On LFSR based Stream Ciphers

Analysis and Design

Patrik Ekdahl

LUND
UNIVERSITY

Ph.D. Thesis, November 21, 2003

© Patrik Ekdahl, 2003

Department of Information Technology
Lund University
P.O. Box 118
S-221 00 Lund, Sweden
http://www.it.lth.se/

Printed in Sweden by Ekop, Lund
October, 2003.

ISBN: 91-628-5868-8
ISRN: LUTEDX/TEIT-1025-SE

To my great relief.

Abstract

Stream ciphers are cryptographic primitives used to ensure privacy in digital communication. In this thesis we focus on stream ciphers built using Linear Feedback Shift Registers (LFSRs). Several different stream ciphers are analysed and new attacks are presented. In addition, two new stream ciphers are presented, both based on the same design.

The first attack is performed on SOBER-t16 and SOBER-t32. A new distinguishing attack is presented for simplified versions of the two ciphers, as well as for the complete version of SOBER-t16.

Next, the cipher A5/1, used in the GSM standard for mobile telephones, is analysed. The resulting attack is an initial state recovery attack which recovers the secret key using approximately 5 minutes of known keystream. The attack takes roughly 5 minutes to perform on today's standard PC.

Bluetooth is a well-known standard for wireless communication and the cipher responsible for the secrecy within that standard is called E_0. An initial state recovery algorithm on E_0 is presented, based on recently discovered correlations within the cipher. These new correlations are stronger than previously known. This attack, however, is only applicable to E_0 in a theoretical perspective, since the required length of the observed keystream is longer than allowed in the Bluetooth standard.

Following this, two distinguishing attacks are presented targeting clock controlled generators; the shrinking generator and the self-shrinking generator. The attack on the shrinking generator is based on a new observation that the majority bits of a block surrounding the tap positions in the LFSR output also fulfils the linear recurrence equation. The attack on the self-shrinking generator identifies two new classes of weak feedback polynomials. For the first class, both a distinguishing attack and an initial state recovery attack are presented. This distinguishing attack is remarkable in the sense that the required length of the observed keystream only grows linearly in the length of

the shift register. For the second class of weak feedback polynomials a distinguishing attack is given.

The final part of this thesis concerns the design of stream ciphers. Two new designs are presented, SNOW 1.0 and SNOW 2.0, the latter being an improvement on the former. These ciphers are designed to be very fast, especially in a software implementation.

Contents

Abstract **i**

Preface **vii**

1 Introduction **1**

1.1 Cryptology 2
1.2 Symmetric-key ciphers 7
1.3 Cryptanalysis 10
1.4 Methods of attack 11
1.5 Thesis outline 14

2 Introduction to stream ciphers **17**

2.1 Stream ciphers 18
2.2 Linear feedback shift registers 22
2.3 Boolean functions 25
2.4 S-Boxes 28
2.5 Some classic stream cipher designs 29
2.6 Generic attacks on stream ciphers 32
2.7 Security of a stream cipher 38
2.8 Summary 39

3 Cryptanalysis of SOBER-t16 and SOBER-t32 41
3.1 A description of SOBER-t16 and t32 42
3.2 Hypothesis testing—the short story 46
3.3 A distinguishing attack on SOBER-t16 without stuttering 49
3.4 A distinguishing attack on SOBER-t16 with stuttering 52
3.5 A distinguishing attack on SOBER-t32 without stuttering 55
3.6 Related results on SOBER-t32 with stuttering 58
3.7 Summary 59

4 Cryptanalysis of $A5/1$ 61
4.1 An overview of the security management in GSM 62
4.2 A description of $A5/1$ 67
4.3 Related work 69
4.4 A basic correlation attack 71
4.5 A refinement of the attack 73
4.6 Simulations of the attack 78
4.7 Summary 80

5 Cryptanalysis of E_0 83
5.1 Bluetooth—system overview 84
5.2 Security in Bluetooth 85
5.3 Attacking E_0 88
5.4 Previous and newer attacks on E_0 92
5.5 Summary 93

6 Cryptanalysis of the shrinking generator 95
6.1 Description of the attack 97
6.2 Analysis of the proposed attack 101
6.3 Simulation results 105
6.4 Related work 107
6.5 Summary 108
6.6 Technical details of the analysis 109

7 Cryptanalysis of the self-shrinking generator 117
7.1 Related work 118
7.2 A first class of weak polynomials 119
7.3 A general class of weak polynomials 123
7.4 Implementation aspects and simulation results 126
7.5 Summary 131

8 SNOW—a new family of stream ciphers **133**
8.1 A description of SNOW 1.0 134
8.2 Attacks on SNOW 1.0 140
8.3 A description of SNOW 2.0 141
8.4 Design differences—SNOW 1.0 vs. SNOW 2.0 145
8.5 Implementation aspects of SNOW 2.0 146
8.6 Attacks on SNOW 2.0 148
8.7 Summary 149

9 Concluding remarks **151**

Bibliography **153**

Preface

This thesis is the conclusion of my work as a Ph.D. student at the Department of Information Technology at Lund University. During these years, parts of the material have been presented at various conferences and published in journals.

Parts of the material have appeared in the following papers:

- P. Ekdahl and T. Johansson. SNOW—a new stream cipher. In *Proceedings of First open NESSIE Workshop*, Heverlee, Belgium, 2000.
- P. Ekdahl and T. Johansson. Distinguishing Attacks on SOBER-t16 and SOBER-t32. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption 2002*, volume 2365 of LNCS, pages 210–224. Springer-Verlag, 2002.
- P. Ekdahl and T. Johansson. A New Version of the Stream Cipher SNOW. in K. Nyberg and H. Heys, editors, *Selected Areas in Cryptography–SAC 2002*, volume 2595 of LNCS, pages 47–61. Springer-Verlag, 2002.
- P. Ekdahl and T. Johansson. Another attack on $A5/1$. *IEEE Transactions on Information Theory*, 49(1):284–289, January 2003.
- P. Ekdahl, W. Meier and T. Johansson. Predicting the Shrinking Generator with Fixed Connections. In E. Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of LNCS, pages 330–344. Springer-Verlag, 2003.

Parts of the material have also appeared as abstracts in the following conference proceedings:

- P. Ekdahl and T. Johansson. Some results on correlations in the Bluetooth stream cipher. In *Proceedings of 10th Joint Conference on Communication and Coding*, Obertauern, Austria, page 16, 2000.

- P. Ekdahl and T. Johansson. Another attack on $A5/1$. in *Proceedings of International Symposium on Information Theory*, page 160. IEEE, 2001.
- P. Ekdahl, T. Johansson and W. Meier. A Note on the Shelf-Shrinking Generator. In *Proceedings of International Symposium on Information Theory*, page 166. IEEE, 2003.

