v_{i+2} = v'_{i+2}$, so the sets of calls are identical. ■

A direct consequence of the proof of Theorem 4.3 is that, although VS is highly impractical to use due to its exponential time and memory complexities, WS will provide the same result as VS at a much lower cost, allowing much larger ℓ-values to be used in practice.

We can also use Theorem 4.3 to improve the efficiency of Algorithm 8. Since all vectors of equal weight are equiprobable, we need only consider vectors of type $00 \ldots 011 \ldots 1$. The improved algorithm is to calculate the probabilities for all $\ell + 1$ such vectors by using a dynamic programming version of Algorithm 9. Recall that the time and memory complexities of Algorithm 8 are $O(n^2\ell^2)$ and $O(n^2\ell)$, respectively, so memory usage is limiting in practice. For the new algorithm we need only $O(\min(n, \ell))$ memory for storing intermediate probability values and $O(\ell)$ storage to hold the resulting probability distribution P_2. An additional improvement is to recognize that the distribution is symmetric in the sense that $P_2(i) = P_2(\ell - i)$ for $i = 0, \ldots, \lfloor \frac{\ell}{2} \rfloor$, so we only need to compute half of it.

Table 4.1: Comparison of online and offline time and memory complexities for a distinguisher using k chunks with various sampling techniques. S-box size n, vector length ℓ, number of chunks k.

	online		offline	
	time	memory	time	memory
APS	$O(k\ell^2)$	$O(1)$	$O(1)$	$O(1)$
LPS	$O(k\ell)$	$O(1)$	$O(1)$	$O(1)$
VS	$O(k\ell)$	$O(2^\ell)$	$O(n^2 2^\ell)$	$O(n^2 2^\ell)$
WS	$O(k\ell)$	$O(\ell)$	$O(n^2\ell^2)$	$O(\ell)$

While the time required is still $O(n^2\ell^2)$, the constants are better. Our simulations show that we save 80-85% in time, and the memory usage is $O(\ell)$, i.e., it no longer depends on the size of the S-box. This is optimal since it equals the length of the vector.

4.3.5 COMPARING SAMPLING TECHNIQUES

We have shown above that both APS and LPS are erroneous as the corresponding samples are not taken independently. Disregarding this deficiency, both techniques are very simple to apply. The distribution P_1 is very easy to compute in each case (no precomputation), and checking if $s_i = s_j$ is trivial. However, a practical drawback of employing APS and LPS is that the resulting distinguisher will be suboptimal, and we have no simple analytical rule for how many additional samples (compared to the optimal case) that we need to collect to ensure a decent distinguisher accuracy.

For optimality, a sampling technique like WS (or VS), that provides independent samples and preserves the information contained in an entire chunk, *must* be used. This optimality comes at the cost of a larger precomputational complexity, i.e., for computing P_1. Table 4.1 summarizes the important complexities corresponding to each sampling technique. Note that we assume that the best dynamic programming variant is used to compute the probability distributions P_1 for VS and WS. The actual performance of the attack using each of the sampling techniques has been simulated. As VS and WS give the exact same distinguisher performance, only WS is included in the simulations. For a fair comparison, we assume that the number of chunks is the same for all variants, i.e., APS and LPS are allowed to use many more

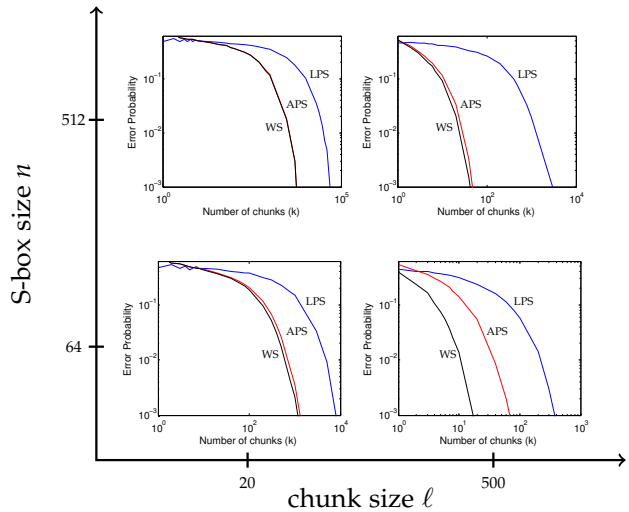


Figure 4.2: Error probability as a function of the number of chunks k.

samples than WS, but all use the same number of observations. The plots in Figure 4.2 shows the error probabilities for APS, LPS and WS as a function of the number of chunks k for S-box sizes $n = 64$ and $n = 512$ when the number of observations in each chunk is $\ell = 20$ and $\ell = 500$.

From the simulations we can see that both LPS and APS are outperformed by WS. It is interesting to see that APS is not very much worse when the same number of chunks is considered. However, we stress again that APS collects $\binom{\ell}{2}$ samples from a chunk, while WS collects only one. This clearly shows the problem of assuming independent samples when they are in fact very dependent.

Looking at Figure 4.2, it seems that the performance of APS approaches that of WS when the S-box size n increases and when the chunk size ℓ decreases. Thus, for large n and small ℓ their performances are practically equal, while for small n and large ℓ, WS clearly outperforms APS. We have simulated many other choices of n and ℓ, and all simulations showed the same tendency.

4.4 A VECTOR WEIGHT DISTINGUISHER FOR HC-128

We can now construct the best known distinguisher for HC-128 using the optimal vector weight sampling technique, and we will see that it is possible to obtain a distinguisher that requires only $2^{152.6}$ 32-bit keystream words.

Now recall Equations (4.2) and (4.4) from Section 4.2.3, which we restate

here for convenience,

$$t_i = [s_i]^0 \oplus [s_{i-3}]^{10} \oplus [s_{i-10}]^8 \oplus [s_{i-1023}]^{23} \oplus [s_{i-1024}]^0 \tag{4.8}$$

$$= [h_1(z_i)]^0 \oplus [h_1(z_{i-3})]^{10} \oplus \tag{4.9}$$
$$[h_1(z_{i-10})]^8 \oplus [h'_1(z_{i-1023})]^{23} \oplus [h'_1(z_{i-1024})]^0.$$

The bit t_i can be assembled from keystream bits by following the recipe given by Equation (4.8), and according to Equation (4.9), each t_i is derived from several lookups into the same table Q (or P). Table Q (or P) contains 512 32-bit words, and lookup into Q (or P) is divided into the upper and lower half according to function h_1 (or h_2). Note that h_1 and h'_1 denote lookup into different tables, and note also that we only use one bit from each 32-bit lookup value for t_i so that the table bit positions differ for each application of h_1. Thus, t_i is essentially derived by xoring lookups from ten *different* lookup tables of size $n = 256$ (with binary entries). One may view t_i as the result of xoring the output of ten different 8-to-1-bit S-boxes. The conceptual composition of t_i is illustrated in Figure 4.3.

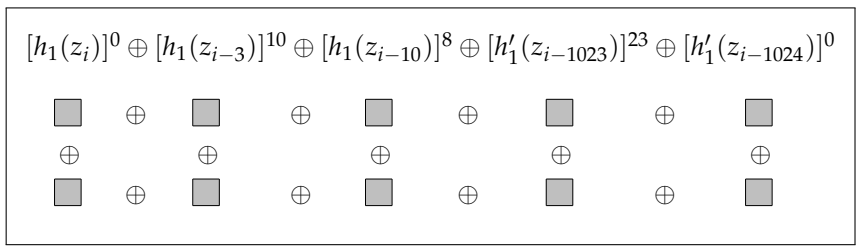


Figure 4.3: Conceptual anatomy of t_i.

HC-128 reuses the same tables with fixed values during several consecutive time instances, so t_i and t_j with $i \neq j$ are pairwise dependent within this time period, which lasts for 501 clockings.

The main idea now is use vector weight sampling by forming vectors containing these 501 consecutive t_i's. Using the notation in Section 4.3, we have S-box size $n = 256$ and vector length $\ell = 501$ for HC-128 in this analysis.

A major obstacle remains. Each t_i is not the output of an 8-to-1-bit S-box (or table)—it is the output of *10 xored* 8-to-1-bit S-boxes.

We can use Algorithm 8 (as is or the improved version) to calculate the probability distribution for the vector weight in the case of one single S-box, but we need to learn how to combine several of these probability distributions with respect to the xor operator. That is, if we have two ℓ-bit words X and Y

Algorithm 10 – Xoring Vector Weight Probability Distributions

Input: vector length n, vector weight probability distributions p and q ($n+1$ values each).
Output: vector weight probability distribution r.

```
r = (0, 0, 0, ..., 0); /* length n + 1 */
for (i = 0; i < n + 1; i++) {
   for (j = 0; j < n + 1; j++) {
      for (k = max(0, i + j − n); k < min(i, j) + 1; k++) {
         /* vector v1 with weight i,
          * vector v2 with weight j,
          * k ones overlapping ⟹ v1 ⊕ v2 has weight i + j − 2k */
         r[i + j − 2k] += p[i] · q[j] · C(i,k)·C(n−i,j−k) / C(n,j);
      }
   }
}
return r;
```

with weight probability distributions P_X and P_Y, respectively, what does the weight probability distribution $P_Z = P_X \oplus P_Y$ of $Z = X \oplus Y$ look like?

Algorithm *XorDistributions* in Algorithm 10 shows how to combine two probability distributions. Given two different S-boxes, potentially with different weight probability distributions, we still know precisely what to expect from their xored output in terms of vector weight.

The algorithm works as follows. Consider vectors v_1 and v_2 with weights i and j, respectively. If k ones are overlapping, the xored vector $v_1 \oplus v_2$ will contain $i + j - 2k$ ones. The algorithm sums the probabilities over all possible tuple values (i, j, k). The hypergeometric expression $\frac{\binom{i}{k}\binom{n-i}{j-k}}{\binom{n}{j}}$ in the algorithm states the probability of a k-bit overlap given two vectors of weight i and j.

Algorithm 10 defines xor between weight probability distributions, so we can now easily and successively deduce the probability distributions for the resulting vector weight over, in turn, 2, 4, 8 and 10 tables. The final weight probability distribution P_1 is given by

$$P_1 = \bigoplus_{i=1}^{10} P_1',$$

where P_1' is the weight probability distribution of a single 8-to-1-bit S-box, determined using Algorithm 8.

Also recall the uniform weight probability distribution P_2 for vectors of length ℓ (regardless of S-box size) from Section 4.3.4. It is given by

$$P_2(w) = \binom{\ell}{w} 2^{-\ell}.$$

Last but not least, we need to compare P_1 and P_2, and we do this by using the divergence measure according to Equation (2.1). For various vector lengths ℓ and number of S-boxes, the required number of samples for our distinguisher is presented in Table 4.2. As a verification, the case $\ell = 10$ over one S-box was also simulated to ensure correctness. The simulation results matched the theoretical findings.

Table 4.2: Number of samples required for various vector lengths n and number of S-boxes.

		vector length ℓ					
		2	10	50	100	250	501
# S-boxes	1	$2^{16.471}$	$2^{11.009}$	$2^{6.380}$	$2^{4.517}$	$2^{2.232}$	$2^{0.665}$
	2	$2^{32.471}$	$2^{26.979}$	$2^{22.213}$	$2^{20.199}$	$2^{17.549}$	$2^{15.544}$
	4	$2^{64.471}$	$2^{58.979}$	$2^{54.213}$	$2^{52.198}$	$2^{49.545}$	$2^{47.537}$
	8	$2^{128.471}$	$2^{122.979}$	$2^{118.213}$	$2^{116.198}$	$2^{113.545}$	$2^{111.537}$
	10	$2^{160.471}$	$2^{154.979}$	$2^{150.213}$	$2^{148.198}$	$2^{145.545}$	$2^{143.537}$

Using this analysis, the actual distinguishing attack on HC-128 would proceed as follows. Instantiate $\ell + 1$ counters, one for each possible vector weight. From all keystream words, compute the ℓ-bit vector samples by repeatedly applying Equation (4.8). Based on the resulting vector weight, increase the corresponding counter. When finished counting, perform the hypothesis test on the resulting distribution (the counter values) to see if the biased HC-128 or the uniform probability distribution yields the closest fit.

Our best result for the full HC-128 with 10 S-boxes is a requirement of $2^{143.6}$ samples for $\ell = 501$. Each sample is derived from the least significant bits of ℓ consecutive keystream words, and the total complexity is at most $2^{152.6}$ keystream words.

The reader may further note that the time units used here correspond to very simple operations involving only xor and shifts of keystream words. These operations are *much* cheaper than the initializations considered in a brute-force search. Key initialization takes about 27300 clock cycles [Wu08],

involving both a key expansion and 1024 table updates. For comparison, one single key initialization corresponds to processing about 2^{10} keystream words using our simple operations, so the complexity of the distinguisher in terms of initializations is no more than 2^{143}.

4.5 ANALYSIS OF THE XORROTATION FAMILY

Operations of and observations on HC-128 have inspired our analysis of the Xorrotation family. We find this analysis is interesting on its own, but in Section 4.7 we will show how the theory can be used in practice by building a distinguisher for HC-128$^{\oplus}$ (defined in Section 4.2.2).

Consider first the function

$$f_{w,r}(x) = x \oplus (x \lll r),$$

where x is a w-bit variable and $\lll$ denotes left rotation with respect to the word length w. For all rotation amounts r in this section we enforce the constraint $0 < r < w$. This construction is often a basic mixing component in cryptographic primitives [ST12, Wu08, Wu04, Riv+09, DL08, Hon+06]. We take a probability distribution approach here and provide a set of lemmas that are practical for cryptanalysis.

For a distribution to be of use to an analyst, it needs to boast a high divergence. This makes it easily distinguishable from a uniform distribution. In this context, all divergences of magnitude 1 and above are extremely high.

In the following we assume w-bit words, and we number the bit positions from least- to most significant bit 0 through $w - 1$.

Definition 4.1 (Probability distribution operator E) A mapping $f : U \longrightarrow V$ is said to *generate* a probability distribution on V (uniformly) in the following way. Starting with an empty array of size $|V|$, let each $x \in U$ contribute probability $2^{-|U|}$ to slot $f(x)$. Summation over all possible domain values produces the probability distribution in question. The probability distribution generated by f is denoted $E(f)$.

If we generalize $f_{w,r}$ by defining $f_{w,r_1,\ldots,r_n}$ according to

$$f_{w,r_1,\ldots,r_n}(x_1,\ldots,x_n) = f_{w,r_1}(x_1) \oplus \cdots \oplus f_{w,r_n}(x_n),$$

then $E(f_{w,r})$ and $E(f_{w,r_1,\ldots,r_n})$ denote the probability distributions generated by $f_{w,r}$ and $f_{w,r_1,\ldots,r_n}$, respectively.

Definition 4.2 (r-orbit) In a w-bit word, the bit positions reachable from bit position i as we apply r-bit rotation again and again—the *orbit* of bit position i under r-bit rotation—is given by the bit set

$$\{(i + kr) \bmod w \mid k \in \mathbb{N}\},$$

and there are $\gcd(w,r)$ distinct orbits, each of length $\frac{w}{\gcd(w,r)}$.

Proposition 4.4 (Divergence of $E(f_{w,r})$) The divergence of the probability distribution $E(f_{w,r})$ is $\gcd(w,r)$.

Proof. For every given w-bit output value $y = y_{w-1} \dots y_0$, the equation system

$$f_{w,r}(x) = y$$

has $2^{\gcd(w,r)}$ solutions. That is, restricting the domain and range of $f_{w,r}$ to only one r-orbit, that corresponding equation system has precisely two solutions, and the restricted mapping is consequently 2-to-1. There are $\gcd(w,r)$ disjoint r-orbits, so the entire mapping $f_{w,r}$ is $2^{\gcd(w,r)}$-to-1.

From this it follows that the probability distribution $E(f_{w,r})$ has precisely $2^{w-\gcd(w,r)}$ nonzero probability entries, each being equal to $2^{\gcd(w,r)-w}$ since x is uniformly distributed over the domain. Using Equation (2.1) we get

$$\begin{aligned} D(E(f_{w,r}) \| U) &= 2^{w-\gcd(w,r)} \left(2^{\gcd(w,r)-w} \log \frac{2^{\gcd(w,r)-w}}{2^{-w}} \right) \\ &= \gcd(w,r). \end{aligned}$$

■

We state a generalized version as Theorem 4.5.

Theorem 4.5 (Divergence of $E(f_{w,r_1,\dots,r_n})$) The divergence of the probability distribution $E(f_{w,r_1,\dots,r_n})$ is $\gcd(w, r_1, r_2, \dots, r_n)$.

Proof. We prove the case $n = 2$, leaving $n > 2$ to the ambitious reader. We will first show that f_{w,r_x,r_y} is $2^{w+\gcd(w,r_x,r_y)}$-to-1. We need to show that, for any given w-bit result $z = z_{w-1} \dots z_0$, the equation system

$$f_{w,r_x,r_y}(x,y) = z$$

has exactly $2^{w+\gcd(w,r_x,r_y)}$ solutions in x and y.

Dividing the domain and range of f_{w,r_x,r_y} into $\gcd(w,r_x,r_y)$-orbits, the resulting equation systems are independent. It is therefore sufficient to show that each of these systems have exactly 2^{d+1} solutions, where $d = \frac{w}{\gcd(w,r_x,r_y)}$.