Acknowledgments

Numerous people have contributed and helped me during my years as a graduate student, without whom this thesis would never have been written. I would like to take the opportunity to express my gratitude towards some of them.

Firstly, my supervisor Professor Thomas Johansson. He has truly inspired me and helped me through some of the difficult times during these years. His patience and encouragement have been very important in addition to his thoroughness in research.

Mats Cedervall, who in my opinion is a brilliant lecturer, captured my curiosity for the subjects taught at the department and interested me in the possibility of doctoral studies. Mirek Novak, who diligently supervised my Master Thesis, also bears a great deal of responsibility for my chosen path.

The secretaries at the department, especially Laila Lembke and Lena Månsson, and the technical support team, never too busy for some relaxing small talk, provided the best possible services, in both administrative and practical matters.

All present and former colleagues at the department, especially those in the crypto group, those conferences would have been a lot less fun without you. I would also like to mention Jan Åberg and Nicklas Ekstrand, who helped me immensely during my first year at the department, and Stefan Höst, from whom I shamelessly stole many of the LaTeX tricks used writing this thesis.

Invaluable help was provided by Amy Boland, who carefully proof-read this thesis. Any errors that still remain are solely my own responsibility.

My parents have always encouraged me in my choices in life, helped me during my studies, and provided me a pole position in every new start. It is easy to grow up with that kind of support.

Finally, my lovely Lina, who supported me not only in the past months by proof-reading parts of this thesis, but has been a constant source of joy and comfort throughout my Ph.D. studies. Every moment with you is a "Kodak moment"—I love you.

Lund, 2003
Patrik Ekdahl

1 Introduction

To many people, cryptography is a strange, secret code used only by the military and secret agencies. Movies and books about spies during the cold war have painted a picture of cryptography being a science used only by people who shun the daylight. This view is strengthened by the fact that many people normally do not know or care about the inherent mechanism of their daily-used technical gadgets. Today, cryptology is an integral part of our lives whether we know it or not. The list of public applications is long, and many spring from the use of the internet. This global communication system has provided us not only with new buzz words such as "*24/7 shopping*", "*one-click-buy*" and "*JIT-services*" but also with new and very convenient ways of performing tasks.

If you use the internet to do bank transfers you use cryptography methods to both identify yourself to the bank, and let the bank identify itself to you, as well as keep your transactions private from other users of the internet. You most certainly want your bank transfers to be your own private business. This year, for the first time in Sweden, there was the possibility of doing the tax income declaration over the internet, and the security needs here are much the same as those for bank transfers. The old fashioned process of signing a paper can today be replaced by a digital signature. But the underlying idea that a signature (by hand or digital means) is difficult to disprove, is still the core of the procedure. Also the opposite applies, that forging a signature should be a problematic task, a property ensured by the digital signature. It is actually less likely that you will have your signature forged if you are using a well-constructed digital scheme, than if you sign a paper by hand.

Another commonly known area where cryptology is used by the broad public is in cellular phones. In the Global System for Mobile communications (GSM)[1], the

[1] GSM originally stood for Groupe Speciale Mobile, the name of the committee that began the standardisation procedure.

mobile phone system used in Europe, there is a built in cipher called $A5/1$ to ensure your conversation is private. However, as will be shown in this thesis, the GSM cipher is not particularly secure.

These are just a few examples of cryptology used in common public applications. Of course, military and government agencies still use strong cryptography to communicate, and with the exception of the last 30 years or so, the military needs for secrecy have been the primary force behind the developments in the area. Informal discussions with representatives of both the Swedish military intelligence service and foreign counterparts have indicated that the military have had an advantage in the design and analysis of both block and stream ciphers, but that this advantage has been gradually decreasing over the last 20-30 years. My impression is that public research has in many ways contributed to the military knowledge in the recent years.

There are two good books covering the development of cryptology up to its modern status, presented in a easy-to-read non-technical fashion. The first is *The Codebreakers* by Kahn [68], and the second is *The Code Book* by Singh [115]. Other books, also intended for the general public, covering more specific events in the history of cryptology and especially the breaking of the German navy cipher Enigma, are [108, 55, 60].

The rest of this chapter is devoted to a general introduction to the topic, which starts in Section 1.1 with a presentation of cryptology. In Section 1.2, symmetric-key ciphers are introduced. Some basic definitions for cryptanalysis and different types of attack are discussed in Section 1.3 and Section 1.4. Finally, the outline of the thesis is given in Section 1.5.

The non-technical reader is invited to read Chapters 1 and 2 for a general introduction to the topic of cryptology, and possibly also the introduction to Chapter 4 for an overview of the security management in the GSM mobile phone system.

1.1 Cryptology

Cryptology is the uniting name for a broad scientific field in which one studies the mathematical techniques of designing, analysing and attacking information security services. Cryptology consists of two subfields; *cryptography* and *cryptanalysis*. Cryptography is the field in which one studies techniques for providing security services and cryptanalysis is the field in which one studies techniques for defeating (attacking) the security services. We will adopt the viewpoint taken in [85] and distinguish four goals for cryptography services;

- *Confidentiality*. The goal is to ensure that the information is only available to authorised users. Synonymous terms are privacy and secrecy. One example of secrecy is message encryption.

- *Data integrity*. The goal is to ensure that only authorised users can alter the information without it being noticed. One familiar example is to write a letter and seal the envelope with sealing wax. Even though this example does not cover all aspects of digital data integrity, it gives an idea of the concepts.

- *Authentication*. The goal is to identify data origin or destination. Both parties in a communication sometimes need to ensure that the other is a legitimate user. This can also be applied to the data itself, as to ensure a specific date and time that the message was sent. A classic example from the movies is when two spies meet for the first time and exchange some predetermined phrases to establish identities; "*We last met in Prague, I believe*" - "*No, I think it was in Berlin, in 1944*". This is called a *challenge/response*. One party challenges the other, and if the other party knows the correct answer, the authentication is successful. For authentication using computers, the scheme is more complex than solely exchanging phrases, and the challenge is normally a computational challenge. If the responding party can correctly calculate an answer to the challenge based on the secret information, the authentication is successful.

- *Non-repudiation*. The goal is to ensure that someone cannot deny a previous commitment or action. As an example we can take a web shop, where customers can order products. The shop does not want the customers to be able to later deny an order or product, neither would the customers want the shop to be able to forge an order from the customer. The infrastructure needed for this information security service is expensive and generally involves a trusted third party. This is probably the reason why non-repudiation for web shops and customers is not widely implemented.

These issues are addressed in cryptography using different *cryptographical primitives*. A primitive is a fundamental tool or algorithm designed to solve a specific information security problem. To achieve the goals listed above, one usually needs to apply several different primitives, and employ them using specific protocols to ensure a secure system overall. In this context, a protocol is a set of rules for communicating. A protocol determines both the order of the messages, possible messages to send/receive, and how to interpret the contents of the messages. For example a poker game is strictly governed by a protocol where the dealer deals the cards clock-wise, then the bets start with the person sitting after the dealer. Typical messages to be communicated in a poker game are "check", "raise", "call", or "fold". Using these words and not altering them as in *"I wanna check out your cards with some more dough in there, dude"*, instead of the correct phrase *"I raise"*, makes the game unambiguous, commits the player to his or her actions, and prevents cheating.

The classic information security goal, to provide secrecy, is obtained by using a primitive called a *cipher*. Some examples of other primitives are *signing* primitives which should provide the same functionality as a handwritten signature, or *Message*

Authentication Codes (MACs) which provide means to ensure that a message has not been altered on the route to the receiver.

The cryptographical primitives can be divided into three categories; *unkeyed primitives*, *symmetric-key primitives*, and *asymmetric-key primitives*, where the latter are also known as *public-key primitives*.

The unkeyed primitives include, for example, *hash functions* which are also used in databases and search algorithms. These unkeyed primitives are mainly used as building blocks in the keyed primitives. In this category, there are also theoretical tools such as random sequences. Random sequences are seldom used per se, mainly because they are quite cumbersome to obtain. Various suggestions on how to achieve true randomness have been proposed over the years and include thermal noise generated by a resistor, background radiation from space and lava light behaviour [62]. All these suggestions are either very expensive or impractical to implement. Less expensive solutions have problems, for example, if you use the timing of a user's keyboard typing or the random movement of the mouse as a source of randomness, there might not be keyboard strokes available at the moment you need random data. If you have a web server, it is more likely that it is tucked away in a computer room with no keyboard or mouse connected to it at all. Apart from availability, there is also a problem of estimating the randomness of the source. How much randomness is there in keyboard timing, given, for example, a microphone recording the strokes?

More important is the theoretical use of random sequences as a mathematical description of something that is almost random, *pseudo-random*. We also use true randomness as a theoretical tool in evaluating the strength of other cryptographical primitives by comparison.

In the presentation of the keyed primitives, i.e. the symmetric-key and asymmetric-key primitives, the focus will be on the cipher primitive, as the topic of this thesis is stream ciphers.

The historical way of encrypting is to use a symmetric-key primitive. The word "symmetric" refers to the fact that the same key is used for both encryption and decryption. A more general definition of symmetric-key encryption is that it should be computationally easy to derive the decryption key from the encryption key and vice versa. Thus the two keys need not be exactly identical, but in most symmetric schemes they are. This implies that the sender and receiver must share the key, and that key must be transferred by some secure means, e.g. a courier, or the two parties communicating must know in advance that they will engage in a private conversation and set up a prior key exchange. Figure 1.1 shows two people, Alice and Bob, communicating using a symmetric cipher, where the adversary Eve, who is trying to listen in on the conversation, is eavesdropping on the insecure communication channel. The names of the parties: Alice, Bob, and Eve are standard designations within the cryptographic literature.

A classical example of a symmetric-key cipher is the *Vigenère* cipher. In this system both the message $\mathbf{m} = m_0 m_1 \ldots m_N$ and the ciphertext $\mathbf{c} = c_0 c_1 \ldots c_N$ are strings

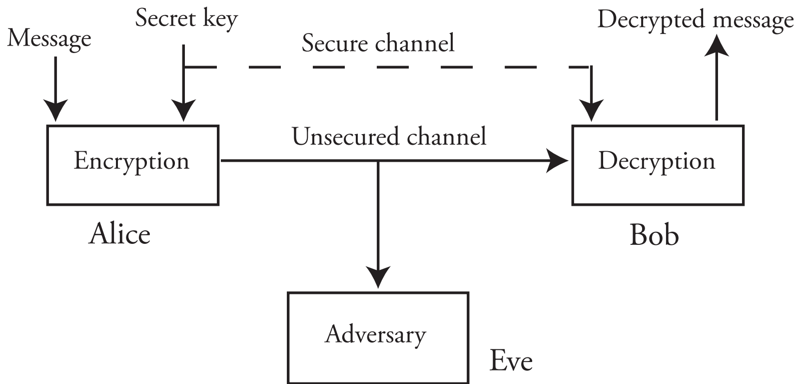


Figure 1.1: Two parties engaged in a symmetric cipher conversation with an adversary listening in on the unsecured channel.

of letters from the English alphabet $\{A, B, \ldots, Z\}$ of arbitrary but finite length $N+1$. The key $\mathbf{K} = k_0 k_1 \ldots k_{l-1}$ is also a string of letters from the English alphabet of length l. The message and the key are transformed to a sequence of integers by the transformation $A \leftrightarrow 0, B \leftrightarrow 1, \ldots, Z \leftrightarrow 25$. These sequences are denoted $\mathbf{m}'$ and $\mathbf{K}'$ for the message and the key respectively. The sequence of integers for the ciphertext is then computed as

$$c'_i = m'_i + k'_{i \bmod l} \bmod 26, \quad i = 0, 1, 2, \ldots, N. \tag{1.1}$$

The integer sequence $\mathbf{c}'$ is then transformed back to ordinary English letters by the same transformation as used previously.

Example 1.1: **Vigenère cipher**
Let the message be $\mathbf{m}$ ="TRUMPETPLAYER" and let the key be $\mathbf{K}$ ="PATRIK". The transformation of the message and the key to integer sequences yields $\mathbf{m}' = 19, 17, 20, 12, 15, 4, 19, 15, 11, 0, 24, 4, 17$ and $K' = 15, 0, 19, 17, 8, 10$. Applying the encryption rule (1.1) we get

	19	17	20	12	15	4	19	15	11	0	24	4	17
+	15	0	19	17	8	10	15	0	19	17	8	10	15
	8	17	13	3	23	14	8	15	4	17	6	14	6

where the addition is modulo 26, i.e. each sum that is larger or equal to 26 is subtracted by 26 to make the result in the range $0 \ldots 25$. Transforming the ciphertext back to English letters gives us a ciphertext $\mathbf{c}$ ="IRNDXOIPERGOG". □