Now let $f_{w,r_x,r_y}(x,y) = z$ with $\gcd(w,r_x,r_y)$-orbit length d and $r_x \geq r_y$. Writing out the corresponding equation system explicitly we have

$$\begin{cases} x_0 + x_{-r_1} + y_0 + y_{-r_2} = z_0, \\ x_1 + x_{1-r_1} + y_1 + y_{1-r_2} = z_1, \\ \vdots \quad\quad \vdots \quad\quad \vdots \quad\quad \vdots \quad\quad \vdots \\ x_{d-1} + x_{d-1-r_1} + y_{d-1} + y_{d-1-r_2} = z_{d-1}, \end{cases}$$

with indices taken modulo d. Choosing the set

$$V = \{x_0, \ldots, x_{\gcd(w,r_x)-1}\} \cup \{y_0, \ldots, y_{d-\gcd(w,r_x)}\},$$

of $d+1$ free variables, we obtain a unique solution to the equation system for every given assignment to z and the variables in V. This can be seen by solving for, in turn, $x_{\gcd(w,r_x)}, \ldots, x_{d-\gcd(w,r_x)}$ followed by interleaving the remaining variables according to $y_{d-\gcd(w,r_x)+1}, x_{d-\gcd(w,r_x)+1}, \ldots, y_{d-1}, x_{d-1}$. This shows that f_{w,r_x,r_y} is 2^{d+1}-to-1 in every $\gcd(w, r_x, r_y)$-orbit, and thus $2^{w+\gcd(w,r_x,r_y)}$-to-1 with the full unrestricted domain and range.

Since the x and y's are uniformly distributed, the probability distribution $E(f_{w,r_x,r_y})$ has precisely $2^{w-\gcd(w,r_x,r_y)}$ nonzero probability entries, each of these being equal to $2^{\gcd(w,r_x,r_y)-w}$. Thus, Equation (2.1) gives us

$$\begin{aligned} D(E(f_{w,r_x,r_y})\|U) &= 2^{w-\gcd(w,r_x,r_y)} \left(2^{\gcd(w,r_x,r_y)-w} \log \frac{2^{\gcd(w,r_x,r_y)-w}}{2^{-w}} \right) \\ &= \gcd(w, r_x, r_y). \end{aligned}$$

■

For the sake of a deepened RX analysis along the lines of Thomsen [TK09] and Rivest [Riv11], we now generalize in another direction and consider the function

$$g_{w,r_1,\ldots,r_n}(x) = x \oplus (x \lll r_1) \oplus \cdots \oplus (x \lll r_n)$$

with $n+1$ clauses (the first one degenerated, zero rotation). For easy reference, we restate the result of Thomsen and Rivest using our terminology and notation.

Theorem 4.6 (Uniform $E(g_{w,r_1,\ldots,r_n})$-distributions for $w = 2^k$) Let $w = 2^k$ for some integer $k \geq 2$. Independently of the values $r_1, \ldots, r_n$, the probability distribution $E(g_{w,r_1,\ldots,r_n})$ is uniform if n is even.

Rivest's proof of Theorem 4.6 is actually more general than it appears to be. It shows how to check if a particular set of rotation values results in a permutation on w-bit words. As far as categorization goes, however, we would still like to see a complete (and preferably simple) set of rules that show what we can expect for *all* values of w and n. Theorem 4.6 shows the case $w = 2^k$ for even n but leaves the remaining (w, n)-space open. We now present a few results that fill in some of the remaining blanks. Note that Proposition 4.4 fully categorizes the case $n = 1$ for all w.

Proposition 4.7 (Using all rotation amounts is a bad idea) For all w, using all rotation values, the divergence of $E(g_{w,1,2,\ldots,w-1})$ is $w-1$.

Proof. When all rotation amounts are used, $g_{w,r_1,\ldots,r_n}$ maps x to either $000\ldots0$ or $111\ldots1$ depending on if the weight (the number of one-bits) of x is even or odd, respectively. The function $g_{w,r_1,\ldots,r_n}$ is therefore 2^{w-1}-to-1 and yields a divergence of $w-1$. ■

Proposition 4.8 (Divergence of $E(g_{w,r_1,\ldots,r_n})$ for an even number of clauses) For all w, the divergence of $E(g_{w,r_1,\ldots,r_n})$ is ≥ 1 if n is odd.

Proof. Let n be odd. We have $g_{w,r_1,\ldots,r_n}(x) = g_{w,r_1,\ldots,r_n}(\overline{x})$, where $\overline{x}$ denotes the binary complement of x. This shows that $g_{w,r_1,\ldots,r_n}$ is at least 2-to-1. The divergence of $E(g_{w,r_1,\ldots,r_n})$ must therefore be at least 1. ■

It is interesting to note that a uniform distribution (zero divergence) is only attainable if an odd number of clauses (even n) are used. As Proposition 4.8 indicates, cryptographic designers generally fare better using a mixing function with three clauses rather than two or four.

In order to show the next theorem we need to make a jump to a polynomial representation of words and rotations so that we operate in $\mathbb{GF}(2)[x]/(h(x))$ with $h(x) = x^w + 1$ as follows. Let the w-bit word $v = v_{w-1}v_{w-2}\ldots v_0$ be represented by the associated polynomial

$$v(x) = \sum_{i=0}^{w-1} v_i x^i.$$

Left rotation by r bits then corresponds to multiplication with x^r modulo $h(x)$, so applying $g_{w,r_1,\ldots,r_n}$ corresponds to multiplication with

$$r(x) = 1 + x^{r_1} + x^{r_2} + \cdots + x^{r_n}$$

modulo $h(x)$. In this model, showing that $r(x)$ is invertible amounts to showing that $\gcd(r(x), h(x)) = 1$. And an invertible $r(x)$ implies that $g_{w,r_1,\ldots,r_n}$ is a permutation, which tells us that the divergence of $E(g_{w,r_1,\ldots,r_n})$ is zero. The reader is referred to [Riv11] for further details.

Definition 4.3 (Perfect word length) Let $n < w-1$, and let U denote the uniform probability distribution corresponding to $E(g_{w,r_1,\ldots,r_n})$. The word length w is

even-perfect if $D(E(g_{w,r_1,\ldots,r_n})\|U) = 0$ for *all* choices of r_i and even n,

odd-perfect if $D(E(g_{w,r_1,\ldots,r_n})\|U) = 1$ for *all* choices of r_i and odd n,

perfect if it is both even-perfect and odd-perfect.

Theorem 4.6 states that the word lengths $2^k, k \geq 2$, are even-perfect. But these are not the only ones. It can also be shown that the following word lengths below 64 are even-perfect;

$$3, 5, 11, 13, 19, 29, 37, 53, 59 \text{ and } 61. \tag{4.10}$$

We first recall Definition 3.5, which we restate here in specialized form ($e = 1$) for convenience.

Definition 4.4 (Primitive root modulo p) Let p be an odd prime. A primitive root modulo p is a number $0 < g < p$ such that g generates the multiplicative group $(\mathbb{Z}/p\mathbb{Z})^*$.

As before, if $g = 2$ is a primitive root modulo p, then the sequence of numbers $2, 2^2, 2^3, \ldots, 2^{p-1}$ taken modulo p runs through all invertible elements of $\mathbb{Z}/p\mathbb{Z}$, or equivalently, runs through all numbers in the interval $[1, p-1]$ in some order.

In particular, not all primes have 2 as a primitive root. The first few primes for which 2 is a primitive root are given in (4.10) above, see Sloane's $A001122$ [Slo11].

We can now show a generalized result for even-perfect word lengths in Theorem 4.9.

Theorem 4.9 (Even-perfect word lengths) The primes with primitive root 2 are even-perfect word lengths.

Proof. Let $h(x) = x^w + 1$ where w is a prime with primitive root 2. We need to show that

$$r(x) = 1 + x^{r_1} + x^{r_2} + \cdots + x^{r_n}$$

is invertible in $\mathbb{GF}(2)[x]/(h(x))$. Since w is a prime with primitive root 2, $h(x)$ factors irreducibly into $(x+1)(x^{w-1} + x^{w-2} + \cdots + x + 1)$. The polynomial $r(x)$ cannot have a factor $x + 1$ since $r(1) = 1$ when n is even. Also, since $n < w - 1$, $r(x)$ cannot be the maximum weight polynomial. We therefore have $\gcd(r(x), h(x)) = 1$, showing that $r(x)$ is invertible. ■

Our simulations lend evidence toward the following conjectures, which we state to inspire future work in this direction. We start by generalizing Theorems 4.6 and 4.9, stating that the even-perfect word lengths we have seen above are also odd-perfect.

Conjecture 4.10 (Perfect word lengths) The following word lengths are perfect:

- powers of 2 ($2^k, k \geq 2$), and

- primes with primitive root 2.

One might further attempt to prove that no other word lengths can be perfect.

Conjecture 4.11 (Existence of optimal rotation sets) For all w and n with $n < w - 1$, a set of rotation values can be found such that the divergence of $E(g_{w,r_1,\ldots,r_n})$ is optimal (0 if n is even and 1 if n is odd).

A proof of Conjecture 4.11 would show that there exist irreducible polynomials in $\mathbb{GF}(2)[x]/(x^w + 1)$ for *all* w and even n. This is by no means a trivial result, and a proof would constitute an advancement in group and ring theory. If Conjecture 4.11 can be proven, it is also reasonable to try to extend the result to $\mathbb{GF}(q)[x]/(x^w + 1)$.

The most interesting and somewhat related result of this kind that we have found is due to Stepanov [Ste85, Ste87] and regards the number of irreducible polynomials of given form over $\mathbb{GF}(q)[x]$, showing an asymptotic formula for when $q \to \infty$.

We expect the following conjecture to be much easier to prove.

Conjecture 4.12 (All-but-one is optimal) For all w, when using all rotation values except one ($n = w - 2$), the divergence of $E(g_{w,1,2,\ldots,w-1})$ is optimal (0 if n is even and 1 if n is odd).

4.6 DIVERGENCE OF PROBABILISTICALLY BIASED DISTRIBUTIONS

In the analysis of HC-128$^{\oplus}$ in Section 4.7, we will encounter probabilistic equalities. These are equalities that are true with some given minimum probability. We will annotate the equality sign of probabilistic equations with their corresponding probability.

We will be using probabilistic equalities in conjunction with the divergence measure D, so we need to determine how the former influence the latter.

Definition 4.5 (Probabilistically biased distribution) Let A be any distribution of size 2^w, and let U be the uniform distribution of the same size. A distribution resulting from sampling A with probability p and U with probability $1 - p$ is said to be probabilistically biased with parameters (p, A), or (p, A)-biased.

Theorem 4.13 (Probabilistic divergence) The divergence of a (p, A)-biased distribution is $p^2 D(A\|U)$.

We limit our proof of Theorem 4.13 to approximation by Taylor series, but the full result should follow directly from a more careful handling and bounding of the involved error terms. However, for our purposes it is sufficient that

the ε_i defined below are small enough, meaning that the Taylor series provides a reasonable approximation.

Proof. Let A be a distribution of size 2^w biased according to

$$A[i] = 2^{-w} + \varepsilon_i,$$

and let U denote the uniform distribution of same size. Letting

$$\varepsilon^2 = 2^w \sum_{i=0}^{2^w-1} \varepsilon_i^2,$$

we can use Taylor series according to Example 2.1 to approximate $D(A\|U)$ as

$$D(A\|U) \approx \frac{\varepsilon^2}{2\ln 2} = \frac{2^w \sum_{i=0}^{2^w-1}(A[i] - 2^{-w})}{2\ln 2}.$$

Now let B be a (p, A)-biased distribution. B is biased with probabilities

$$B[i] = pA[i] + (1-p)U[i] = pA[i] + (1-p)2^{-w} = 2^{-w} + \varepsilon_i^{'}.$$

So,

$$\varepsilon_i^{'} = p(A[i] - 2^{-w})$$

and

$$D(B\|U) \approx p^2 \frac{2^w \sum_{i=0}^{2^w-1}(A[i] - 2^{-w})}{2\ln 2} \approx p^2 D(A\|U). \qquad \blacksquare$$

4.7 APPLYING XORROTATION ANALYSIS TO HC-128$^{\oplus}$

We now illustrate how Theorems 4.13 and 4.5 can be applied in a beautiful way to produce a new distinguisher for HC-128$^{\oplus}$. In particular, this will show that HC-128 becomes weak if its $+$ operators are replaced with $\oplus$.

The reader should be particularly careful here not to be fooled into thinking that attacking this construction is a trivial task. Despite the removal of the nonlinearity provided by modular addition, it is still *not* easy to construct low complexity distinguishers for HC-128$^{\oplus}$. This is because we still have to deal with the S-boxes.

Table updates in the original HC-128 are performed according to

$$P_i = P_i + (P_{i-3}^{10} \oplus P_{i-511}^{23}) + P_{i-10}^{8}.$$

During the first half session we have 256 table updates with keystream generation

$$s_i = (Q_a + Q_b) \oplus P_i, \tag{4.11}$$

where $0 \leq i, a \leq 255$ and $256 \leq b \leq 511$. Similarly, the second half session provides 256 table updates with keystream generation

$$s_j = (Q_c + Q_d) \oplus P_j, \tag{4.12}$$

where $0 \leq c \leq 255$ and $256 \leq j, d \leq 511$. This completes one full update of table P. The subsequent session updates table Q, and for the three first updates we have

$$\begin{aligned} s_k &= (P_l + P_m) \oplus Q_k \\ &= (P_l + P_m) \oplus (Q_e + (Q_f^{10} \oplus Q_g^{23}) + Q_h^8), \end{aligned} \tag{4.13}$$

with $0 \leq l, e, g \leq 255$ and $256 \leq m, f, h \leq 511$. The P's and Q's in Equations (4.11), (4.12) and (4.13) denote lookups into the same tables.

Now consider the HC-128 variant for which all $+$ operators are replaced by $\oplus$. Choosing any one equation triplet from Equations (4.11), (4.12) and (4.13), we have $i = l$ and $j = m$ (and thus $P_i = P_l$ and $P_j = P_m$) each with probability 2^{-8}. We also have $a = e, c = g$ or $a = g, c = e$ with combined probability 2^{-15} (assume $a = e, c = g$ without loss of generality). We similarly have $b = f, d = h$ or $b = h, d = f$ with combined probability 2^{-15} (assume $b = h, d = f$). Using (4.11) and (4.12) linearly together with all the equations (4.13) gives us 3×256 equation triplets for every 512 keystream words. With probability $\frac{3 \times 256}{512} \times 2^{-46} > 2^{-45.42}$ we therefore have

$$s_i \oplus s_j \oplus s_k \overset{2^{-45.42}}{=} \underbrace{(Q_b \oplus Q_b^8)}_{N_1} \oplus \underbrace{(Q_c \oplus Q_c^{23})}_{N_2} \oplus \underbrace{(Q_d \oplus Q_d^{10})}_{N_3}, \tag{4.14}$$

where the left-hand side consists of known keystream words only, and N_1, N_2 and N_3 are observations from $E(f_{32,8})$, $E(f_{32,23})$ and $E(f_{32,10})$, respectively. According to Theorem 4.5, their combined distribution $E(f_{32,8,23,10})$ has divergence $\gcd(32, 8, 23, 10) = 1$. Equation (4.14) shows that we have a $(2^{-45.42}, E(f_{32,8,23,10}))$-biased distribution, which according to Theorem 4.13 results in a divergence of about $2^{-90.9} \times \gcd(32, 8, 23, 10) = 2^{-90.9}$. This yields a distinguisher requiring roughly $2^{90.9}$ 32-bit keystream words.

If we use evaluation of the left-hand side of Equation (4.14) over all three k-values—four xor operations on 32-bit keystream words—as time unit, we obtain a time complexity of $2^{89.9}$. In absolute terms, this measure is *much* cheaper, a factor of at least 2^{10}, than the cost of an initialization. For comparison, if we were to consider initializations instead, the time complexity of our distinguisher would be less than 2^{80}.

From all of the above it is very clear that the $+$ operator plays a vital role in HC-128.

4.8 CONCLUDING REMARKS

Four different sampling techniques for random S-box outputs have been considered and analyzed. We have proved that it is not possible to take two independent samples from one random S-box chunk, which implies that APS and LPS are suboptimal as they impose a higher error probability on the resulting distinguisher. We have also proved that WS is equivalent to the optimal VS, and that WS is much more practical than VS. WS is thus the preferred sampling technique. We also presented an improved algorithm for the offline computation of P_1 for WS.

Even though APS and LPS are not optimal, they are very simple to apply. However, it is not immediately clear how to compensate for the dependency between the samples. For large S-boxes that are frequently rerandomized, the APS technique is very close to optimal, meaning that the samples are in some sense "almost" independent in this case.

Using the optimal WS technique together with nontrivial algorithms for computing the relevant probability distributions, we constructed a distinguisher for the eSTREAM portfolio stream cipher HC-128. We have shown that the distinguisher has a data complexity of at most $2^{152.6}$ keystream words, and a time complexity that corresponds to about 2^{143} initializations. This is the most efficient distinguishing result known for HC-128. It exploits more information than any other distinguisher, and it is an open problem if it is possible to improve this result even further.

Last but not least, we also presented some new and general theoretical results on probability distributions related to RX-systems. We showed how to apply the new theory to a nontrivial system that uses RX operations in combination with S-boxes. We did this by building a new distinguisher for HC-128$^{\oplus}$. The total time complexity of the new distinguisher is $2^{89.9}$ very simple operations (xor and comparison of 32-bit keystream words) and the distinguisher requires about $2^{90.9}$ 32-bit keystream words, corresponding to at most 2^{80} initializations. This highlights the crucial part that the $+$ operator plays in HC-128.

5

Greedy Algebraic Cryptanalysis

This chapter is derived from the article »Greedy Distinguishers and Nonrandomness Detectors« [Sta10b], which was written solely by ourselves and presented at INDOCRYPT in 2010. A prequel to this paper was presented at the ECRYPT II workshop Tools for Cryptanalysis in 2010 as the conference abstract »Automated Algebraic Cryptanalysis« [Sta10a].

5.1 INTRODUCTION

The title of this chapter may get us convicted of false advertising if we do not explain it. In the cryptographic community, it seems that the word algebraic has taken the refined meaning of solving a set of equation systems. The meaning intended here—symbolic manipulation of variables—lies closer to the original interpretation of the word. We will not be solving equation systems. Instead, we will be examining cipher output in terms of boolean functions.

The output of a sensibly designed cipher should appear random to an external observer. Given a random-looking bit sequence, that observer should not be able to tell if the sequence is genuinely produced by the cipher in question or not. This simple idea is the core of cryptographic distinguishers and nonrandomness detectors.

Recently we have seen several attempts at finding distinguishers and nonrandomness detectors and many of the best ones seem to be built using the maximum degree monomial test (see [Saa06, EJT07]) or some derivative of it. This test is superb for detecting nonrandomness, and it also provides a window into the internals of the cryptographic algorithm we are examining. The maximum degree monomial test can provide statements such as *The IV bits*

are not mixed properly, which can be invaluable to the algorithm designer.

The core of this test is a bit set, and the efficiency of the test is largely determined by how this bit set is selected. For this selection process, it seems that guesswork has been the most prominent ingredient. The reason for this may be that systematic methods have seemed too complicated to find or use, or simply that the importance of bit set selection has been underestimated. By far, the best systematic approach we have seen so far was due to Aumasson et al. [ADMS09]. They used a genetic algorithm to select a bit set, and this is a very reasonable approach for unknown and complex search spaces. The complexity of the search space depends on the algorithm we are examining, but are they really so complex that we need to resort to such methods? In this chapter we present a very simple deterministic and systematic approach that outperforms all other methods we have seen so far. We call it the Greedy Bit Set Algorithm.

Stream ciphers have an initialization phase, during which they warm up for a number of rounds before they are deemed operational. Block ciphers are not explicitly initialized in this way, but they do operate in rounds. For our purposes, this can be translated into an initialization phase.

How many rounds are needed to warm up properly? This is a question that every algorithm designer has been faced with, but we have not yet seen any satisfactory answer to this question. We make some observations that lead us to a definition of the nonrandomness threshold, which helps us to better understand the nature of the problem. The Greedy Bit Set Algorithm is a tool that can and should be used by designers to determine realistic lower bounds on the initialization period for their algorithm.

We go on to show how the Greedy Bit Set Algorithm performs against a wide variety of new and old stream and block ciphers, and we find new record-breaking results for Trivium, Grain-128 and Grain v1 [DP08, HJM07, HJMM06, HJMM08]. We reveal weaknesses in Trivium reduced to 1026 out of 1152 initialization rounds in 2^{45} complexity, thereby significantly improving all previous efforts. By using a cluster we are able to improve this result even further to 1078 rounds at 2^{54} complexity. For Trivium we also present a new 806-round distinguisher of complexity 2^{44}. Both distinguishing and nonrandomness records are also set for Grain-128. We show nonrandomness in 256 (out of 256) initialization rounds, and present a 246-round distinguisher with complexity 2^{42}. For Grain v1 we show nonrandomness for 110 (out of 160) initialization rounds for a cost of only 2^9.

The chapter is organized as follows. In Section 5.2 we give an overview of the black box model attack scenario and explain the maximum degree monomial test. We also briefly describe the software tools developed. In Section 5.3 we present our Greedy Bit Set Algorithm. In Section 5.4, we comment on the importance of key weight and define the nonrandomness threshold. In

Sections 5.5 and 5.7 we present and summarize our findings for the various algorithms. Finally, some concluding remarks are given in Section 5.8.

As a frame of reference, this chapter takes Filiol [Fil02], Saarinen [Saa06] and Englund et al. [EJT07] as a starting point, and the most relevant previous work is due to Aumasson et al. [Aum+09, ADMS09] (see also Knudsen and Rijmen [KR07], Vielhaber [Vie07], Dinur and Shamir [DS09], and Fischer, Khazaei and Meier [FKM08]).

5.2 BACKGROUND

5.2.1 BOOLEAN FUNCTIONS

This short introduction to boolean functions is largely intended to follow the educational presentation in [Saa06], which provides additional details.

Let us begin by defining boolean functions. Although the definition is very simple, almost trivial, it provides a very general and powerful way to view some functions.

Definition 5.1 (Boolean function) The mapping

$$f : \{0,1\}^n \longrightarrow \{0,1\}^l,$$

where $n, l \geq 1$, is called an l-valued (or vectorial) n-ary boolean function. A single-valued (1-valued) boolean function may simply be referred to as a boolean function.

As an example, consider the boolean function

$$f(x_1, x_2, x_3) = 1 + x_1 + x_2 + x_1x_3 + x_1x_2x_3, \tag{5.1}$$

for which $x_1, x_2, x_3 \in \mathbb{Z}/2\mathbb{Z}$, written here in algebraic normal form.

Definition 5.2 (Algebraic normal form) The algebraic normal form (ANF) of a boolean function $f(x_1, \ldots, x_n)$ is the representation given by

$$\begin{aligned} f(x_1, \ldots, x_n) = & a_0 + \\ & a_1x_1 + a_2x_2 + \cdots + a_nx_n + \\ & a_{1,2}x_1x_2 + \cdots + a_{n-1,n}x_{n-1}x_n + \\ & \ldots + \\ & a_{1,2,\ldots,n}x_1x_2 \cdots x_n, \end{aligned}$$

where the coefficients are in the boolean domain $\{0,1\}$.

The reader may note that no squares or higher order terms are possible in the monomials, since $x^2 = x$ if $x \in \{0, 1\}$.

Every boolean function f has a unique ANF, and the ANF itself is very convenient for analysis of statistical properties and other interpretations, but we are not always given the ANF of a boolean function to start with. If the function f given in Equation (5.1) were to represent a 3-to-1-bit S-box, it would most likely have been given as a listing of all possible output values as in Table 5.1. Such a table is called a truth table, and it completely defines the

Table 5.1: Truth table for $f(x_1, x_2, x_3) = 1 + x_1 + x_2 + x_1x_3 + x_1x_2x_3$.

x_1, x_2, x_3	$f(x_1, x_2, x_3)$	x_1, x_2, x_3	$f(x_1, x_2, x_3)$
0, 0, 0	1	0, 0, 1	1
1, 0, 0	0	1, 0, 1	1
0, 1, 0	0	0, 1, 1	0
1, 1, 0	1	1, 1, 1	1

function. The truth table of an n-ary boolean function has 2^n entries, and each output value is either 0 or 1. It is therefore clear that there are precisely 2^{2^n} distinct n-ary boolean functions.

It is possible to efficiently compute the ANF of any n-ary boolean function from its truth table, and Algorithm 11 shows how this is done. The running time of Algorithm 11 is $n2^{n-1}$, if we count modular additions, and it uses 2^n bits of temporary memory for u and v. Note that this algorithm is practical only for values of n of moderate size.

We will be using one very specific property of boolean functions.

Theorem 5.1 (ANF coefficients in random boolean functions) Let X be a random variable that takes the value of the coefficient $a_{1,2,\ldots,n}$ of the maximum degree monomial (see Definition 5.2) in the ANF of a boolean function f that is drawn uniformly at random from the universe of all 2^{2^n} such functions. Then,

$$\Pr(X = 1) = \frac{1}{2}.$$

Proof. The truth table to ANF mapping is its own inverse, so the entropy of the truth table is preserved. A truth table with 2^n entries is mapped to a polynomial in ANF with 2^n monomial coefficients. ■

Theorem 5.1 provides a statistical perspective on the interpretation of properties of boolean functions, a perspective that will be very useful for us. Of

Algorithm 11 – ANF Computation from Truth Table

Input: Truth table t of size 2^n for f.
Output: ANF representation of f.

$u = (0, \ldots, 0)$; /* length 2^{n-1} */
$v = (0, \ldots, 0)$; /* length 2^{n-1} */
for $(i = 0; i < n; i{+}{+})$ {
  for $(j = 0; j < 2^{n-1}; j{+}{+})$ {
    $u[j] = t[2j]$;
    $v[j] = t[2j] \oplus t[2j+1]$;
  }
  $t = u \| v$;
}
return t;

Algorithm 12 – MDM Coefficient from Truth Table

Input: Truth table t of size 2^n for f.
Output: The coefficient $a_{1,2,\ldots,n}$ of the MDM in the ANF of f.

$b = 0$; /* single bit counter */
for $(i = 0; i < 2^n; i{+}{+})$ {
  b ^= $t[i]$; /* xor */
}
return b;

course, the maximum degree monomial is no special case, the theorem is equally valid for any other coefficient a in the ANF representation of f, as long as f is drawn uniformly at random.

We will soon see that the maximum degree monomial $x_1 x_2 \cdots x_n$ and its coefficient $a_{1,2,\ldots,n}$ in Definition 5.2 is of special interest to us, so in Algorithm 12 we explicitly describe how we can compute $a_{1,2,\ldots,n}$ from a truth table without recovering the entire ANF polynomial. It is quite clear that the running time of Algorithm 12 is 2^n if we, again, count modular additions.

5.2.2 BLACK BOX MODEL

It is time to review our take on distinguishers and nonrandomness detectors from Sections 2.2.3 and 2.2.4, so that we can bring in the boolean function perspective in a constructive way.

Distinguishers may be built for block ciphers, stream ciphers, MACs, and so on, so adopting a black-box view of the cryptographic primitive is instruc-

tive. While we have concentrated our efforts on stream ciphers, we have also included testing of block ciphers. However, the black box model makes us treat all primitives in a similar way.

Consider the set-up in Figure 5.1, which divides entities into three groups; public input parameters (left), hidden/secret input parameters (top) and output (right).

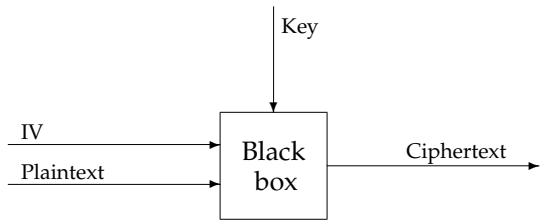


Figure 5.1: Black box view of a cipher.

A distinguisher attempts to determine if a given black box produces truly random output or not. Since no cryptographic primitive produces truly random output, a distinguisher can be thought of as a classifier. Given an output-producing black box, the distinguisher answers RANDOM or CIPHER, depending on its assertion. The distinguisher is said to be efficient if it significantly outperforms guessing, where the meaning of significantly depends on the application.

But what are the rules of the game, how can the distinguisher examine the functionality of the black box?

Consider first the hidden input parameters. The key is regarded as a hidden parameter, which in this context means that the distinguisher neither knows the key nor may alter its value. The key may be thought of as residing inside the black box, the internals of which is not available to us. The fixed key black box scenario is analogous to the case of hardware devices containing a hard-wired fixed key.

Now consider the public input parameters. Plaintext and IV are regarded as public parameters, so the distinguisher may invoke the black box several times with different plaintexts and IVs, but the key is kept fixed. Several different outputs may be produced in this way, and the distinguisher may use all of these outputs to influence its decision.

So, a distinguisher may modify all public input parameters, but not hidden ones. Nonrandomness detectors, on the other hand, may modify *all* input parameters, both public and hidden ones. While distinguishers are bound by practical constraints of the physical world, such as not having access to a hidden key, nonrandomness detectors boldly use a wider range of parameters.

Although this flexibility may seem awkwardly artificial at first, it is actually quite natural if one considers the purpose of a nonrandomness detector. Their primary merit is that they can do a better job of detecting nonrandomness. This is invaluable for the cryptographic community, as we can get earlier indications on weaknesses in specific algorithms. Distinguishers can show weaknesses in how IV and plaintext bits are handled, while nonrandomness detectors, in addition, can show weaknesses in how key bits are handled.

Nonrandomness detectors are less or even not useful in a real-world fixed key black box scenario, since they are related-key creatures by construction. But on a grand scale, we are wise to embrace them as finer precision tools for evaluating security aspects of cryptographic primitives. The reason: nonrandomness detectors provide a way of analysis and better understanding of random-like functionality.

Let S denote the set of bits that serve as input parameters[1] to a distinguisher or a nonrandomness detector. Explicitly summarizing the above, we have the following definitions.

Definition 5.3 (Distinguisher) A {RANDOM, CIPHER}-classifier whose input parameter bit set S includes only public input parameters is called a distinguisher.

Definition 5.4 (Nonrandomness detector) A nonrandomness detector is a {RANDOM, CIPHER}-classifier whose input parameter bit set S is not restricted. In particular, S may include *both* public and hidden input parameters.

Note that using a known or chosen key makes the {RANDOM, CIPHER}-classifier a nonrandomness detector, as we are then restricting the entropy of the key space, effectively allowing key bits in S. A related discussion can be found in [KR07].

5.2.3 THE MDM SIGNATURE

We now aim at unifying the boolean function view from Section 5.2.1 with the black box model from Section 5.2.2 in a constructive way. Algebraic techniques in general have recently been shown to be very powerful, and the maximum degree monomial (MDM) test stands out as a highly efficient randomness test. We have used this test in the following natural setting.

Consider a black box cipher that has been modified to produce output during its l initialization rounds. For simplicity, let us assume that the cipher is a

[1] In our experiments, we have not examined the effect of allowing plaintext bits in S, but this has the potential of working very well for block ciphers. However, this is not the case for (synchronous) stream ciphers, which are our main target here, as the keystream is independent of the plaintext.

stream cipher that outputs one keystream bit per clocking, and that keystream output is normally suppressed during its l initialization rounds. Although initialization and keystream generation can be very different in principle, this is a very common situation for stream ciphers as designers tend to go for reuse of structure in order to minimize the cipher footprint.

If we view the black box as a boolean vector function f, we are in effect defining

$$f : \{0,1\}^m \longrightarrow \{0,1\}^l.$$

Here, we have the input parameter space $B = \{0,1\}^m$ spanned by all key and IV bits $K = \{0,1\}^{m_k}$ and $V = \{0,1\}^{m_v}$, respectively, with $m = m_k + m_v$ so that $B = K \times V$. Note that the definition of f also implicitly defines l single-valued boolean functions $f_i, 1 \leq i \leq l$, one for each output bit. The function f fully describes the entire initialization procedure of the stream cipher. For every possible input, we can compute the corresponding initialization output.

If our computational abilities were unlimited, we would be able to calculate the l-bit block

$$\begin{aligned} \sigma &= \sigma_1 \| \sigma_2 \| \ldots \| \sigma_l \\ &= \bigoplus_{\mathbf{x} \in \{0,1\}^m} f(\mathbf{x}), \end{aligned} \tag{5.2}$$

which is the MDM signature of the entire stream cipher. Here, σ denotes the entire l-bit MDM signature, and the individual bits are denoted $\sigma_1, \ldots, \sigma_l$.

The i^{th} signature bit σ_i is the coefficient of the maximum degree monomial $x_1 x_2 \cdots x_m$ in the ANF of the single-valued boolean function f_i, since the summation is precisely as in Algorithm 12. That is, the i^{th} signature bit σ_i says something about the suppressed initialization output at the i^{th} clocking.

Theorem 5.1 tells us that an ideal cipher produces a random-looking MDM signature. This is because each f_i will be drawn uniformly at random from the universe of all 2^{2^m} boolean functions in the ideal case.

We desperately need to take care of one particularly disturbing detail before we continue. We have assumed unlimited computational abilities above, so we need to modify the above notion of the MDM signature into something that is much more practical. Instead of considering the entire input variable bit space $B = \{0,1\}^m$ at once, we can choose a rectilinear subspace $S = \{0,1\}^n$ of the bit space B, reducing the number of variables from m to n. By rectilinear subspace we mean to say that out of the m bits that make up the basis for B, we choose a subset of n bits and let S be the subspace generated by those n bits. For the remaining $m - n$ variables we choose a fixed variable assignment $\mathcal{A}$ and restrict our view of the function f to the subspace S in the obvious way.

Definition 5.5 (MDM signature $\sigma_{\mathcal{A},S}$) Let the l-valued boolean function

$$f : \{0,1\}^m \to \{0,1\}^l \tag{5.3}$$

represent the suppressed initialization output of a cryptographic primitive, one bit for each of its l initialization rounds. Given a rectilinear subspace $S = \{0,1\}^n$ over n free input parameters of the entire bit space $B = \{0,1\}^m$, and a fixed variable assignment $\mathcal{A}$ for the remaining $m - n$ variables, let

$$f_{\mathcal{A}} : \{0,1\}^n \to \{0,1\}^l \tag{5.4}$$

denote the function obtained from f by employing the variable assignment $\mathcal{A}$ in the obvious way. If the boolean function f is clear from the context, the l-bit MDM signature over the variable assignment $\mathcal{A}$ and bit set S may be denoted $\sigma_{\mathcal{A},S}$, or even simpler by σ if $\mathcal{A}$ and S are also explicitly or implicitly given. We have

$$\sigma_{\mathcal{A},S} = \bigoplus_{\mathbf{x} \in S} f_{\mathcal{A}}(\mathbf{x}). \tag{5.5}$$

While S really denotes a subspace of B, we will abuse this notation in the sequel by alternatively specifying S as a bit set, meaning the set of input bits that span the subspace in question. For rectilinear subspaces S, this usage does not cause any notational conflict.[2]

Example 5.1 (Trivium MDM signature) Trivium performs 1152 clockings during initialization and suppresses one output bit per clocking. Comparing to (5.3) in Definition 5.5, we therefore have

$$f : \{0,1\}^{160} \to \{0,1\}^{1152},$$

since Trivium takes 160 input bits, 80 for the key and 80 for the IV. The MDM signature for Trivium according to (5.5) in Definition 5.5 is an 1152-bit block.

Now choose a rectilinear subspace $S = \{v_0, v_3, \ldots, v_{78}\}$ consisting of every third IV bit, 27 bits in total. We let the variable assignment $\mathcal{A}$ for the remaining variables be the all-zero assignment. Comparing to (5.4) in Definition 5.5, we have

$$f_{\mathcal{A}} : \{0,1\}^{27} \to \{0,1\}^{1152}.$$

The 1152-bit MDM signature for Trivium over variable assignment $\mathcal{A}$ and bit set S is then

$$\sigma_{\mathcal{A},S} = \underbrace{\overbrace{000000\ldots000000}^{929 \text{ zeros}}10010\ldots}_{1152 \text{ bits}},$$

[2] It is possible to generalize Definition 5.5 to non-rectilinear subspaces. We will see an example of this in Section 5.5.1.1.

which is computed by xoring 2^{27} different suppressed initialization outputs, each being represented by an 1152-bit sequence.