The prerequisites for encryption were radically changed in 1976 when a celebrated paper by Diffie and Hellman [27] formed the basis of a new class of cryptographical primitives where the key need not be symmetric. Two years later, Rivest, Shamir and Adleman [97] presented their famous encryption scheme (the RSA encryption algorithm) based on the new principles; one key is used for encryption and another key is used for decryption. The important point here is that the decryption key should not be computable from the encryption key. This class of encryption schemes is called asymmetric-key primitives or public-key primitives. Public-key refers to the fact that the encryption key used for sending a message to a person A is public. Anyone can encrypt a message for A , but only A has the private key needed to decrypt the message. The crucial point here is that the encryption is not easily reversed. Even though the attacker knows how to encrypt, she cannot do the decryption. It can be imagined as if B wants to send A a private message, A first sends B an unlocked padlock. B puts the message in a box and locks the box with the padlock and sends it back to A . If the box with the message is intercepted on the way back to A it is still safe since nobody except A has the key which unlocks the padlock. Even if the attacker knows how to lock the padlock (encrypt), she cannot unlock it easily (decrypt). In technical terms it is said that the encryption process is hard to *invert* and in order to perform the inversion, you need to use a *trap-door*. The trap-door is the secret which makes it possible to decrypt. Figure 1.2 shows Bob sending Alice a private message using a public-key cipher.

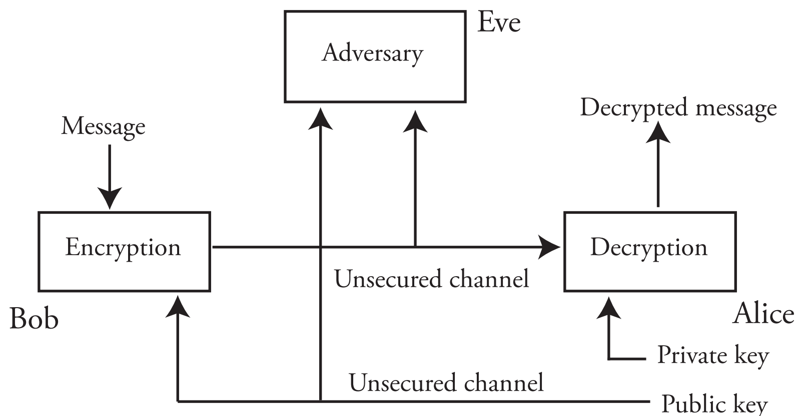


Figure 1.2: Two parties engaged in a public-key cipher conversation. The adversary can see both the ciphertext and the public key.

The public-key encryption schemes give more user freedom compared to the symmetric-key primitives. Basically anyone can send an encrypted message to you if you publish your public key on the web. But the asymmetric schemes also require more

cryptographical infrastructure, usually called the *Public Key Infrastructure* (PKI). In general we need a trusted third party to validate the public keys before use, such that the origin of the public key is authenticated. The public-key primitives are also much slower than the symmetric primitives and do not qualify for encryption of large bulks, e.g. hard disk encryption or high speed network encryption. Sometimes a combination of public-key and symmetric primitives is the solution. Two parties that do not have a prior agreement about a symmetric key can use public-key primitives to set up a secret symmetric key and then change to a symmetric bulk encryption primitive.

Interesting as the public-key primitives are, we must confine ourselves to the topic of this thesis and take an in-depth look at the symmetric-key ciphers only.

1.2 Symmetric-key ciphers

Let us first make a more precise definition of what we mean with a symmetric-key cipher. We will assume that the encryption and decryption keys are identical, which is the most common scenario. Firstly, some terminology will be introduced.

Definition 1.2: *An alphabet is a set of symbols.*

For example, an alphabet could be the set of binary digits $\{0, 1\}$, or the set of English letters $\{A, B, \ldots, Z\}$. Let

- $\mathcal{M}$ be the *message space* or the *plaintext space*. $\mathcal{M}$ consists of finite, nonempty strings of symbols from a suitable alphabet. An element of $\mathcal{M}$ is called a *plaintext message* or simply a *message*.
- $\mathcal{C}$ be the *ciphertext space*. Similarly, $\mathcal{C}$ consists of finite, nonempty strings of symbols from a suitable alphabet, not necessarily the same alphabet as for $\mathcal{M}$. An element of $\mathcal{C}$ is called a *ciphertext*.
- $\mathcal{K}$ be the *key space*. An element of $\mathcal{K}$ is called a *key*. $\mathcal{K}$ is the set of possible (valid) keys.

Additionally we denote by $E_k, k \in \mathcal{K}$, a set of bijective transformations from $\mathcal{M}$ to $\mathcal{C}$. For fixed k, E_k is called the *encryption function* or *encryption transformation*. Finally we denote by D_k a set of bijective transformations from $\mathcal{C}$ to $\mathcal{M}$. For fixed k, D_k is called the *decryption function* or the *decryption transformation*. The bijectivity implies that the cardinality of $\mathcal{M}$ and $\mathcal{C}$ must be equal. One could indeed consider an injective encryption function, but the decryption transformation would then first need to determine if the given ciphertext is valid. Thus we adopt these somewhat less cumbersome definitions.

Since we are only discussing symmetric-key ciphers we will drop "symmetric-key" and refer to them as "ciphers" throughout this thesis.

Definition 1.3: *A cipher is a set of encryption and decryption transformations* $\{E_k : k \in \mathcal{K}\}$ *and* $\{D_k : k \in \mathcal{K}\}$ *respectively. Additionally we require that* $m = D_k(E_k(m))$ *for all* $m \in \mathcal{M}$ *and* $k \in \mathcal{K}$*, i.e. for every valid key and every message, the decryption of an encrypted message gives the original message back.*

Note that the definitions above do not allow for infinitely long strings to be encrypted. That does not, however, imply any restrictions on the encryption, since each message must be finite in order to have an interpretation. Infinitely long strings cannot have an interpretation and thus there is no need to encrypt them.

If there is a very long message to be communicated, the delay would be too large if the encryption of the first letter of a message depended on the last letter of the message. Also during decryption, it is undesirable to have to wait for the entire (long) ciphertext to arrive through the channel before any decrypted plaintext can be produced. The solution is, of course, to divide the plaintext into smaller entities or blocks, where the processing of each block is only dependent on the current block being processed and possibly also on the previously processed blocks. To achieve this, there are two standard approaches for the design of the cipher, *a block cipher* or *a stream cipher*. To make the discussion more precise we assume that the messages are binary strings. This does not imply any restrictions on the types of messages that can be encrypted, since all kinds of information can be encoded into a binary string representation.