The extremely long sequence of leading zeros in the MDM signature in Example 5.1 is very striking. We conclude that the sequence appears random-like close to where the first 1-bit appears, at round 930. We say that we have observed 929 zero rounds, and one interpretation of this is that 929 initialization rounds are not sufficient to properly mix the IV bits in S. Note that this in itself is a nonrandomness result (chosen key), motivated by Theorem 5.1.

To be a little more explicit, the IV bits $v_0, v_3, \ldots, v_{78}$ of Trivium are not jointly multiplied during any of the first 929 initialization rounds. This interpretation is the key to drawing conclusions on the mixing abilities of the primitive, and it gives us a glimpse into the internal workings of the cipher.

Running the MDM test is as simple as Definition 5.5 and Example 5.1 suggest. Choose a bit set S and a variable assignment $\mathcal{A}$, calculate the MDM signature and count the number of zero rounds. This will immediately produce a nonrandomness result. This is true no matter how we choose S or assign the remaining variables in $\mathcal{A}$. In particular, S may contain both hidden and public input parameters—both key and IV bits.

However, some caution is required for building a distinguisher. In this case it is necessary but not sufficient for the bit set S to contain public input parameters only. It is necessary not to involve key bits in S, as the MDM signature summation would then require related-key knowledge that is not available in a distinguishing situation. The insufficiency part is due to performance. In Example 5.1, where we use only IV bits in S, we would be tempted to conclude that the performance of the corresponding distinguisher is 929 rounds—that it can distinguish up to and including 929 rounds of Trivium initialization output from a truly random sequence. Unfortunately, this conclusion is gravely incorrect. The 929 zero round count was obtained by performing the MDM test against a *fixed* key, all zeros.

To assess the efficiency of the distinguisher, its performance needs to be sampled over *random* keys. This can be done by running several MDM tests over the same bit set S. The IV bits in $\mathcal{A}$ may be set to any fixed value, but the key bits must be chosen uniformly at random for each test. When we performed 16 MDM tests over random keys, setting the IV bits in $\mathcal{A}$ to zero, we got minimum, median, average and maximum zero round counts of 783, 794, 794.4 and 809, respectively. For Trivium, the MDM test works much better for keys and IVs of low weight, so we have a perceived performance gap that is substantial. We will say something more on the issue of key weight in Section 5.4.

Which zero round count you should report, to some extent depends on your view on key space. Consider the hardware analogy, in which the black

box contains a hidden but fixed key. Since we have no idea which key our distinguisher is facing, we need to report the minimum number of rounds if we need the distinguisher to work for (virtually) every key.

On the other hand, the round count itself can be more important in some cases. For example, cryptanalysts may want to report the maximum round count for a higher impact factor. They can do so, but there is a tradeoff. The distinguisher then only works for a fraction of the key space, but the round count is higher. Also, algorithm designers should focus on the maximum round count for issues such as determining a suitable duration for a cipher's initialization phase.

When it comes to time and storage complexities, nonrandomness results are in some sense cheaper to come by than distinguishers. Nonrandomness detection over a bit set $S = \{0,1\}^n$ has a time complexity of 2^n cipher initializations and requires $O(l)$ bits of storage.

For efficiency assessment of a distinguisher over a bit set $S = \{0,1\}^n$, evaluation over N randomly sampled keys has a time complexity of $N2^n$ cipher initializations and requires $O(l)$ bits of storage. The time required for running this distinguisher depends on whether we count the minimum or maximum number of zero rounds. If we count the minimum, so that the distinguisher works for the entire key space, the running time is simply 2^n initializations. Taking the maximum number of zero rounds, the black box distinguishing model is changed to assume availability of several black boxes, each one with a different but fixed key. In this scenario, we need to examine around N different black boxes before we find one that our distinguisher works for. The total running time for the distinguisher in this model is therefore about $N2^n$. Taking the maximum costs us a factor of N in time and requires the assumption that several black boxes are available for analysis. Higher values for N increase the confidence level of the zero round number estimate.

The MDM test seems to be highly efficient. In a cyptanalytical sense, it works very well in practice for some of the cryptographic algorithms. Also, the MDM test successively xors all intermediate output sequences to produce the final MDM signature, so it only requires a negligible amount of storage. Furthermore, one does not need to know anything at all about the internals of the algorithm that is being tested. The algorithm will quite politely but candidly reveal how susceptible any black box algorithm is to the MDM test.

5.2.4 RELATING TO HIGHER ORDER DIFFERENTIALS

There is a very natural correspondence between the MDM signature and higher order differentials, which were introduced by Lai [Lai94] and Knudsen [Knu95]. But before we say something about higher order differentials, let us very briefly introduce ordinary lowest-order differentials.

Differential cryptanalysis is a cryptanalytical technique that Biham and Shamir [BS91] introduced to the research community in the late 1980s. The technique was initially applied to block ciphers, but more recently it has been applied to stream ciphers and hash functions as well. The fundamental idea of differential cryptanalysis is to study the relationship between input and output differences, for which a chosen plaintext scenario is a natural setting.

Assume that a b-bit block cipher E is given. For two different plaintext blocks p and p', the input difference is $p \oplus p' = \Delta_p$. The difference between the two resulting ciphertexts $c = E(p)$ and $c' = E(p')$ is $\Delta_c = c \oplus c'$, so that we have

$$\Delta_c = E(p) \oplus E(p \oplus \Delta_p). \tag{5.6}$$

The pair (Δ_p, Δ_c) is called a differential. For a carefully selected differential (Δ_p, Δ_c), the probability that Equation (5.6) holds for a randomly chosen plaintext p should be considerably higher than $\frac{1}{2^b}$. Obviously, this property immediately implies a distinguisher, but key recovery methods are also possible.

While differential cryptanalysis considers the sum of two values, higher order differentials generalize this concept by addressing larger sums. Recently, Knellwolf and Meier [KM12] pointed out that ANF tests on boolean functions[3] can often be translated into high order differentials. This is particularly true for the MDM test.

Definition 5.6 (An n^{th} order derivative of a boolean function) Let $f : \{0,1\}^m \longrightarrow \{0,1\}$ be a single-valued boolean function, and let $S = \{0,1\}^n$ be a linear subspace of $B = \{0,1\}^m$ with $\dim S = n$. The n^{th} order derivative of f with respect to S is

$$\Delta_S f(\mathbf{y}) = \bigoplus_{\mathbf{x} \in S} f(\mathbf{y} \oplus \mathbf{x}). \tag{5.7}$$

In order to comply with our notation in Section 5.2.3, Definition 5.6 is expressed as if S is a rectilinear subspace of B, but this condition is not necessary. The subspace S may be any linear subspace of B.

Recall the l-valued boolean function $f_{\mathcal{A}}$ in (5.4) from Definition 5.5, and let $f_{i,\mathcal{A}}$, $1 \leq i \leq l$, denote the corresponding single-valued boolean function. Comparing Equation (5.7) to Equation (5.5), it is clear that the i^{th} MDM signature bit $\sigma_{i,\mathcal{A},S}$ is given by

$$\begin{aligned} \sigma_{i,\mathcal{A},S} &= \bigoplus_{\mathbf{x} \in S} f_{i,\mathcal{A}}(\mathbf{x}) \\ &= \Delta_S f_{i,\mathcal{A}}(\mathbf{0}). \end{aligned}$$

[3] For more details on monomial distribution tests, d-monomial test, the derived functions approach for key recovery, cube attacks and cube testers, see [KM12].

If we extend the Δ_S notation so that the summation in Equation (5.7) is also valid for l-valued boolean functions f, then we can write the entire MDM signature as

$$\begin{aligned}\sigma_{\mathcal{A},S} &= \bigoplus_{\mathbf{x}\in S} f_{\mathcal{A}}(\mathbf{x}) \\ &= \bigoplus_{\mathbf{x}\in S} f_{1,\mathcal{A}}(\mathbf{x}) \| \bigoplus_{\mathbf{x}\in S} f_{2,\mathcal{A}}(\mathbf{x}) \| \dots \| \bigoplus_{\mathbf{x}\in S} f_{l,\mathcal{A}}(\mathbf{x}) \\ &= \Delta_S f_{1,\mathcal{A}}(\mathbf{0}) \| \Delta_S f_{2,\mathcal{A}}(\mathbf{0}) \| \dots \| \Delta_S f_{l,\mathcal{A}}(\mathbf{0}) \\ &= \Delta_S f_{\mathcal{A}}(\mathbf{0}).\end{aligned}$$

5.2.5 TWO PRACTICAL DISTINGUISHING MODELS

For future reference, relating to the discussion in Section 5.2.3, we explicitly define the following two distinguishing models.

Definition 5.7 (Standard distinguishing model) In the standard distinguishing model, the attacker has access to only one output-producing black box, and she may only modify the public input parameters of the black box. A distinguisher in this model must be able to correctly {RANDOM, CIPHER}-classify the output of any given black box with high probability.

Definition 5.8 (Multiple choice distinguishing model) In the multiple choice distinguishing model, the attacker has access to any number N of output-producing black boxes, all of which produce either truly random or cipher output. The N black boxes are each chosen uniformly at random from the universe of all such black boxes. The attacker may only modify the public input parameters of each black box. A distinguisher in this model must be able to correctly {RANDOM, CIPHER}-classify the output of any such given set of N black boxes with high probability. The value N is chosen by the distinguisher.

The complexity of a distinguisher in the standard distinguishing model must be stated so that it can distinguish any given black box with high probability. In the multiple choice distinguishing model, the corresponding complexity may involve the parameter N, as we have seen in Section 5.2.3.

The standard distinguishing model should be familiar to most readers. It is a practical model, in the sense that the described scenario appears in real-life situations, and that the attacker may extract some nontrivial information from the black box.

The multiple choice distinguishing model is also practical in the very same sense, since the only additional requirement—availability of several black

boxes—is, obviously, also realistic. Furthermore, note that although each black box has a different key, there is no related-key ingredient in Definition 5.8. This is ensured by the condition that the N black boxes must each be chosen uniformly at random from the universe of all black boxes. The condition preserves practicality, effectively providing the perfect environment for a key space restricted distinguisher.

Of course, a distinguisher ceases to be practical once its total complexity reaches beyond the limit of our computational abilities, but this is a non-issue in our context.

5.2.6 BLACK BOX FRAMEWORK

As part of performing the research presented in this chapter, we put together a specialized cryptographic library that permits output of initialization data. The library was written in C and supports bitsliced implementations and threading to make good use of multiple cores. This is something that the MDM test benefits from since it is embarrassingly parallelizable. A unified interface makes it simple to author generic tests that can be used for all supported algorithms, and LaTeX-graphs of the results can be generated. This framework is an excellent tool for testing future generators. Interested researchers may find both ready-to-use executables and source code at [Sta10c]. The stream ciphers analyzed in this chapter are listed in Table 5.2.

Table 5.2: Stream ciphers subjected to GreedyBitSet.

Edon80 [GMK08]	Grain v1 [HJM07]
Grain-128 [HJMM06]	HC-128 [Wu08]
HC-256 [Wu04]	MICKEY v2 [BD08]
Rabbit [Boe+03]	Salsa20/12 [Ber08]
Sosemanuk [Ber+08]	Trivium [DP08]

Our primary target is stream ciphers, but we also apply the MDM test to block ciphers as a proof of concept. For block ciphers, we encrypt the plaintext block consisting of all zeros, and output all suppressed intermediate blocks. This gives us a reduced round output for the block ciphers. We use the zero round count measure for both stream and block ciphers. The block ciphers analyzed in this chapter are listed in Table 5.3.

5.2.7 VARIATIONS

Variations on the suggested MDM theme are, of course, abundant.

Table 5.3: Block ciphers subjected to GreedyBitSet.

AES-128 [CN01]	AES-256 [CN01]
Camellia [NC00]	CLEFIA [Shi+07]
DES [CN99]	HIGHT [Hon+06]
PRESENT [Bog+07]	RC5 [Riv95]
RC6 [RRSY98]	SEED [Lee+05]
SMS4 [DL08]	TEA [WNa]
XTEA [WNb]	

First of all, the suppressed output bits that we analyze can be defined in many different ways. For stream ciphers, we have simply taken the suppressed keystream as output, as explained in Section 5.2.6. However, depending on the internals of the cipher, higher biases may be attained if some nontrivial combination of keystream is analyzed instead. This approach requires abandonment of the black box view in favor of using additional information on the internal cipher structure.

Also, it is not necessary to aim at the keystream at all. It is also possible to examine the randomness of some part of the state by defining the output accordingly. This approach may provide insight on biases inside the state itself, which can be crucial in a cryptanalytical situation.

Furthermore, we have used MDM signature bit constantness (equal to zero) to define the very simple but natural zero round count measure, motivated by Theorem 5.1. We look for a consecutive string of zeros in the beginning of the MDM signature, but this measure ignores what happens after the first nonzero bit appears. In our simulations we have seen that the MDM signature will at times continue to be very sparse well after the first nonzero bit appears. This suggests that other bias tests are much more appropriate than the very strict and absolute zero round count. Other tests may be defined to detect weaker biases at much later rounds.

As stated in Section 5.2.6, when we apply the MDM test to block ciphers, we encrypt the plaintext block consisting of all zeros. All suppressed intermediate blocks are output to produce a reduced round output. However, using the zero round count measure in this context, quite frankly, makes much less sense than it does for stream ciphers. Consider the MDM signature part corresponding to two consecutive output blocks—two MDM blocks. As we consider only a consecutive initial string of zeros in the MDM signature, this translates to the quite harsh condition that the first MDM block must be en-

tirely canceled out before we have a look at the succeeding one. Repeating the idea presented above, the succeeding block may well be very sparse. In the block cipher scenario it would make more sense to try for depth rather than breadth. That is, it would make more sense to look at successive outputs of one specific bit position in the block, ignoring surrounding bits in favor of reaching as many rounds as possible.

As an alternative one may consider the block weight, aiming for a significant deviation from the random half-weight case. Unfortunately, there is a problem with this approach. Usually we do not have so many rounds to play with, so the statistical test quickly becomes meaningless for very short bit strings or, as in this case, when we generate too few samples for our statistical test. Of course, one could generate more samples by performing the tests several times using randomly chosen[4] IVs, or some similar strategy. It would be interesting to see how efficient we can make the tests for block ciphers, using these ideas and others.

For block ciphers, it would also make sense to include plaintext bits as part of the public parameters. The resulting distinguisher would in this case require several related plaintexts, which is a reasonable modification of the attack model.

One may also consider replacing the maximum degree monomial test with any other reasonable test of the ANF. For example, a more efficient test may be devised by counting the number of $n-1$ or $n-2$ degree monomials, or by considering any other more or less involved statistical property of boolean functions. Ultimately, the properties of the cipher decide which ANF tests, if any, that are efficient for proving nonrandom behavior.

Last but not least, we almost always perform the MDM signature summation in Definition 5.5 over a rectilinear subspace S. But nothing prevents us from first utilizing a linear transformation of the basis, so that we perform the original summation over a *tilted* cube instead. The generalized case provides additional degrees of freedom as the resulting search space is expanded quite drastically by this approach. The bit flipping test that we define in Section 5.5.1.1 can be viewed as a very simple application of this idea, but much more complex tests can be constructed.

5.3 THE GREEDY BIT SET ALGORITHM

The trick to obtaining good results with the MDM test is to find an efficient bit set S for summation, a bit set that produces many zero rounds. The well known greedy heuristic provides a very simple but highly successful algo-

[4]Randomly chosen over a subspace of the IV space that does not include bits used in the ANF test.

Algorithm 13 – GreedyBitSet

Input: Key k, IV v, permissible bit space P, desired bit set size n.
Output: Bit set S of size n.

```
S = ∅;
for (i = 0; i < n; i++) {
    S = AddBestBit(k, v, P, S);
}
return S;
```

Algorithm 14 – AddBestBit

Input: Key k, IV v, permissible bit space P, bit set S of size n.
Output: Bit set of size $n + 1$.

```
b = none; /* best bit so far */
max = −1; /* best zero round count so far */
for (all e ∈ P\S) {
    z = numZeroRounds(σ_{A_{S∪{e},k,v}, S∪{e}});
    if (z > max) {
        max = z;
        b = e;
    }
}
return S ∪ {b};
```

rithm that in some cases outperforms all methods we have seen so far. The algorithm is made explicit in Algorithm 13, and the subroutine AddBestBit is specified as Algorithm 14.

Note that the temporary variable S is used to successively build the bit set bit by bit, and that k and v are fixed input parameters. The values of the key and IV bits in k and v are used in the obvious way to form the variable assignment $\mathcal{A}_{S,k,v}$ for the variables that lie outside S.

The bit space parameter P determines if key and/or IV bits may be used to build the resulting bit set. That is, P can be either K, V or $B = K \times V$, depending on if we want to build a bit set with only key bits, only IV bits or both, respectively.

Further note that Algorithm 13 illustrates the straightforward greedy add best bit strategy for building the resulting bit set S. GreedyBitSet can, by avoiding unnecessary recalculations, easily be implemented to sport a run-

ning time of precisely[5]

$$1 + \sum_{0 \leq i < n} (m - i)2^i < m2^n$$

initializations[6] for building a bit set of size n, where m is the size of the permissible bit space P.

As a generalization, one may allow other bit set building strategies, or a non-empty starting bit set. In this somewhat generalized form we denote an instance of the algorithm

$$\text{GreedyBitSet}(\text{primitive, strategy, starting bit set, bit space, key, IV}).$$

For example, running the Greedy Bit Set Algorithm with the add best bit strategy (Add1) on Trivium starting with an empty bit set, allowing only IV bits in the bit set ($P = V$), using the all ones key and setting all remaining IV bits to zero may be denoted

$$\text{GreedyBitSet}(\text{Trivium, Add1}, \emptyset, V, \mathbf{1}, \mathbf{0}).$$

Instead of starting with an empty bit set one may begin by computing a small optimal bit set and go from there. For most of our results below we have used optimal bit sets of sizes typically around five or six.