Block ciphers

A block cipher is a device that takes as input an L_{key} bit key and an n bit plaintext string and produces an n bit ciphertext string as output. The parameter n is called the *block size*. From the discussion prior to, and contained in Definition 1.3 we note that

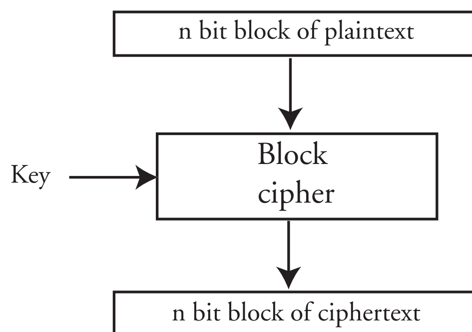


Figure 1.3: General structure of a block cipher.

a block cipher must define a permutation on the set of n bit binary strings. Thus, if the key is fixed and we encrypt all possible n bit messages, then we will get all possible

n bit strings as output. The Vigenère cipher defined in (1.1) is a block cipher with a block size equal to the key length.

Block ciphers are one of the most versatile cryptographical primitives. Apart from their use as ciphers for encryption, they serve as building blocks in numerous other primitives, for example authentication techniques and data integrity primitives.

Famous block ciphers include the DES,*(Data Encryption Standard)* [85] and the newly nominated standard AES *(Advanced Encryption Standard)* [25]. The DES was developed in the 1970's mainly by IBM, and has a 64 bit block size and a 56 bit key. Much effort has been put into finding weaknesses in DES but it has resisted surprisingly well over these last 30 years. One of the reasons for developing the new standard was that the key size of 56 bits is too small for the computing power of today's computers. A dedicated highly parallel machine could find the correct key by exhaustively trying all possible keys in a time frame of less than 6 hours. The cost of such a machine is rapidly decreasing and in 1997 would cost about US$100,000 to build. If we build a more advanced machine for about US$1,000,000, the time frame for breaking DES is estimated to be 35 minutes [121]. This is of course unacceptable for a cipher used in bank transfers and other highly secret environments. The solution taken has been to encrypt several times using a setup called *triple-DES*. This increases the key size but also reduces the encryption speed radically. The aim of the new standard AES was to deliver a new cipher with larger key size (128 and 256 bits), larger block size (128 bits) and with a much faster encryption/decryption time both in hardware and software.

One problem of a direct application of a block cipher as an encryption primitive is that patterns in the plaintext are visible for a fixed key. If two plaintext blocks are identical, the block cipher will encrypt them to identical ciphertexts. This reveals some information about the plaintext and is normally considered as a weakness for an encryption system. Therefore, a block cipher is almost never used in this so-called *Electronic Code Book* (ECB) mode of operation. There are other modes of operations that for example chain the output of one block encryption to the next. The ciphertext block of the first encryption is bitwise added to the plaintext block of the second before encryption. Then the second ciphertext block is added to the third plaintext block before encryption, et cetera. This solves the problem of patterns but introduces the severe problem of *error propagation*. If the ciphertext is distorted by noise during transmission, all subsequent plaintexts will be distorted during decryption due to this chaining. Other modes of operation are discussed in [85].

Stream ciphers

A stream cipher operates on individual characters in the underlying alphabet, with a time-varying function. Since the encryption is time-varying, we do not have the problem of patterns in the plaintext being encrypted to identical patterns in the ciphertext. We also have limited error propagation since each character of the plaintext

is encrypted individually. Compared to block ciphers, stream ciphers are also easier, smaller and cheaper to build in hardware. These properties make stream ciphers a suitable cryptographical solution in telecommunication systems or high speed network devices for attaining secrecy.

A thorough description of stream ciphers will be given in Chapter 2. Instead, we will now focus on the important area of cryptanalysis.

1.3 Cryptanalysis

As stated previously, cryptanalysis is the study of techniques for breaking a cryptographical primitive. When evaluating the strength of a cipher, we generally compare it to the generic attack of exhaustively searching over all possible keys to find the right one. This attack is called *exhaustive key search*. No practical cipher will be more secure than the time it takes to test all keys, so in a sense, this is the highest achievable strength of a cipher. There is however, one famous exception to this, the ultimately secure cipher; the *one-time pad* (OTP).

The OTP is the Vigenère cipher with a key length equal to the length of the message. Using binary strings we define it as follows. Let the message bits be denoted $\mathbf{m} = m_1 m_2 \ldots m_l$ and the key bits as $\mathbf{K} = k_1 k_2 \ldots k_l$. The ciphertext is computed by the exclusive or (xor[2]) of the message bits with the key bits

$$c_t = m_t \oplus k_t, \quad t = 1 \ldots l.$$

The important condition for the security of the OTP is that the key bits must be completely random and used only once.

Shannon [111] showed in his fundamental paper in 1949 that this encryption system is *unconditionally secure*. This property means that no matter how big or fast a computer the attacker has, she can never find out which plaintext was sent.

The unconditional security of the OTP comes from the fact that each plaintext is equally likely. Since the key bits are chosen randomly and independently of each other, each key is equally likely. If a key bit is 1, it means that we "flip" the corresponding message bit when computing the ciphertext. Thus, for each given ciphertext and each possible plaintext, we can find a key that decrypts the ciphertext into that particular plaintext, making each plaintext equally likely[3].

The obvious drawback of the OTP is that the key must be as long as the message that is to be encrypted[4]. This is in fact a necessary condition for any cipher claiming unconditional security. Since the key should be secret, one could argue that if we want to send an encrypted message, we first have to send a key that is as long as the message

[2] The xor operation is defined as $0 \oplus 0 = 0, 1 \oplus 0 = 1, 0 \oplus 1 = 1$ and $1 \oplus 1 = 0$.

[3] If the possible messages have a priori probabilities, the plaintexts (conditioned on the observed ciphertext) will not be equally likely after encryption, but sustain their a priori probabilities.

[4] To put it more precisely, the *entropy* of the key must be larger or equal to the entropy of the message.

through a secure channel. So, then we could have sent the message through the secure channel in the first place, instead of the key. The OTP is of course unpractical in such a setting, but could become useful if a secure channel is available during a small, limited time.

This describes the property of unconditional security. For practical ciphers the situation is not as clear as for the OTP. Often we talk about *computationally secure* ciphers instead. This means that, given the possibilities of today's computers and the predicted increase in performance of tomorrow's computers, the adversary cannot defeat the system. We also define the *computational security* of a cipher to be the computational effort required, by the best currently known attacks, to break the cipher.

Another way of describing the security of a cipher is to try to prove that breaking the cipher is equivalent to solving a difficult mathematical problem, like factoring integers or solving the discrete log problem. This way of arguing is called *provable security*, which is somewhat misleading since there is never a proof of the complexity of the underlying problem. The notion is that these problems have been studied by mathematicians for centuries and are probably very difficult to solve. Nevertheless, there is no guarantee that someone will not find a fast algorithm to factorise the integers, and then, for example, the RSA-type of public-key primitives are irreparably broken.

We see that apart from the unconditionally secure ciphers, there is really no proof of security anywhere. The best we can do is to evaluate the known attacks for a certain construction but we can never foresee future attacks and ideas for cryptanalysis. That is why cryptography is so unlike most other engineering areas. Usually the engineer can predict the worst case scenarios when constructing a building or an aeroplane and design accordingly. In cryptography the attacker is an adaptive, intelligent human being, who can learn from new scientific results and apply them to an older cipher. If that cipher is, for example, a world-wide standard in the financial market, then the worst case scenario may be a financial collapse.

1.4 Methods of attack

The first and most important rule for the designer of a cryptographic primitive is called *Kerckhoff's Principal*:[5]

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

Thus, the security of the design shall not rely on secret components of the primitive, since it is likely that the design will be leaked to an attacker. For example, the design of Enigma, one of the German ciphers used during World War II, was leaked when the allied forces captured German troops and naval ships. A more modern example is the cipher $A5/1$ in the GSM cell phone system. The design was initially

[5] He actually wrote *five* of them, but it is only the second which has become famous.

kept secret by the telecom industry but in 1999 the algorithm was reverse engineered by Briceno et al. [12]. The alleged structure of the cipher was later confirmed by the telecom companies. According to an invited talk by Babbage given at *Selected Areas in Cryptography* (SAC) in 2002, one of the main reasons for the decision to keep the algorithm secret was that the export regulations, at that time, prohibited the development of a full strength algorithm. In the new standard for the third generation mobile phones, the team of experts has published the proposed algorithms for public review [37].

Recalling the adversary Eve from Figure 1.1 on page 5, we will define some of the attacks that she might mount. From the figure, it may seem as though she can only listen in on the encrypted communication and not actively engage in the conversation, but in general we must allow her to be very active on the channel. She can: listen to, record, alter, resend, insert, or delete messages on the channel. From Kerckhoff's Principal we also assume that she knows everything concerning the cryptographical system and the protocols used between Alice and Bob. We now classify the methods of attack according to the amount of information the adversary Eve has obtained and the goal of the attack.

- **Ciphertext-only attack.**
  This is what most people think of when we talk about breaking a cipher. All the adversary sees is the ciphertext communicated between Alice and Bob. Trying to decrypt a message given only the ciphertext is the most difficult attack since the attacker has the least amount of information.

- **Known plaintext attack.**
  In this scenario the adversary knows both the plaintext and the corresponding ciphertext. It might seem slightly improbable that both the plaintext and the ciphertext are revealed, but there are many situations where this could happen. Many messages have predictable beginnings and endings, for example email messages. Maybe Alice is out of the office and has an auto-reply function on her email. When Eve sends her an email, she gets the auto-reply in plaintext. Later, when Bob sends Alice an email, the auto-replier sends the same message to him, but now encrypted. Alice could also send the same message encrypted to many recipients including Eve, who now has the plaintext and ciphertexts of the other recipients.

- **Chosen plaintext attack.**
  Here the adversary has access to an oracle, which can encrypt any given plaintext under the correct key. This is an even more powerful type of attack than the known plaintext attack, since the attacker can now choose plaintexts that are especially favourable for breaking the cipher. Sometimes we distinguish between an *online* and an *offline* attack. In the offline attack all plaintexts that should be encrypted are prepared in advance and handed to the oracle for encryption.

In the online attack the adversary can choose the next plaintext based on the previously received ciphertext from the oracle. This kind of attack is not as unrealistic as it would first seem. We could imagine an insider at a company who sneaks in during the lunch break and uses Alice's unlocked computer. The insider feeds the prepared plaintext to Alice's computer which acts as the oracle.

- **Chosen ciphertext attack.**
  This is similar to the chosen plaintext attack, but now the adversary has access to *two* oracles instead, one that encrypts any given plaintext and one that decrypts any given ciphertext except the ones the attack is trying to break. This attack is naturally more powerful than all the previous attacks, since the adversary has more freedom.

The primary goal of the attacks described above is to recover the plaintexts of a set of given ciphertexts or to recover the secret key. Another type of attack which is not as powerful, is the distinguishing attack. In this attack the adversary tries to detect a difference between the actual cipher and the ideal cipher. In the next chapter we will introduce the *additive* stream ciphers, and argue that the OTP is the ideal cipher for this class.

A distinguishing attack on an additive stream cipher can be used to verify or falsify that a specific message was encrypted. This is very relevant if the number of possible messages is limited and the attacker has the power to test them all. The attacker cannot use the retrieved information in the future to decrypt other messages, only to decide if a guessed plaintext is correct.

Grasping the numbers

In cryptology, extremely large numbers are very commonly used when stating results such as, for example, the required number of observed bits for an attack. Likewise, extremely small numbers are used when computing probabilities of, for example, correctly guessed bits in the cipher. Sometimes it can be hard to grasp the magnitude of these values and put them in a physical context. To aid the reader, different physical quantities and measures are presented in Table 1.1. Most values are taken from similar tables found in *Applied Cryptology* by Schneier [107], and *Information Theory, Inference, and Learning Algorithms* by MacKay [76].

Base 2	Base 10	Physical quantity
2^{8192}	10^{2466}	Number of distinct 1-kilobyte files.
2^{1000}	10^{301}	Number of binary strings of length 1000.
2^{266}	10^{80}	Number of electrons in the universe.
2^{190}	10^{57}	Number of electrons in the solar system.
2^{171}	$10^{51.5}$	Number of electrons in the earth.
2^{58}	$10^{17.5}$	Age of the universe in seconds.
$2^{36.5}$	10^{11}	Number of neurons in the human brain.
$2^{36.5}$	10^{11}	Number of bits stored on a DVD.
$2^{32.5}$	$10^{9.8}$	Number of bits in the human genome.
$2^{32.6}$	$10^{9.8}$	Population of earth.
$2^{23.3}$	10^{7}	Number of bits in the *E. Coli* genome, or in a floppy disk.
2^{20}	10^{6}	1,048,576.
2^{10}	10^{3}	1,024.
2^{0}	10^{0}	1.
$2^{-22.7}$	$10^{-6.8}$	Probability of winning the top prize in the Swedish *Lotto* lottery.
2^{-25}	$10^{-7.5}$	Probability of error in transmission of coding DNA, per nucleotide, per generation.
2^{-33}	$10^{-9.9}$	Probability of being killed by lightning, per day (U.S. statistics).
2^{-60}	10^{-18}	Probability of an undetected error in a hard disk drive, after error correction.