An alternative bit set building strategy is denoted Add$N$, which operates by successively adding the N bits that together produce the highest zero round count when added to the existing bit set. These bit sets should heuristically be better than the ones produced using the Add1 strategy as local optima are more likely to be avoided. The performances of the Add1 and Add2 strategies for Grain-128 are compared in Figure 5.2, where the red and black curve represent the Add1 and Add2 strategies, respectively. GreedyBitSet with Add$N$ strategy can be implemented with a running time of precisely[7]

$$1 + \sum_{0 \leq i < k} \binom{m - iN}{N} 2^i < m^N 2^k$$

initializations for building a bit set of size kN.

We have standardized the graphs for uniform comparison between different algorithms. Given a bit set, the portion of leading zero rounds in the initialization rounds is denoted bit set efficiency.

For an ideal cipher, a bit set of size n produced by the Greedy Bit Set Algorithm will admit around $\log(m - n)$ zero rounds.

[5]There are m choices for the first bit, $m - 1$ choices for the second bit, and so on.

[6]Counting initializations for stream ciphers, and encryptions for block ciphers.

[7]There are $\binom{m}{N}$ choices for the first bit, $\binom{m-N}{N}$ choices for the second bit, and so on.

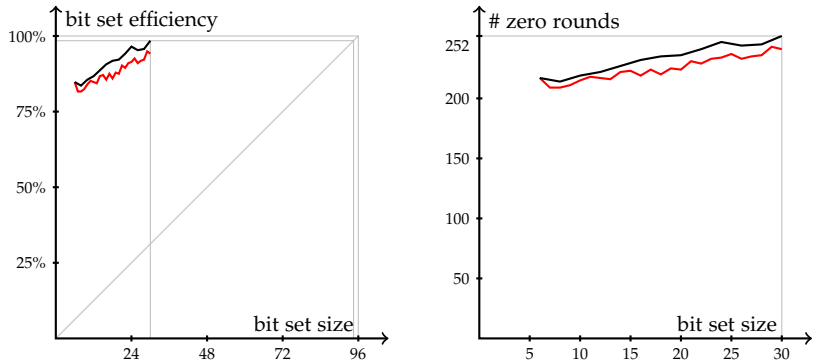


Figure 5.2: Add1 (red) vs. Add2 strategy (black) for Grain-128.

5.4 THE NONRANDOMNESS THRESHOLD

For some ciphers we have found that the result of the MDM test depends heavily on the weight of the key. A typical example of this is Trivium, for which the test seems to work best for the all zeros key and worst for the all ones key. Figure 5.3 shows the efficiency of the bit sets produced by Algorithm 13 for Trivium, starting with an empty set, using an all zero IV, for these two keys.

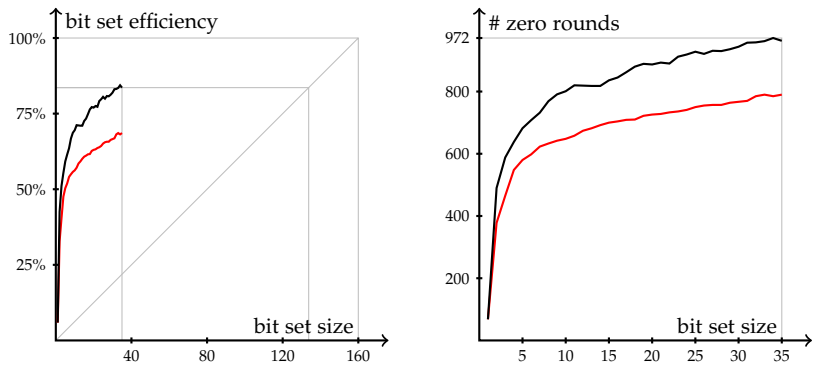


Figure 5.3: Key weight comparison for Trivium. The all zeros key (black) works better than the all ones key (red).

For Trivium it seems that the all zeros and all ones keys are extreme cases. All other keys we have tried end up producing a curve that lies between these two, and a curve produced by averaging over several randomly chosen keys

certainly falls between as well.

Consider an algorithm analyst that needs to determine a reasonable number for how many initialization rounds that are needed for balancing initialization time and security in Trivium. Using the graphs in Figure 5.3, the analyst can see that 1000 rounds will just barely withstand signs of improper mixing in this setting. At 972 rounds we start finding keys that allow us to prove that the bit mixing is inadequate. As we keep reducing the number of rounds, more and more keys show the same vulnerability. At 790 rounds, more or less all keys simultaneously chant *inadequate mixing* in four-part harmony. The algorithm designer should, of course, in this case decide on an initialization round count well above 972. How much more is debatable.

Recall that we use a bit set $S = \{0,1\}^n$, which is a subset of the entire bit space $B = \{0,1\}^m$. The highest round count value 972 obtained above should really be viewed as a lower bound of a theoretical threshold for bit sets of size n—the nonrandomness n-threshold. That is the nature of the limit we are exploring here, a threshold for the existence of proof of inadequate bit mixing. Testing a specific bit set of size n over a single key and IV, at the cost of 2^n initializations, provides a lower bound for this threshold. The true threshold value is conceptually obtained by repeating the MDM test several times taking the maximum over all possible keys, IVs and bit sets of size n for a total complexity of $\binom{m}{n}2^{m-n}$ such trials, or equivalently, $\binom{m}{n}2^m$ cipher initializations.

Definition 5.9 (Nonrandomness n-threshold) The maximum number of zero rounds attainable from an MDM signature according to

$$\max_{\substack{S \leq B,\ dim(S)=n \\ k \in K,\ v \in V}} \text{numZeroRounds}\left(\sigma_{\mathcal{A}_{k,v,S},S}\right)$$

is called the nonrandomness n-threshold. Here, B, K and V are the bit set space, key space and IV space, respectively, and S is iterated over all (rectilinear) subspaces of size n of B.

5.5 RESULTS

The algorithms are grouped according to susceptibility to the MDM test below, where particularly interesting algorithms are given room for elaboration. An algorithm is given a susceptibility rating high, significant, moderate or low according to its tendency to submit to the MDM test as the bit set size gets larger.

A direct comparison of our results to the previous and new best ones can be found in Tables 5.9 and 5.10 in Section 5.7.

5.5.1 HIGH SUSCEPTIBILITY

5.5.1.1 TEA AND THE BIT FLIP TEST

TEA is the top candidate. Starting with an empty bit set, we reach a full 100% zero round count after only two key bits have been added. It is the key bits that are the weak link, and this is a previously known deficiency in TEA (see [KSW97]).

The picture becomes quite different when one considers IV bit mixing. Allowing IV bits only results in a susceptibility that seems to be inherently low, as can be seen in Figure 5.4.

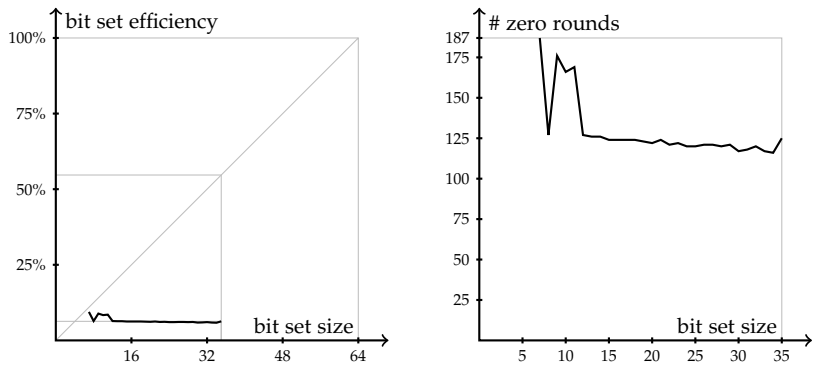


Figure 5.4: IV bit sets for TEA show decreasing efficiency.

It seems that the shortcomings in key bit mixing have been properly dealt with for XTEA, as the Greedy Bit Set Algorithm cannot show anything beyond a low susceptibility level for any bit type.

There is a lesson to be learned from the TEA flaw. It is revealed by simultaneously flipping two key bits, in which case the output does not change. We can devise an automated test for these simple symmetry faults. A Bit Flip Test can be defined by xoring the two output sequences produced before and after flipping all bits in a given bit set. Trying all bit sets of small size will catch design flaws such as the one in TEA. The Bit Flip Test is, in fact, an MDM test for a bit set of size 1 with a prior change of basis. Instead of summing over a perfect cube, we sum over a *tilted* cube that is the result of a linear transformation of the basis, as hinted in Section 5.2.7.

Two such two-bit configurations are known for TEA, and we have verified that no other ones exist. We have also verified by exhaustive search that none of the other algorithms we are considering here show any such bit flip weaknesses for small bit set sizes (five or so).

Table 5.4: The bit set of size 40 used to prove nonrandomness in the full Grain-128. IV bits only, bit order preserved, starting optimal 6-set $\{v_{34}, v_{59}, v_{63}, v_{64}, v_{67}, v_{69}\}$ at top left.

34	59	63	64	67	69	55	61	25	85	35	58	2	73	30
38	5	6	10	44	24	50	3	77	91	95	12	13	41	72
19	29	15	79	7	37	21	45	8	71					

The Bit Flip Test should really be part of every algorithm designer's toolbox. This test, and many others, should be used routinely to check for design errors or unexpected behavior.

5.5.1.2 GRAIN-128

For Grain-128, IV bits have a tendency to be more efficient than key bits and, as with Trivium, low weight keys work better than high weight keys. Running GreedyBitSet on the all zeros key with the Add2 strategy up to bit set size 40, IV bits only, we produced a nonrandomness detector for the full Grain-128 with its 256 initialization rounds. The successive development from the optimal 6-bit set to a bit set size of 40 is shown in Figure 5.5, and the (zero indexed) bit set is shown in Table 5.4. The order in which the bits have been added to the bit set has been preserved, and the first six bit indices form the optimal 6-set. The remaining IV bits were set to zero.

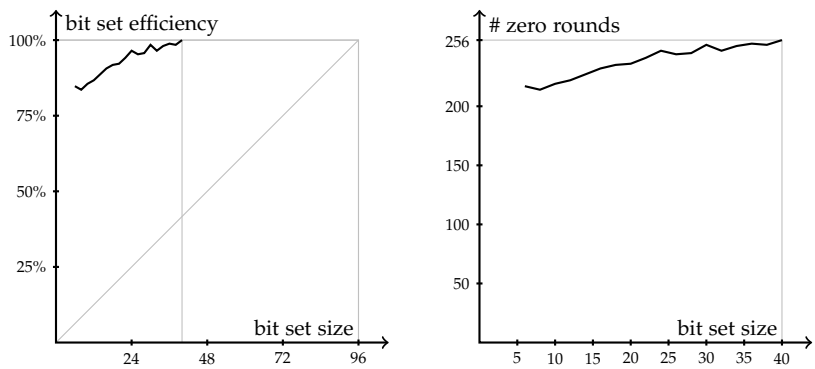


Figure 5.5: Insufficient IV bit mixing in the full 256-round Grain-128.

We now turn our attention from nonrandomness detectors to distinguishers. The best previous distinguishing result on Grain-128 were due to Knell-

wolf et al. [KMNP10] and Aumasson et al. [Aum+09]. In [KMNP10], they distinguish Grain-128 up to 215 rounds in the standard distinguishing model and recover parts of the key up to 213 rounds. In [Aum+09], taking the maximum number of rounds over 64 random key trials, they found a 237-round distinguisher for a bit set of size 40. To use our terminology, we have interpreted this as a multiple choice distinguisher as per Definition 5.8.

For our distinguishers, we ignore the two last bits of the bit set in Table 5.4, because they do not contribute positively to the zero round count. The resulting bit set of size 38 turns out to provide a 246-round distinguisher, measured by taking the maximum zero round count observed over 16 random key trials for a (practical) complexity of 2^{42} initializations. As explained in Section 5.2.5, this is a distinguisher in the multiple choice model.

The minimum number of zero rounds was 228, so the same bit set also produces a 228-round distinguisher in the standard distinguishing model according to Definition 5.7. This is a distinguisher with a (practical) complexity of 2^{38} initializations.

To summarize the case for Grain-128, we have found one nonrandomness detector showing that 256 (out of 256) rounds are insufficient for mixing the IV bits. This detector uses a bit set of size 40 and has a complexity of 2^{40}.

We also found a 246-round multiple choice model distinguisher with complexity 2^{42}. This distinguisher uses a bit set of size 38. Finally, we found a 228-round distinguisher with a complexity of 2^{38} initializations, in the standard distinguishing model.

A first key recovery result on Grain-128 was published after [Sta10b]. It is due to Dinur and Shamir [DS11], who show a full key recovery for Grain-128 reduced to 250 rounds with complexity of about 2^{100} (bit operations). The key recovery can also be adapted to the full 256 round Grain-128, but key recovery succeeds only for a fraction 2^{-10} of the key space in this case, which has complexity 2^{113}.

This attack was improved upon by Dinur et al. in [Din+11], where they claim a single-key attack that recovers the entire key of the full Grain-128 without, as before, restricting the key space. The time complexity of this attack is about 2^{90} initializations for 7.5% of the keys, but no complexity is specified for the remaining key space.

Also, if key recovery attacks in the related-key model are considered, then Lee et al. [LJSH08] provide full key recovery, which in their preferred version needs four keys and $2^{26.6}$ chosen IVs, and has a time complexity corresponding to about 2^{27} initializations.

5.5.1.3 TRIVIUM

There are several interesting observations to be made for Trivium, apart from the importance of key weight that we have already established in Section 5.4.

When employing GreedyBitSet to produce a greedy bit set, it is natural to aim at building a practical distinguisher. We therefore consider the case $P = V$ first, and Figure 5.6 shows the effects of the bit set development procedure when we allow only IV bits in the bit set.

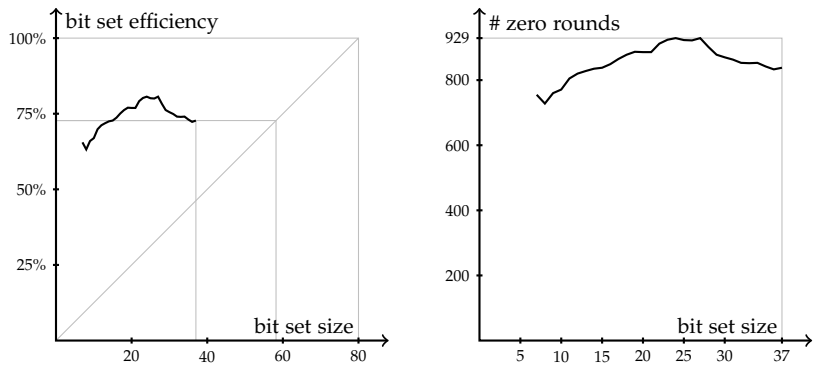


Figure 5.6: GreedyBitSet(Trivium, Add1, Opt7, V, $\mathbf{0}$, $\mathbf{0}$)

Building the bit set works magnificently up to size 27, but something seems to happen at that point. The bit set efficiency rapidly decreases as we keep expanding the bit set further. To see if this course of events is a mere coincidence, let us compare with the case $P = K$, which we have depicted in Figure 5.7.

The result is disappointing in that the bit set efficiency seems to reach some kind of optimum here as well. However, since the behavior is the same for both cases $P = V$ and $P = K$, the effect that we see is more likely to be caused by structural properties (it is).

We extend the analysis to the case $P = K \times V$, allowing both key and IV bits in the bit set. A typical run on GreedyBitSet is shown in Figure 5.8.

For $P = K \times V$, we do not get the same performance drop at bit set size 27 that we saw for both $P = K$ and $P = V$. Apart from a small dent in the curve at bit set size 27, there is virtually no performance drop at all. The bit set keeps getting better and better as it grows in size.

First of all, from the results above, let us conclude that key and IV bits are equally effective for Trivium. This conclusion comes as no surprise if one considers the initialization procedure. The key and IV are copied into one register each, but the initialization procedure makes no discernible dis-

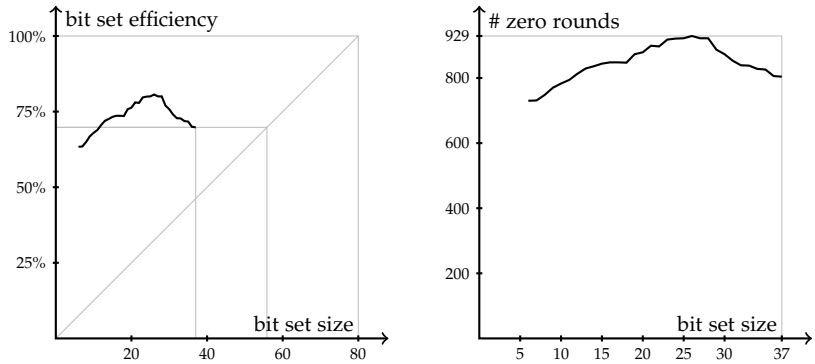


Figure 5.7: GreedyBitSet(Trivium, Add1, Opt6, $K, \mathbf{0}, \mathbf{0}$)

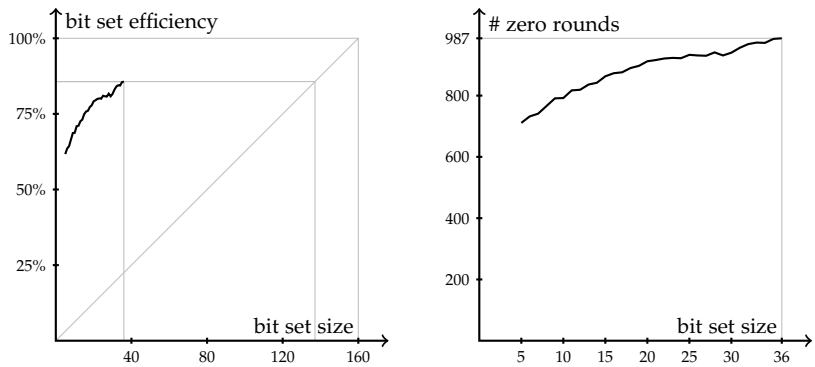


Figure 5.8: GreedyBitSet(Trivium, Add1, Opt5, $K \times V, \mathbf{0}, \mathbf{0}$)

tinction between key and IV bits, they are handled the same way. However, limiting the bit set to either key bits or IV bits seems to set a performance limit. Allowing both kinds of bits in our bit set will take us much further.