Table 1.1: Large and small numbers. Conversion between base 2 and base 10 is approximate.

1.5 Thesis outline

This thesis consists of three main parts, all concerning stream ciphers built using linear feedback shift registers. The first part, including Chapter 1 and Chapter 2 is a general introduction to cryptology and in particular stream ciphers. Chapters 4 through 7 are concerned with cryptanalysis of LFSR based stream ciphers. The final part, Chapter 8, presents a new design, the SNOW family of stream ciphers. Next, some more details of the following chapters are given.

In Chapter 2 we more formally introduce the main topic of the work; the stream ciphers and their components. Since all of the considered ciphers are based on *Linear*

Feedback Shift Registers (LFSRs) we discuss some of their properties and the advantages and disadvantages of using them in a design. An LFSR is a device use to produce a sequence of symbols, with good statistical properties. However, this sequence is very easy to predict given a certain amount of generated symbols. This is due to the linear property of the device. Therefore, in order to use LFSRs in a cryptographical primitive, and particularly in a stream cipher, the linearity must be destroyed. Thus, Boolean functions and S-Boxes are introduced together with their basic properties. Also in this chapter, some classical stream cipher designs are presented, followed by an introduction to the most common attacks on stream ciphers.

In Chapter 3, two different distinguishing attacks are presented, targeting the stream ciphers SOBER-t16 and SOBER-t32. The SOBER-16 cipher is an additive stream cipher with a symbol size of 16 bits. The final step before generating the output keystream is a so called stuttering unit, which irregularly decimates the output sequence. Firstly, a simplified version of SOBER-t16 where this stuttering unit is removed, is attacked. The approach taken is to linearise the nonlinear filter and measure the correlation between the output using that assumption and the true output from the nonlinear filter. The correlation is measured for the 16 bit input and output words of the nonlinear filter. Then, the stuttering unit is reinserted, and the attack is applied again, but now with the additional complexity of guessing where in the decimated output the needed observations are visible. For SOBER-t32, a different attack based on a bitwise correlation through the nonlinear filter is analysed. This attack also targets the simplified version of SOBER-t32, where the stuttering unit is removed. Finally in this chapter, we give some related work by other authors, extending these results to the case where the stuttering unit in SOBER-t32 is present.

Chapter 4 begins with a brief overview of the security management in the GSM cell phone system. This serves as an introduction to the attack on the stream cipher $A5/1$, used for privacy in GSM. The attack is based on a very weak key initialisation, where the two components, the secret key and the public frame number, are initialised in a linear fashion. This allows for a separation of the LFSR output from these two contributions. During initialisation, the cipher is irregularly clocked 100 times before producing keystream material. By weighting the probabilities of the number of clockings actually performed, the attack can estimate the secret key part based on the keystream output with high probability.

In Chapter 5, another nowadays widely used cipher is analysed, the E_0 stream cipher. This cipher is defined in the Bluetooth standard for short-range wireless communication. The cipher consists of four LFSRs, the outputs of which are xored together with the output of a finite state machine. A new stronger correlation in the output sequence from the finite state machine is presented and using that, an initial state recovery attack is mounted. The attack requires a much longer received keystream sequence than allowed in the Bluetooth standard, so the attack is merely a theoretical attack on E_0 and not on its usage in Bluetooth. These results were first presented in 2000 and several better attacks have been presented by other authors and are briefly

discussed.

A distinguishing attack on the shrinking generator is presented in Chapter 6. The shrinking generator is a clock controlled stream cipher built from two linear feedback shift registers and an additional selection logic to irregularly decimate the output keystream. The main idea for the attack is that the majority bit of the blocks, centred around the tap positions in the feedback polynomial, fulfil the linear recursion more often than random. The attack attempts to estimate those majority bits and from that distinguish the keystream output from a random sequence. A theoretical analysis of the expected complexity and required number of observed keystream bits is also given and compared to recent work by other authors.

In Chapter 7, another clock controlled stream cipher, the self-shrinking generator, is analysed. Two classes of weak feedback polynomials are given. For the first class we present both a distinguishing attack and a initial state recovery attack. The distinguishing attack is very efficient and the required length of the observed keystream only grows linearly in the length of the shift register. For the second class of weak feedback polynomials, which is a much more general class, a distinguishing attack is presented. The exact complexity of this approach is still an open problem, but we show the validity and efficiency of the attack using simulation results.

The design part of this thesis is presented in Chapter 8. Two new stream ciphers SNOW 1.0 and SNOW 2.0 are introduced. Both versions are built from a single linear feedback shift register and for each clocking of the ciphers, 32 pseudo-random bits are produced. This makes the SNOW family very fast in especially software implementations. Encryption speeds of up to 4.5 CPU clock cycles per byte have been achieved. SNOW 1.0 was a candidate for the NESSIE project, but was removed during the final phase due to two different attacks. Both these attacks are discussed and the next version, SNOW 2.0, is presented, in which several of the weaknesses in SNOW 1.0 have been improved. A very recent approach for attacking SNOW 2.0 is also discussed.

Finally, in Chapter 9, some concluding remarks are given and some possible further work in this area is discussed.

2

Introduction to stream ciphers

In this chapter we will introduce a class of primitives called stream ciphers. The main properties of stream ciphers separating them from (pure) block ciphers are that the encryption function works on individual symbols (letters) of the underlying alphabet and that the encryption function is time-varying. Block ciphers tend to have large block sizes: 64, 128, or 256 bits, whereas the symbol size used in a stream cipher is smaller, typically 1, 8, 16 or 32 bits. This symbol size (if larger than 1 bit) is often equal to the word size of the CPU, to take maximum advantage of the data bus size. Many dedicated hardware stream ciphers have a symbol size of 1, for example the stream cipher E_0 used in Bluetooth [10].

Many of the properties of stream ciphers make them suitable for use in telecommunication and low-level network encryption. They are normally much faster than block ciphers and do not cost more to implement in terms of hardware gates or memory, nor software memory. They also have limited error propagation if the encrypted data is corrupted on the channel, and limited buffer requirements since the symbol size is relatively small and each symbol is encrypted independently of the others.

This chapter begins with a general introduction and presentation of stream ciphers. Section 2.2 gives some basic definitions and properties of Linear Feedback Shift Registers (LFSRs), a component widely used inside stream ciphers. The next sections, Sections 2.3 and 2.4, are an introduction to Boolean functions and vector Boolean functions (S-Boxes). Section 2.5 discusses some classical stream cipher design principles and Section 2.6 introduces some well known approaches to cryptanalysis on stream ciphers. In Section 2.7, a general discussion of what it means for a stream cipher to be secure is given. Finally, in Section 2.8, a summary of the chapter is given.