Why is this not a contradiction, and what is the magic behind the bit set of size 27?

When we examine the bit sets more carefully, it is evident that using *every third* bit for our bit set turns out to be the most efficient choice. This is due to the threefold structure of Trivium, and this is not a new observation (see [MB07]). It does not seem to matter much which third we choose, but once we have started to build up our set we do best if we stick to that implicit third. Trivium keys and IVs are 80 bits long, so we run out of bit space when we have reached a bit set size of 27 bits.

Do not underestimate the importance of these observations, because they reveal a fundamental insight. Using a black box scenario, we do not explicitly exploit the internal structure of the underlying primitive in our analysis. However, despite this, black box testing is fully capable of delivering structural information relating to the given cipher. In the examples we have seen, GreedyBitSet picks up on and exploits the internal structure of Trivium and delivers it in the shape of an efficient bit set.

The best nonrandomness detector ($P = K \times V$) that we have found using GreedyBitSet takes us 1026 out of 1152 rounds. This was for the zero key, which we noted before was heavily biased. The greedy strategy was to start from the optimal 5-set and to use the Add2-strategy up to bit set size 29, via the Add1-strategy up to bit set size 37, to finally just guessing the last few bits for a total bit set size of 45. The resulting bit set is listed in Table 5.5.

Table 5.5: Bit set used for the 1026-round Trivium nonrandomness detector (zero indexed).

Key bits	1	4	7	10	12	16	19	22	25	31	34	37
	40	43	46	49	52	55	58	61	64	70	73	76
IV bits	1	4	7	10	16	19	25	28	31	34	37	40
	43	46	49	52	55	58	64	67	70			

The every-third-structure mentioned above is evident in this bit set, and the natural follow-up question is how many zero rounds the full 54-bit set with 27 key and 27 IV bits would take us. We were able to find the answer to this question, but since the task of xoring 2^{54} initializations is rather computationally intensive, it cost us more than one million core hours of computation on a cluster to get there. In the end, we showed that we get 1078 out of 1152 zero rounds, which covers a whopping 93.5% of the entire initialization phase. This result was obtained for the (canonical) bit set listed in Table 5.6 This is our best nonrandomness result for Trivium, and to the best of our knowledge, this nonrandomness result is still world-leading.

Considering the bit set performance drops we saw in Figures 5.6 and 5.7, it is reasonable to assume that we will see the same effect once we try to go beyond this supposedly near-optimal 54-bit set. The subspace consisting of every third bit is then exhausted, and adding any other bit will ruin the pattern in the same way as before.

If we think about the process in terms of boolean function monomials, using a neighboring bit bridges the distance between other bits, because they may then be multiplied together at an earlier round.

Table 5.6: Bit set used for the 1078-round Trivium nonrandomness detector (zero indexed).

Key bits	0	3	6	9	12	15	18	21	24
	27	30	33	36	39	42	45	48	51
	54	57	60	63	66	69	72	75	78
IV bits	0	3	6	9	12	15	18	21	24
	27	30	33	36	39	42	45	48	51
	54	57	60	63	66	69	72	75	78

We also present practical 803 and 806-round distinguishers for Trivium. As noted before, one can use the internal structure of Trivium by using every third IV bit for the bit set. Unfortunately, we run out of bits after 27 of them have been added. We can, however, skip exploiting the threefold structure and, instead, just use the fact that multiplication is always performed between neighboring state variables. Using *every second* IV bit for the bit set, starting at k_0, will avoid fast initial term growth and take us 803 rounds over randomly selected keys. This was the minimum number of rounds obtained over 16 trials, so this is a distinguisher in the standard distinguishing model, and the resulting time complexity is therefore 2^{40} initializations. Taking the maximum number of zero rounds over 16 trials produces an 806-round distinguisher in the multiple choice model, and the time complexity of this distinguisher is 2^{44} initializations.

A noteworthy recent result is due to Knellwolf et al. [KMNP12]. They construct a practical distinguisher with a time complexity of about 2^{25} for a weak subclass of 2^{26} keys for Trivium reduced to 961 out of 1152 rounds. This can be translated into a multiple choice distinguisher with a total time complexity of about $2^{80-26+25} = 2^{79}$ initializations, which borders the brute-force time complexity.

5.5.2 SIGNIFICANT SUSCEPTIBILITY

Grain v1 seems to be highly susceptible if one considers the successive efficiencies of the small optimal IV bit sets that are obtained for the all-ones key, setting the remaining IV values to ones. These bit sets are optimal in terms of nonrandomness counting zero rounds, see Figure 5.9.

The slope of the curve looks promisingly steep, and we have reached a bit set efficiency of 65% (104 out of 160 zero rounds) after only eight iterations. We also started a search for the optimal IV bit set of size 9, which gave us a bit

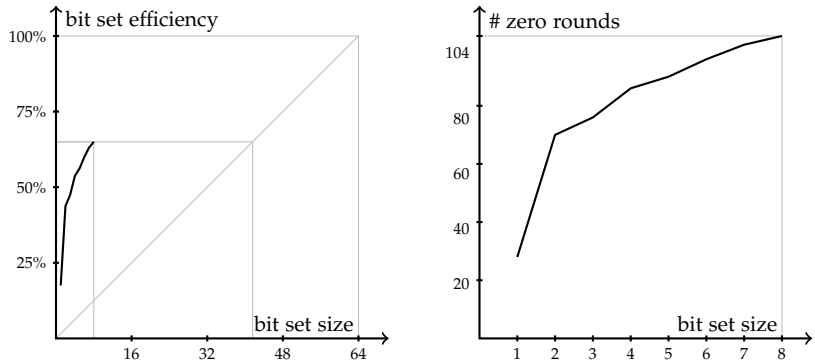


Figure 5.9: Optimal IV bit sets for Grain v1.

set producing 110 out of 160 zero rounds. This last bit set is not necessarily optimal, as the search did not cover the entire search space, but in terms of a nonrandomness result it sets a new record for Grain v1.

The eight optimal IV bit sets of sizes 1 through 8 are explicitly listed in Table 5.7, together with the good bit set of size 9. Here we also list the performances of each bit set in terms of the corresponding zero round count for nonrandomness (NR), a standard model distinguisher (SD) and a multiple choice distinguisher (MD). The SD and MD efficiencies were evaluated by taking the minimum and maximum zero round counts over 2^{10} randomly chosen keys, setting remaining IV bits to ones. Thus, for an IV bit set of size n, the time complexities of the corresponding SD and MD are 2^n and 2^{n+10}, respectively.

The natural next step is now to see if we can increase the zero round count of the optimal 8-bit set by expanding it using GreedyBitSet. However, the result of this greedy search is much less impressive than one might have wished for, see Figure 5.10.

Grain v1 is susceptible, as one can see from the direction of the curve in Figure 5.10, but the level of susceptibility seems to be limited as the extrapolated greedy curve will not hit the roof for any bit set of relevant size. Furthermore, there is a large drop in the first iteration of the greedy algorithm. We have seen this decline for other primitives as well, but the drop is usually not as dramatic as it is here. The greedy approach does not seem to work very well at first. It picks up the pace during the following rounds to show a positive slope, but it fails to recover from the initial recession.

One interpretation of this is that the search landscape defined by the MDM test is not very smooth for Grain v1—the zero round count values can vary

Table 5.7: Good and optimal IV bit sets for Grain v1. The bit sets of size 1 to 8 are optimal, while that of size 9 may not be.

size	bit set (zero indexed)									NR	MD	SD
1	12									28	28	9
2	0	5								70	64	34
3	9	14	50							76	75	46
4	9	19	34	57						86	79	56
5	8	37	49	56	57					90	80	56
6	5	7	27	29	43	45				96	72	53
7	12	17	18	46	47	53	59			101	82	63
8	4	16	20	22	37	43	44	60		104	82	67
9	11	18	21	22	28	45	55	56	60	110	88	68

considerably. The greedy approach is to perform a local search in the immediate neighborhood of the current best solution, and the optimal bit sets are local optima that are exceptionally good. It is possible that the optimal set of any size is extremely efficient compared to a randomly chosen bit set of that size. This suggests that there may be other ways of defining a close neighborhood for a search that is more efficient than the greedy approach, or perhaps employing non-rectilinear MDM summation in some way will be more efficient.

The plot in Figure 5.9 can be viewed in light of the nonrandomness n-threshold. The curve matches the best attainable bit set efficiency that can be obtained for each IV bit set size, which corresponds to restricting the maximum in Definition 5.9 to be taken over subspaces S of the permissible bit set subspace $P = V$, using the all-ones key and IV according to

$$\max_{S \leq P,\ dim(S)=n} \text{numZeroRounds}\left(\sigma_{\mathcal{A}_{\mathbf{1},\mathbf{1},S,S}}\right).$$

Of course, this curve sets an upper bound on the zero round count performance of *any* existing search algorithm, given the key and IV restrictions. The very same curve can also be seen as a lower bound for the true nonrandomness n-threshold, as lifting the restrictions in key and IV space may lead to increased zero round counts.

Our best nonrandomness result for Grain v1 is 104 zero rounds with the IV bit set of size 8 shown in Table 5.7, setting all key bits and the remaining

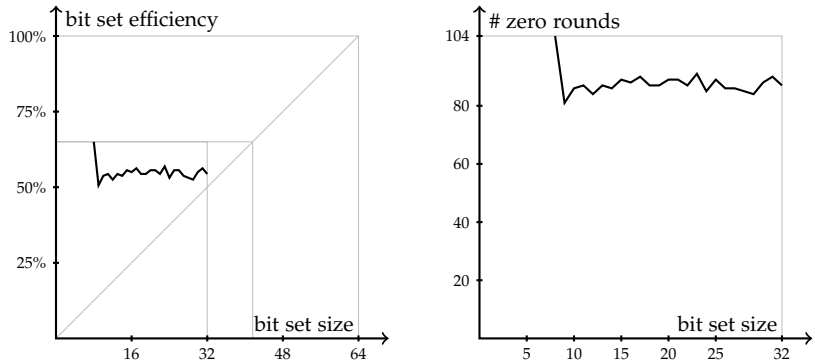


Figure 5.10: GreedyBitSet(Grain v1, Add1, Opt8, $V, \mathbf{1}, \mathbf{1}$)

IV bits to ones. As we have seen, this bit set produces a standard model distinguisher and a multiple choice distinguisher for 67 and 82 zero rounds, respectively.

Our best distinguisher is a 90-round greedy multiple choice distinguisher that was derived from a greedy IV bit set of size 35 by taking the maximum zero round count over 16 random keys for a complexity of 2^{39}. The IV bit set is given in Table 5.8.

Table 5.8: Greedy IV bit set of size 35 used by 90-round multiple choice distinguisher for Grain v1.

1	22	26	37	45	47	55	12	16	4	28	29
36	0	39	31	34	10	11	7	32	9	50	13
25	59	5	3	57	53	51	42	33	38	8	

The best distinguisher for Grain v1 is a 104-round distinguisher due to Knellwolf, Meier and Naya-Plasencia [KMNP10]. A side-effect of this distinguisher is that some key material is also recovered.

As for Grain-128, if we consider key recovery attacks for Grain v1 in the related-key model, Lee et al. [LJSH08] provide full key recovery using four keys and $2^{22.6}$ IVs in time corresponding to about $2^{22.9}$ initializations.

5.5.3 MODERATE OR LOW SUSCEPTIBILITY

AES, DES, CLEFIA and HIGHT all start at and stay within a bit set efficiency in the range 25-50%. These algorithms show only very slight or no sign of

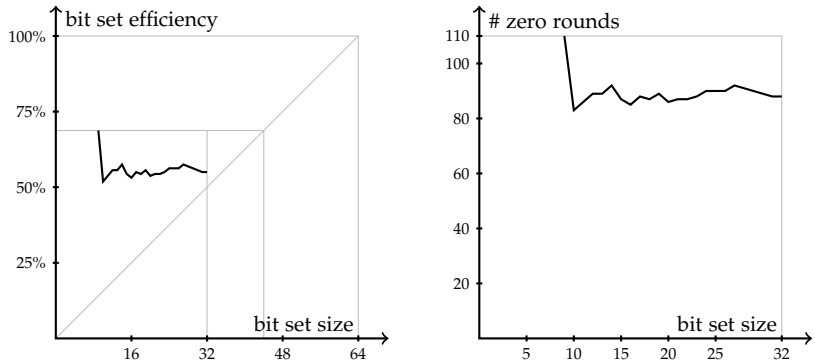


Figure 5.11: GreedyBitSet(Grain v1, Add1, $S, V, \mathbf{1}, \mathbf{1}$), where S is the IV bit set of size 9 given in Table 5.7.

budging as the bit set size increases.

The remaining ciphers have a bit set efficiency below 25%. Edon80 deviates from the norm by having a somewhat erratic curve, but it seems to stay within the 0-25% efficiency range. Sosemanuk does show a tendency to be affected by the MDM test, but all other algorithms seem to be more or less inherently non-susceptible.

It is interesting to see that the bit set efficiency for IV bits in RC5, and for IV bits in RC6 and key bits in XTEA to a lesser extent, show a *decreasing* tendency as the search progresses and bit set sizes increase. The curve for RC5 IV bits can be seen in Figure 5.12.

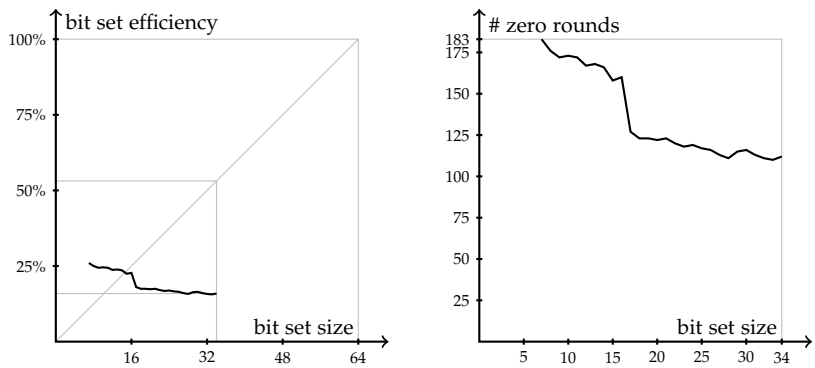


Figure 5.12: IV bit sets for RC5 show decreasing efficiency.

HC-128 and HC-256 set a record of sorts at the low end by showing no significant susceptibility while producing an extremely large amount of initial data.

The yet unmentioned and remaining algorithms; Camellia, MICKEY v2, PRESENT, Rabbit, Salsa20/12, SEED and SMS4, all seem to be inherently non-susceptible.

5.6 ON THE EXISTENCE OF WEAK BITS

Let us elaborate on the concept of *weak* bits, see [ADMS09, FKM08]. Weak bits are such that they significantly increase the efficiency (the number of zero rounds) of a bit set if they are added to it. The first question one might ask is: Do weak bits exist at all? The Greedy Bit Set Algorithm answers this question and reveals some deeper insight into the concept of weakness. Our algorithm successively builds larger bit sets by repeatedly adding the weakest remaining single bit (Add1 strategy). For Trivium, bits at every third bit position eagerly reappear among the top ranked bits again and again as the bit set size steadily increases. The bits at other (off-third) positions do not show up as top ranked at all. This zero round distribution regularity is clear evidence that Trivium has weak bits. Other algorithms show no sign of weak bits. This does not prove their non-existence in any way, but we surmise that any bit selection strategy for a truly perfect algorithm should not perform much better than random choice. For Grain-128, there are signs of bit weakness, but they are much less conclusive than for Trivium.

The existence of weak bits is algorithm dependent. Also, when we use GreedyBitSet we successively expand a bit set with the *currently* weakest bit. This means that the existence of weak bits does not only depend on the choice of test, but also on the current state of the test. As for drawing conclusions on the existence of globally weak bits, defining how to measure bit weakness is only the first step into a rather non-trivial enterprise.

One consequence of this is that one cannot prove any general performance guarantees for GreedyBitSet stating that we will obtain a sufficiently efficient bit set with some supposedly high probability. As we have seen, for Trivium we do, for RC5 we do not.

Also, more intelligent analysis of the zero round distribution over the remaining bit space could lead to better practical assessment measures for bit weakness that could be used to improve The Greedy Bit Set Algorithm.

5.7 RESULTS SUMMARY

We have shown how to find efficient bit sets in a systematic and deterministic way by using the Greedy Bit Set Algorithm. The record-breaking distinguishers and nonrandomness detectors derived from using the Greedy Bit Set Algorithm show that this algorithm has the capacity to outperform all other bit set selection schemes we have seen so far. Tables 5.9 and 5.10 compare the previous best results to the current ones for Trivium and Grain-128, respectively.

Table 5.9: Trivium record comparison. SD = standard distinguishing model, MD = multiple choice distinguishing model, NR = nonrandomness.

Type	Rounds	Time	Authors	Rounds	Time	Authors
SD	-	-	-	803	2^{40}	[Sta10b]
MD	806	2^{44}	[Sta10b]	961	2^{79}	[KMNP12]
NR	885	2^{27}	[ADMS09]	1078	2^{54}	[Sta10b]

Table 5.10: Grain-128 record comparison. SD = standard distinguishing model, MD = multiple choice distinguishing model, NR = nonrandomness, * = with full key recovery.

Type	Rounds	Time	Authors	Rounds	Time	Authors
SD	215	2^{25}	[KMNP10]	228	2^{38}	[Sta10b]
MD	246	2^{42}	[Sta10b]	256	2^{94}	[Din+11]*
NR	-	-	-	256	2^{40}	[Sta10b]

We presented a nonrandomness detector showing that Grain-128 with full 256-round initialization does not behave sufficiently random. This detector uses an IV bit set of size 40 and has a complexity of 2^{40}. We also presented a 246-round distinguisher over random keys with complexity 2^{42}.

For Trivium we found a greedy 1026-round nonrandomness detector with complexity 2^{45}. Using a cluster, we went on to find a nonrandomness detector for 1078 out of 1152 rounds with 2^{54} complexity. We also presented a 806-round distinguisher with 2^{44} complexity.

For Grain v1 we showed nonrandomness up to 110 rounds with complexity 2^9, and a 90-round distinguisher with complexity 2^{39}.

5.8 CONCLUDING REMARKS

With the exception of TEA, all block ciphers we have tested seem reasonably resistant to the maximum degree monomial test. Due to differences in how zero rounds are measured in stream and block ciphers, one should, however, not immediately draw the conclusion that block ciphers are safer than stream ciphers.

The Greedy Bit Set Algorithm can be examined with more elaborate strategy variants, bit selection schemes, randomness tests, cryptographic algorithms, allowing plaintext bits in the bit set, and so on. The most urgent and constructive goal, however, would be to explain why the MDM test fails miserably for some algorithms. What minimal set of properties is guaranteed to render the MDM test useless?

Automated cryptanalysis can be performed on many, if not most, cryptographic primitives. A toolbox of various tests, MDM-based and others, should be at the disposal of every algorithm designer. At the very least, as we have seen here, such a toolbox can be used to reveal unexpected design weaknesses and to give better estimations on the required number of initialization rounds.

List of Acronyms

ACISP Australasian Conference on Information Security and Privacy.

AES Advanced Encryption Standard.

ANF Algebraic normal form, see Definition 5.2.

APS All pairs sampling, see Section 4.3.1.

ARX A system that uses three operations; addition, xor and rotations.

AX A system that uses two operations; addition and xor.

DES Data Encryption Standard.

FCSR Feedback with carry shift register.

FSE Fast Software Encryption, an IACR conference.

GSM Global System for Mobile Communications, Groupe Spécial Mobile.

IACR International Association for Cryptologic Research.

IEEE Institute of Electrical and Electronics Engineers.

ISIT International Symposium on Information Theory, an IEEE conference.

IV Initialization vector.

LFSR Linear feedback shift register.

LPS Linear pairs sampling, see Section 4.3.2.

LSB Least significant bit.

MAC Message authentication code.

MDM Maximum degree monomial, see Section 5.2.1.

MSB Most significant bit.

NFSR Nonlinear feedback shift register.

OTP One-time pad.

RSA A public-key cryptosystem invented by Rivest, Shamir and Adleman.

RX A system that uses two operations; rotations and xor.

VS Vector sampling, see Section 4.3.3.

WS Weight sampling, see Section 4.3.4.

xor Exclusive or, a binary operation, see Table 1.1.

References

[AB05] F. Arnault and T. Berger, »F-FCSR: Design of a new class of stream ciphers«, *Fast Software Encryption 2005*, ed. by H. Gilbert and H. Handschuh, vol. 3557, Lecture Notes in Computer Science, Springer-Verlag, 2005, pp. 83–97, ISBN: 978-3-540-26541-2, DOI: 10.1007/11502760_6,

[ABL06] F. Arnault, T. Berger and C. Lauradoux, »Update on F-FCSR stream cipher«, 2006, available at: http://www.ecrypt.eu.org/stream/p3ciphers/ffcsr/ffcsr_p3.pdf, last accessed on May 13, 2013.

[ABLM07] F. Arnault, T. P. Berger, C. Lauradoux and M. Minier, »X-FCSR - A New Software Oriented Stream Cipher Based Upon FCSRs«, *Progress in Cryptology—INDOCRYPT 2007*, ed. by K. Srinathan, C. Pandu Rangan and M. Yung, vol. 4859, Lecture Notes in Computer Science, Springer-Verlag, 2007, pp. 341–350, ISBN: 978-3-540-77025-1, DOI: 10.1007/978-3-540-77026-8_26.

[ABP11] F. Arnault, T. Berger and B. Pousse, »A matrix approach for FCSR automata«, *Cryptography and Communications* 3 (2 2011), pp. 109–139, ISSN: 1936-2447, DOI: 10.1007/s12095-010-0041-z.

[ADMS09] J.-P. Aumasson, I. Dinur, W. Meier and A. Shamir, »Cube Testers and Key Recovery Attacks On Reduced-Round MD6 and Trivium«, *Fast Software Encryption 2009*, ed. by O. Dunkelman, vol. 5665, Lecture Notes in Computer Science, Springer-Verlag, 2009, pp. 1–22, ISBN: 978-3-642-03316-2, DOI: 10.1007/978-3-642-03317-9_1.

[Arn+09] F. Arnault, T. Berger, C. Lauradoux, M. Minier and B. Pousse, »A New Approach for F-FCSRs«, *Selected Areas in Cryptography—SAC 2009*, ed. by M. J. Jacobson Jr., V. Rijmen and R. Safavi-Naini, vol. 5867, Lecture Notes in Computer Science, Springer-Verlag, 2009, pp. 433–448, ISBN: 978-3-642-05443-3, DOI: 10.1007/978-3-642-05445-7_27.

[Aum+09] J.-P. Aumasson, I. Dinur, L. Henzen, W. Meier and A. Shamir, »Efficient FPGA Implementations of High-Dimensional Cube Testers on the Stream Cipher Grain-128«, *Workshop on Special Purpose Hardware for Attacking Cryptographic Systems (SHARCS'09)*, 2009, available at: http://eprint.iacr.org/2009/218, last accessed on May 13, 2013.

[BD08] S. Babbage and M. Dodd, »The MICKEY Stream Ciphers«, *New Stream Cipher Designs*, ed. by M. Robshaw and O. Billet, vol. 4986, Lecture Notes in Computer Science, Springer-Verlag, 2008, pp. 191–209, DOI: 10.1007/978-3-540-68351-3_15.

[Ber+08] C. Berbain, O. Billet, A. Canteaut, N. Courtois, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin and H. Sibert, »Sosemanuk, a Fast Software-Oriented Stream Cipher«, *New Stream Cipher Designs*, vol. 4986, Lecture Notes in Computer Science, Springer-Verlag, 2008, pp. 98–118, DOI: 10.1007/978-3-540-68351-3_9.

[Ber08] D. J. Bernstein, »The Salsa20 Family of Stream Ciphers«, *New Stream Cipher Designs*, vol. 4986, Lecture Notes in Computer Science, Springer-Verlag, 2008, pp. 84–97, DOI: 10.1007/978-3-540-68351-3_8.

[BJV04] T. Baignères, P. Junod and S. Vaudenay, »How far can we go beyond linear cryptanalysis?«, *Advances in Cryptology—ASIACRYPT 2004*, ed. by P. J. Lee, vol. 3329, Lecture Notes in Computer Science, Springer-Verlag, 2004, pp. 432–450, ISBN: 978-3-540-23975-8, DOI: 10.1007/978-3-540-30539-2_31.

[BMP09] T. Berger, M. Minier and B. Pousse, »Software Oriented Stream Ciphers Based upon FCSRs in Diversified Mode«, *Progress in Cryptology—INDOCRYPT 2009*, ed. by B. Roy and N. Sendrier, vol. 5922, Lecture Notes in Computer Science, Springer-Verlag, 2009, pp. 119–135, ISBN: 978-3-642-10627-9, DOI: 10.1007/978-3-642-10628-6_8.

[Boe+03] M. Boesgaard, M. Vesterager, T. Pedersen, J. Christiansen and O. Scavenius, »Rabbit: A New High-Performance Stream Cipher«, *Fast Software Encryption 2003*, vol. 2887, Lecture Notes in Computer Science, Springer-Verlag, 2003, pp. 307–329, DOI: 10.1007/978-3-540-39887-5_23.

[Bog+07] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin and C. Vikkelsoe, »PRESENT: An Ultra-Lightweight Block Cipher«, *Cryptographic Hardware and Embedded Systems—CHES 2007*, vol. 4727, Lecture Notes in Computer Science, Springer-Verlag, 2007, pp. 450–466, DOI: 10.1007/978-3-540-74735-2_31.

[BS91] E. Biham and A. Shamir, »Differential Cryptanalysis of DES-like Cryptosystems«, *Journal of Cryptology* 4.1 (1991), pp. 3–72, ISSN: 0933-2790, DOI: 10.1007/BF00630563.

[CLRS09] T. Cormen, C. Leiserson, R. Rivest and C. Stein, »Introduction to Algorithms, Third Edition«, MIT Press, 2009, ISBN: 0-262-03384-4, available at: http://mitpress.mit.edu/books/introduction-algorithms, last accessed on May 13, 2013.

[CN01] U.S. Department of Commerce and NIST, »Announcing the ADVANCED ENCRYPTION STANDARD (AES)«, November 2001, available at: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, last accessed on May 13, 2013.

[CN99] U.S. Department of Commerce and NIST, »DATA ENCRYPTION STANDARD (DES)«, October 1999, available at: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf, last accessed on May 13, 2013.

[CT91] T. Cover and J. A. Thomas, »Elements of Information Theory«, Wiley series in Telecommunication, Wiley, 1991, ISBN: 0-471-06259-6, available at: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0471241954.html, last accessed on May 13, 2013.

[DH76] W. Diffie and M. E. Hellman, »New Directions in Cryptography«, *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654, ISSN: 0018-9448, DOI: 10.1109/TIT.1976.1055638.

[Din+11] I. Dinur, T. Güneysu, C. Paar, A. Shamir and R. Zimmermann, »An Experimentally Verified Attack on Full Grain-128 Using Dedicated Reconfigurable Hardware«, *Advances in Cryptology—ASIACRYPT 2011*, ed. by D. H. Lee and X. Wang, vol. 7073, Lecture Notes in Computer Science, Springer-Verlag, 2011, pp. 327–

343, ISBN: 978-3-642-25384-3, DOI: 10.1007/978-3-642-25385-0_18.

[DL08] W. Diffie and G. Ledin, »SMS4 Encryption Algorithm for Wireless Networks«, 2008, available at: http://eprint.iacr.org/2008/329, last accessed on May 13, 2013.

[DP08] C. De Cannière and B. Preneel, »Trivium«, *New Stream Cipher Designs*, ed. by M. Robshaw and O. Billet, vol. 4986, Lecture Notes in Computer Science, Springer-Verlag, 2008, pp. 244–266, DOI: 10.1007/978-3-540-68351-3_18.

[DS09] I. Dinur and A. Shamir, »Cube Attacks on Tweakable Black Box Polynomials«, *Advances in Cryptology—EUROCRYPT 2009*, ed. by A. Joux, vol. 5479, Lecture Notes in Computer Science, Springer-Verlag, 2009, pp. 278–299, ISBN: 978-3-642-01000-2, DOI: 10.1007/978-3-642-01001-9_16.

[DS11] I. Dinur and A. Shamir, »Breaking Grain-128 with Dynamic Cube Attacks«, *Fast Software Encryption 2011*, ed. by A. Joux, vol. 6733, Lecture Notes in Computer Science, Springer-Verlag, 2011, pp. 167–187, ISBN: 978-3-642-21701-2, DOI: 10.1007/978-3-642-21702-9_10.

[Dun] O. Dunkelman, »Phorum5: ECRYPT forum, post 'A small observation on HC-128'«, available at: http://www.ecrypt.eu.org/stream/phorum/read.php?1,1143, last accessed on January 14, 2011.

[ECRa] ECRYPT, »D.SYM.3 – The eSTREAM Portfolio 2009 Annual Update, ICT-2007-216676«, ed. by C. Cid and M. Robshaw, available at: http://www.ecrypt.eu.org/stream/D.SYM.3-v1.1.pdf, last accessed on May 13, 2013.

[ECRb] ECRYPT, »eSTREAM: ECRYPT Stream Cipher Project, IST-2002-507932«, available at: http://www.ecrypt.eu.org/stream/, last accessed on May 13, 2013.

[EJT07] H. Englund, T. Johansson and M. S. Turan, »A Framework for Chosen IV Statistical Analysis of Stream Ciphers«, *Progress in Cryptology - INDOCRYPT 2007*, ed. by K. Srinathan, C. Pandu Rangan and M. Yung, vol. 4859, Lecture Notes in Computer Science, Springer-Verlag, 2007, pp. 268–281, ISBN: 978-3-540-77025-1, DOI: 10.1007/978-3-540-77026-8_20.

[Fil02] E. Filiol, »A New Statistical Testing for Symmetric Ciphers and Hash Functions«, *ICICS—2002*, ed. by R. Deng, F. Bao, J. Zhou and S. Qing, vol. 2513, Lecture Notes in Computer Science, Springer-Verlag, 2002, pp. 342–353, ISBN: 978-3-540-00164-5, DOI: 10.1007/3-540-36159-6_29.

[FKM08] S. Fischer, S. Khazaei and W. Meier, »Chosen IV Statistical Analysis for Key Recovery Attacks on Stream Ciphers«, *Progress in Cryptology—AFRICACRYPT 2008*, ed. by S. Vaudenay, vol. 5023, Lecture Notes in Computer Science, Springer-Verlag, 2008, pp. 236–245, ISBN: 978-3-540-68159-5, DOI: 10.1007/978-3-540-68164-9_16.

[GK02] M. Goresky and A. Klapper, »Fibonacci and Galois Representations of Feedback–With–Carry Shift Registers«, *IEEE Transactions on Information Theory* 48.11 (November 2002), pp. 2826–2836, ISSN: 0018-9448, DOI: 10.1109/TIT.2002.804048.

[GK12] M. Goresky and A. Klapper, »Algebraic Shift Register Sequences«, Cambridge books online, Cambridge University Press, 2012, ISBN: 9781107014992, available at: http://www.cs.uky.edu/~klapper/pdf/algebraic.pdf, last accessed on May 13, 2013.

[GK97] M. Goresky and A. Klapper, »Arithmetic Crosscorrelations of Feedback with Carry Shift Register Sequences«, *IEEE Trans. Info. Theory* 43 (1997), pp. 1342–1346, available at: http://www.math.ias.edu/~goresky/pdf/arith.jour.pdf, last accessed on May 13, 2013.

[GMK08] D. Gligoroski, S. Markovski and S. J. Knapskog, »The Stream Cipher Edon80«, *New Stream Cipher Designs*, ed. by Matthew Robshaw and Olivier Billet, vol. 4986, Lecture Notes in Computer Science, Springer-Verlag, 2008, pp. 152–169, ISBN: 978-3-540-68350-6, DOI: 10.1007/978-3-540-68351-3_12.

[HJ08] M. Hell and T. Johansson, »Breaking the F-FCSR-H Stream Cipher in Real Time«, *Advances in Cryptology—ASIACRYPT 2008*, ed. by J. Pieprzyk, vol. 5350, Lecture Notes in Computer Science, Springer-Verlag, 2008, pp. 557–569, ISBN: 978-3-540-89254-0, DOI: 10.1007/978-3-540-89255-7_34.

[HJ11] M. Hell and T. Johansson, »Breaking the Stream Ciphers F-FCSR-H and F-FCSR-16 in Real Time«, *Journal of Cryptology* 24 (3 2011), pp. 427–445, ISSN: 0933-2790, DOI: 10.1007/s00145-009-9053-2.

[HJB09] M. Hell, T. Johansson and L. Brynielsson, »An overview of distinguishing attacks on stream ciphers«, *Cryptography and Communications* 1.1 (1 2009), pp. 71–94, ISSN: 1936-2447, DOI: 10.1007/s12095-008-0006-7.

[HJM07] M. Hell, T. Johansson and W. Meier, »Grain - A stream cipher for constrained environments.«, *International Journal of Wireless and Mobile Computing, Special Issue on Security of Computer Network and Mobile Systems.* 2.1 (May 2007), pp. 86–93, ISSN: 1741-1084, DOI: 10.1504/IJWMC.2007.013798.

[HJMM06] M. Hell, T. Johansson, A. Maximov and W. Meier, »A Stream Cipher Proposal: Grain-128«, *International Symposium on Information Theory—ISIT 2006*, IEEE, July 2006, pp. 1614–1618, DOI: 10.1109/ISIT.2006.261549.

[HJMM08] M. Hell, T. Johansson, A. Maximov and W. Meier, »The Grain Family of Stream Ciphers«, *New Stream Cipher Designs*, ed. by M. Robshaw and O. Billet, vol. 4986, Lecture Notes in Computer Science, Springer-Verlag, 2008, pp. 179–190, ISBN: 978-3-540-68350-6, DOI: 10.1007/978-3-540-68351-3_14.

[Hon+06] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B.-S. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim and S. Chee, »HIGHT: A New Block Cipher Suitable for Low-Resource Device«, *Cryptographic Hardware and Embedded Systems—CHES 2006*, ed. by L. Goubin and M. Matsui, vol. 4249, Lecture Notes in Computer Science, Springer-Verlag, 2006, pp. 46–59, ISBN: 978-3-540-46559-1, DOI: 10.1007/11894063_4.

[HT93] R. V. Hogg and E. A. Tanis, »Probability and statistical inference«, 4th ed., MacMillan Publishing Co., 1993.

[JM05] E. Jaulmes and F. Muller, »Cryptanalysis of ECRYPT Candidates F-FCSR-8 and F-FCSR-H«, 2005, available at: http://www.ecrypt.eu.org/stream/papersdir/046.ps, last accessed on May 13, 2013.

[JM06] E. Jaulmes and F. Muller, »Cryptanalysis of the F-FCSR stream cipher family«, *Selected Areas in Cryptography—SAC 2005*, ed. by B. Preneel and S. Tavares, vol. 3897, Lecture Notes in Computer Science, Springer-Verlag, 2006, pp. 20–35, ISBN: 978-3-540-33108-7, DOI: 10.1007/11693383_2.

[Ker83] A. Kerckhoffs, »Kerckhoffs's Principle«, January 1883, available at: http://www.petitcolas.net/fabien/kerckhoffs/, last accessed on May 13, 2013.

[KG94] A. Klapper and M. Goresky, »2-adic Shift Registers«, *Fast Software Encryption'93*, ed. by R. J. Anderson, vol. 809, Lecture Notes in Computer Science, Springer-Verlag, 1994, pp. 174–178, ISBN: 978-3-540-58108-6, DOI: 10.1007/3-540-58108-1_21.

[KG97] A. Klapper and M. Goresky, »Feedback shift registers, 2-adic span, and combiners with memory«, *Journal of Cryptology* 10.2 (2 1997), pp. 111–147, ISSN: 0933-2790, DOI: 10.1007/s001459900024.

[Kla05] A. Klapper, »A Survey of Feedback with Carry Shift Registers«, *Sequences and Their Applications—SETA 2004*, ed. by T. Helleseth, D. Sarwate, H. Song and K. Yang, vol. 3486, Lecture Notes in Computer Science, Springer-Verlag, 2005, pp. 56–71, ISBN: 978-3-540-26084-4, DOI: 10.1007/11423461_3.

[KM12] S. Knellwolf and W. Meier, »High order differential attacks on stream ciphers«, *Cryptography and Communications* 4.3–4 (2012), pp. 203–215, ISSN: 1936-2447, DOI: 10.1007/s12095-012-0071-9.

[KMNP10] S. Knellwolf, W. Meier and M. Naya-Plasencia, »Conditional Differential Cryptanalysis of NLFSR-Based Cryptosystems«, *Advances in Cryptology—ASIACRYPT 2010*, ed. by M. Abe, vol. 6477, Lecture Notes in Computer Science, Springer-Verlag, 2010, pp. 130–145, ISBN: 978-3-642-17372-1, DOI: 10.1007/978-3-642-17373-8_8.

[KMNP12] S. Knellwolf, W. Meier and M. Naya-Plasencia, »Conditional Differential Cryptanalysis of Trivium and KATAN«, *Selected Areas in Cryptography—SAC 2011*, ed. by A. Miri and S. Vaudenay, vol. 7118, Lecture Notes in Computer Science, Springer-Verlag, 2012, pp. 200–212, ISBN: 978-3-642-28495-3, DOI: 10.1007/978-3-642-28496-0_12.

[KN10] D. Khovratovich and I. Nikolić, »Rotational Cryptanalysis of ARX«, *Fast Software Encryption 2010*, ed. by S. Hong and T. Iwata, vol. 6147, Lecture Notes in Computer Science, Springer-Verlag, 2010, pp. 333–346, ISBN: 978-3-642-13857-7, DOI: 10.1007/978-3-642-13858-4_19.

[Knu95] L. R. Knudsen, »Truncated and Higher Order Differentials«, *Fast Software Encryption'94*, ed. by B. Preneel, vol. 1008, Lecture Notes in Computer Science, Springer-Verlag, 1995, pp. 196–211, ISBN: 978-3-540-60590-4, DOI: 10.1007/3-540-60590-8_16.

[KR07] L. R. Knudsen and V. Rijmen, »Known-Key Distinguishers for Some Block Ciphers«, *Advances in Cryptology—ASIACRYPT 2007*, vol. 4833, Lecture Notes in Computer Science, Springer-Verlag, 2007, pp. 315–324, DOI: 10.1007/978-3-540-76900-2_19.

[KSW97] J. Kelsey, B. Schneier and D. Wagner, »Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA«, *Information and Communications Security, First International Conference—ICIS '97 Beijing, China, November 11–14, 1997 Proceedings*, vol. 1334, Lecture Notes in Computer Science, Springer-Verlag, 1997, pp. 233–246, available at: http://www.schneier.com/paper-relatedkey.pdf, last accessed on May 13, 2013.

[KY10] A. Kircanski and A. M. Youssef, »Differential Fault Analysis of HC-128«, *Progress in Cryptology—AFRICACRYPT 2010*, ed. by D. J. Bernstein and T. Lange, vol. 6055, Lecture Notes in Computer Science, Springer-Verlag, 2010, pp. 261–278, ISBN: 978-3-642-12677-2, DOI: 10.1007/978-3-642-12678-9_16.

[Lai94] X. Lai, »Higher Order Derivatives and Differential Cryptanalysis«, *Communications and Cryptography*, ed. by R. E. Blahut, D. J. Costello Jr., U. Maurer and T. Mittelholzer, vol. 276, The Springer International Series in Engineering and Computer Science, Springer-Verlag, 1994, pp. 227–233, ISBN: 978-1-4613-6159-6, DOI: 10.1007/978-1-4615-2694-0_23.

[Lee+05] H. J. Lee, S. J. Lee, J. H. Yoon, D. H. Cheon and J. I. Lee, »The SEED Encryption Algorithm«, 2005, available at: http://tools.ietf.org/html/rfc4269, last accessed on May 13, 2013.

[LJSH08] Y. Lee, K. Jeong, J. Sung and S. Hong, »Related-key chosen IV attacks on Grain-v1 and Grain-128«, *13th Australasian Conference on Information Security and Privacy, ACISP 2008*, ed. by Y. Mu, W. Susilo and J. Seberry, vol. 5107, Lecture Notes in Computer Science, Springer-Verlag, 2008, pp. 321–335, ISBN: 978-3-540-69971-2, DOI: 10.1007/978-3-540-70500-0_24.

[LN97] R. Lidl and H. Niederreiter, »Finite Fields«, 2nd, vol. 20, Encyclopedia of Mathematics and its Applications, Cambridge University Press, 1997.

[LQ09] Y. Liu and T. Qin, »The key and IV setup of the stream ciphers HC-256 and HC-128«, *International Conference on Networks Security, Wireless Communications and Trusted Computing*, vol. 2, April 2009, pp. 430–433, ISBN: 978-0-7695-3610-1, DOI: 10.1109/NSWCTC.2009.111.

[Mai+10] S. Maitra, G. Paul, S. Raizada, S. Sen and R. Sengupta, »Some observations on HC-128«, *Designs, Codes and Cryptography* (2010), pp. 1–15, ISSN: 0925-1022, DOI: 10.1007/s10623-010-9459-8.

[Mar03] G. Marsaglia, »Xorshift RNGs«, *Journal of Statistical Software* 8.14 (July 2003), pp. 1–6, ISSN: 1548-7660, available at: http://www.jstatsoft.org/v08/i14/paper, last accessed on May 13, 2013.

[Mat94] M. Matsui, »Linear Cryptanalysis Method for DES Cipher«, *Advances in Cryptology—EUROCRYPT'93*, ed. by T. Helleseth, vol. 765, Lecture Notes in Computer Science, Springer-Verlag, 1994, pp. 386–397, ISBN: 978-3-540-57600-6, DOI: 10.1007/3-540-48285-7_33.

[MB07] Alexander Maximov and Alex Biryukov, »Two Trivial Attacks on Trivium«, *Selected Areas in Cryptography—SAC 2007*, ed. by Carlisle Adams, Ali Miri and Michael Wiener, vol. 4876, Lecture Notes in Computer Science, Springer-Verlag, 2007, pp. 36–55, ISBN: 978-3-540-77359-7, DOI: 10.1007/978-3-540-77360-3_3,

[MOV97] A. Menezes, P. van Oorschot and S. Vanstone, »Handbook of Applied Cryptography«, CRC Press, 1997.

[NC00] NTT and Mitsubishi Electric Company, »Specification of Camellia - A 128-bit Block Cipher«, 2000, available at: http://info.isl.ntt.co.jp/crypt/eng/camellia/dl/01espec.pdf, last accessed on May 13, 2013.

[PL05] F. Panneton and P. Lécuyer, »On the xorshift random number generators«, *ACM Trans. Model. Comput. Simul.* 15 (4 October 2005), pp. 346–361, ISSN: 1049-3301, DOI: 10.1145/1113316.1113319.

[PMR10] G. Paul, S. Maitra and S. Raizada, »A Combinatorial Analysis of HC-128«, 2010, available at: http://eprint.iacr.org/2010/387, last accessed on May 13, 2013.

[PP05] S. Paul and B. Preneel, »Solving Systems of Differential Equations of Addition«, *Information Security and Privacy*, ed. by C. Boyd and J. González Nieto, vol. 3574, Lecture Notes in Computer Science, Springer-Verlag, 2005, pp. 275–303, ISBN: 978-3-540-26547-4, DOI: 10.1007/11506157_7.

[PR04] R. Pagh and F. F. Rodler, »Cuckoo hashing«, *Journal of Algorithms*, vol. 51, 2, Duluth, MN, USA: Academic Press, Inc., May 2004, pp. 122–144, DOI: 10.1016/j.jalgor.2003.12.002.

[Riv+09] R. L. Rivest, B. Agre, D. V. Bailey, C. Crutchfield, Y. Dodis, K. E. Fleming, A. Khan, J. Krishnamurthy, Y. Lin, L. Reyzin, E. Shen, J. Sukha, D. Sutherland, E. Tromer and Y. L. Yin, »The MD6 hash function: A proposal to NIST for SHA-3«, April 2009, available at: http://groups.csail.mit.edu/cis/md6/docs/2009-04-15-md6-report.pdf, last accessed on May 13, 2013.

[Riv11] R. L. Rivest, »The invertibility of the XOR of rotations of a binary word«, *International Journal of Computer Mathematics* 88.2 (January 2011), pp. 281–284, DOI: 10.1080/00207161003596708.

[Riv95] R. L. Rivest, »The RC5 encryption algorithm«, *Fast Software Encryption'95*, vol. 1008, Lecture Notes in Computer Science, Springer-Verlag, 1995, pp. 86–96, DOI: 10.1007/3-540-60590-8_7.

[RRSY98] R. L. Rivest, M. J. B. Robshaw, R. Sidney and Y. L. Yin, »The RC6 Block Cipher«, 1998, available at: http://people.csail.mit.edu/rivest/Rc6.pdf, last accessed on May 13, 2013.

[Saa06] M.-J. O. Saarinen, »Chosen-IV Statistical Attacks on eSTREAM Stream Ciphers«, 2006, available at: http://www.ecrypt.eu.org/stream/papersdir/2006/013.pdf, last accessed on May 13, 2013.

[SH12] P. Stankovski and M. Hell, »An Optimal Sampling Technique for Distinguishing Random S-boxes«, *ISIT12*, July 2012, pp. 846–850, DOI: 10.1109/ISIT.2012.6284680.

[Sha49] C.E. Shannon, »Communication Theory of Secrecy Systems«, *Bell System Technical Journal* 28 (1949), pp. 656–715.

[Shi+07] T. Shirai, K. Shibutani, T. Akishita, S. Moriai and T. Iwata, »The 128-Bit Blockcipher CLEFIA (Extended Abstract)«, *Fast Software Encryption 2007*, ed. by A. Biryukov, vol. 4593, Lecture Notes in Computer Science, Springer-Verlag, 2007, pp. 181–195, ISBN: 978-3-540-74617-1, DOI: 10.1007/978-3-540-74619-5_12.

[SHJ09] P. Stankovski, M. Hell and T. Johansson, »An Efficient State Recovery Attack on X-FCSR-256«, *Fast Software Encryption 2009*, ed. by O. Dunkelman, vol. 5665, Lecture Notes in Computer Science, Springer-Verlag, 2009, pp. 23–37, ISBN: 978-3-642-03316-2, DOI: 10.1007/978-3-642-03317-9_2.

[SHJ12a] P. Stankovski, M. Hell and T. Johansson, »An Efficient State Recovery Attack on the X-FCSR Family of Stream Ciphers«, *Journal of Cryptology, Online First^{TM}* (November 2012), pp. 1–22, ISSN: 0933-2790, DOI: 10.1007/s00145-012-9130-9.

[SHJ12b] P. Stankovski, M. Hell and T. Johansson, »Analysis of Xor-rotation with Application to an HC-128 Variant«, *ACISP12*, ed. by W. Susilo, Y. Mu and J. Seberry, vol. 7372, LNCS, Springer-Verlag, 2012, pp. 419–425, ISBN: 978-3-642-31447-6, DOI: 10.1007/978-3-642-31448-3_31.

[Sin00] S. Singh, »The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography«, 1st, New York, NY, USA: Anchor Books, Random House Inc., September 2000, ISBN: 0-385-49532-3.

[Slo11] N. J. A. Sloane, »The primes with primitive root 2, Sequence *A*001122 of The On-Line Encyclopedia of Integer Sequences«, 2011, available at: http://oeis.org/A001122, last accessed on May 13, 2013.

[SRHJ12] P. Stankovski, S. Ruj, M. Hell and T. Johansson, »Improved Distinguishers for HC-128«, *Designs, Codes and Cryptography* 63 (2 2012), pp. 225–240, ISSN: 0925-1022, DOI: 10.1007/s10623-011-9550-9.

[ST12] National Institute of Standards and Technology, »Secure Hash Standard«, March 2012, available at: http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf, last accessed on May 13, 2013.

[Sta10a] P. Stankovski, »Automated algebraic cryptanalysis«, *Proceedings of the ECRYPT Workshop on Tools for Cryptanalysis 2010*, ed. by F.-X. Standaert, ECRYPT II, June 2010, available at: http://www.ecrypt.eu.org/symlab/tools2010/tools2010-proceedings.pdf, last accessed on May 13, 2013.

[Sta10b] P. Stankovski, »Greedy Distinguishers and Nonrandomness Detectors«, *Progress in Cryptology—INDOCRYPT 2010*, ed. by G. Gong and K. C. Gupta, vol. 6498, Lecture Notes in Computer

Science, Springer-Verlag, 2010, pp. 210–226, DOI: 10.1007/978-3-642-17401-8_16.

[Sta10c] P. Stankovski, »Maximum Degree Monomial Toolkit with Source Code«, 2010, available at: http://www.eit.lth.se/staff/paul.stankovski/phdprojects, last accessed on May 13, 2013.

[Ste85] S. A. Stepanov, »On the number of irreducible polynomials in $\mathbb{F}_q[x]$ of special form«, *Russian Math. Surveys* 40 (4 1985), pp. 219–220, DOI: 10.1070/RM1985v040n04ABEH003656.

[Ste87] S. A. Stepanov, »The Number of Irreducible Polynomials of a Given Form over a Finite Field«, *Mathematical Notes* 41 (3 1987), pp. 165–169, ISSN: 0001-4346, DOI: 10.1007/BF01158241.

[TK09] S. S. Thomsen and L. R. Knudsen, »Cryptographic hash functions«, PhD thesis, Technical University of Denmark, February 2009, available at: http://orbit.dtu.dk/fedora/objects/orbit:82593/datastreams/file_5025771/content, last accessed on May 13, 2013.

[TQ09] T. Tian and W.-F. Qi, »Linearity properties of binary FCSR sequences«, *Designs, Codes and Cryptography* 52 (3 2009), pp. 249–262, ISSN: 0925-1022, DOI: 10.1007/s10623-009-9280-4.

[Vie07] M. Vielhaber, »Breaking ONE.FIVIUM by AIDA an Algebraic IV Differential Attack«, 2007, available at: http://eprint.iacr.org/2007/413/, last accessed on May 13, 2013.

[Wag02] D. Wagner, »A Generalized Birthday Problem«, *Advances in Cryptology—CRYPTO 2002*, ed. by M. Yung, vol. 2442, Lecture Notes in Computer Science, Springer-Verlag, 2002, pp. 288–304, ISBN: 978-3-540-44050-5, DOI: 10.1007/3-540-45708-9_19.

[WikiBP] Wikipedia, »Birthday problem — Wikipedia, The Free Encyclopedia«, available at: http://en.wikipedia.org/wiki/Birthday_problem, last accessed on May 13, 2013.

[WikiHis] Wikipedia, »History of cryptography — Wikipedia, The Free Encyclopedia«, available at: http://en.wikipedia.org/wiki/History_of_cryptography, last accessed on May 13, 2013.

[WikiLFSR] Wikipedia, »Linear feedback shift register — Wikipedia, The Free Encyclopedia«, available at: http://en.wikipedia.org/wiki/Linear_feedback_shift_register, last accessed on May 13, 2013.

[WNa] D. J. Wheeler and R. M. Needham, »TEA, a Tiny Encryption Algorithm«, available at: http://www.cix.co.uk/~klockstone/tea.pdf, last accessed on May 13, 2013.

[WNb] D. J. Wheeler and R. M. Needham, »TEA extensions«, available at: http://www.cix.co.uk/~klockstone/xtea.pdf, last accessed on May 13, 2013.

[WSJ13] H. Wang, P. Stankovski and T. Johansson, »A Generalized Birthday Approach for Efficiently Finding Linear Relations in ℓ-sequences«, Accepted for publication in Designs, Codes and Cryptography, March 2013.

[Wu] H. Wu, »Phorum5: ECRYPT forum, post 'Re: A small observation on HC-128'«, available at: http://www.ecrypt.eu.org/stream/phorum/read.php?1,1143, last accessed on July 3, 2011.

[Wu04] H. Wu, »A New Stream Cipher HC-256«, *Fast Software Encryption 2004*, vol. 3017, Lecture Notes in Computer Science, Springer-Verlag, 2004, pp. 226–244, available at: http://eprint.iacr.org/2004/092, last accessed on May 13, 2013.

[Wu08] H. Wu, »The Stream Cipher HC-128«, *New Stream Cipher Designs*, vol. 4986, Lecture Notes in Computer Science, Springer-Verlag, 2008, pp. 39–47, available at: http://www.ecrypt.eu.org/stream/p3ciphers/hc/hc128_p3.pdf, last accessed on May 13, 2013.

[xkcd] xkcd, »Security«, 2011, available at: http://xkcd.com/538/, last accessed on May 13, 2013.

[ZD11] L. Zhiqiang and P. Dingyi, »Constructing a Ternary FCSR with a Given Connection Integer«, 2011, available at: http://eprint.iacr.org/2011/358, last accessed on May 13, 2013.

LUND
UNIVERSITY

Series of licentiate and doctoral dissertations
Department of Electrical and Information Technology
ISSN 1654-790X
No. 50
ISBN 978-91-7473-526-0
http://www.eit.lth.se