2.1 Stream ciphers

Stream ciphers are divided into *synchronous* and *self-synchronous*. We first take a look at the encryption process for a synchronous stream cipher. It can be described at time $t \geq 0$ by the equations

$$\begin{aligned} \sigma_{t+1} &= f(\sigma_t, k), \\ z_t &= g(\sigma_t, k), \\ c_t &= h(z_t, m_t), \end{aligned}$$

where σ_0 is the *initial state* and may depend on the key k. f is the *next-state function*, g is the function which produces the *keystream* z_t, $t \geq 0$ and h is the *output function* which combines the keystream and the plaintext to produce the ciphertext c_t, $t \geq 0$. The procedure for encryption is pictured in Figure 2.1. Basically it works as a finite

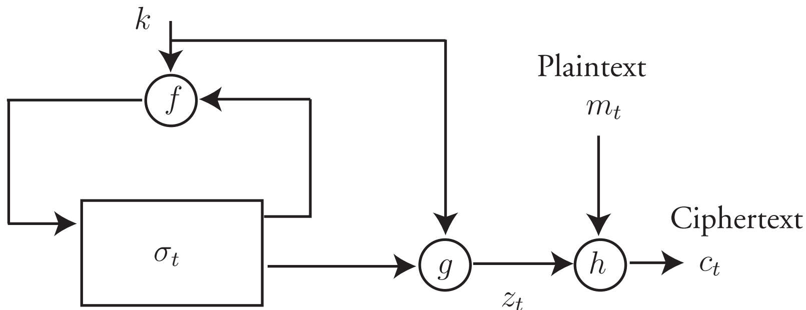


Figure 2.1: General structure of a synchronous stream cipher.

state machine, where the current state σ_t together with the key produces the output keystream. Then the next state is determined from the key and the current state using the next-state function. We can formally state the following definition.

Definition 2.1: *A synchronous stream cipher is a finite state machine for which the keystream is generated from the key, but independently of the plaintext message and the ciphertext.*

So, at each time instance $t \geq 0$, the cipher produces a new keystream symbol $z_t \in \mathcal{Z}$, where $\mathcal{Z}$ typically is the binary field $\mathbb{F}_2$ or some extension $\mathbb{F}_{2^W}$ of the binary field. The *symbol size* for the stream cipher is then defined to be W bits. The message $\mathbf{m}$ is split into symbols of size W bits $\mathbf{m} = m_0, m_1, m_2, \ldots, m_{N-1}$, where $m_t \in \mathcal{M}$ and encrypted symbol by symbol using the output function h. The output is a sequence of ciphertext symbols $\mathbf{c} = c_0, c_1, c_2, \ldots, c_{N-1}$, where $c_t \in \mathcal{C}$. Here, $\mathcal{M}$ and $\mathcal{C}$ are the plaintext alphabet and ciphertext alphabet respectively.

It is very common to use the xor function as the output function h. Since xor is the field addition operation, we normally denote such a cipher, having $\mathcal{Z} = \mathbb{F}_{2^W}$ for some $W \geq 1$ and xor as the output function, an *additive* stream cipher. Note that for an additive stream cipher to work, we require that both the plaintext alphabet $\mathcal{M}$, and the ciphertext alphabet $\mathcal{C}$, are equal to $\mathcal{Z}$, $\mathcal{M} = \mathcal{C} = \mathcal{Z}$, otherwise the field addition operation is not valid.

We can also describe the additive stream cipher as a *pseudo-random number generator* or a *keystream generator* whose output is xored to the plaintext. The key is used to initialise or seed the generator, which starts to produce pseudo-random bits. We note the similarity to the One-Time-Pad; instead of having the complete keystream as the secret key, we want to have a smaller key which is used for seeding. Then we want the generator to produce a keystream which is as random looking as possible. Figure 2.2 shows a keystream generator used as an additive stream cipher. The decryption of

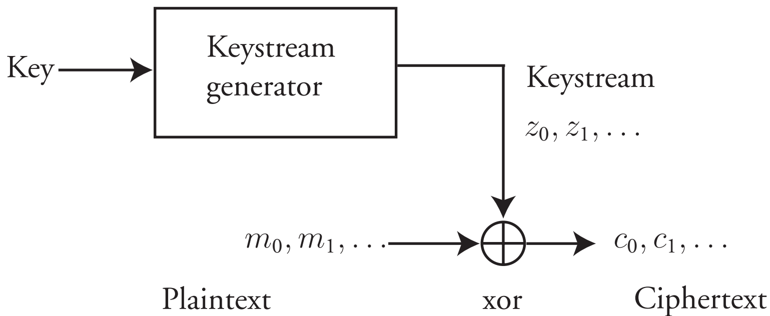


Figure 2.2: An additive stream cipher constructed from a keystream generator.

the ciphertext is very easy using a synchronous stream cipher. The receiver only needs to generate the same keystream as the sender and apply the inverse[1] function of h, $h^{-1}(z_t, c_t) = m_t$. For the additive stream cipher, we have $h = h^{-1}$ which results in the useful property that the decryption device is exactly the same as the encryption device.

As was briefly mentioned before, a synchronous stream cipher is not particularly vulnerable to errors in the transmission, since each symbol is encrypted independently. An active attacker can change a particular ciphertext symbol and render the corresponding plaintext symbol erroneous, but all other symbols will remain correct. The receiver has no direct possibility of validating the received message, thus additional mechanisms for message authentication are needed to defend against these kind of attacks.

[1] with respect to the plaintext symbol m_t, since the keystream symbol should be identical (and thus fixed) for both sender and receiver.

However, if the attacker deletes or inserts a symbol into the ciphertext—or if that happens by accident—the sender and receiver lose synchronisation and all plaintext symbols following the deleted one will become erroneous during decryption. To defeat this problem, perfect synchronisation between sender and receiver needs to be ensured and techniques for detecting loss of synchronisation employed.

There are many applications where the synchronisation problem is much more severe than a corrupt ciphertext symbol. For example, in a streaming video sequence some erroneous symbols will only affect the picture over a limited area and during a small time frame. But a synchronisation error will generate a completely useless video. In such a setting, we could use a frame based communication protocol, where the message sequence is first divided into smaller *frames* which are numbered with a frame number, see Figure 2.3. We then add a feature to the stream cipher called an *Initialisation Value* (IV), which is a publicly known value used in the initialisation of the stream cipher together with the secret key. Now, with a fixed key but with a changing IV, the stream cipher will produce different sequences of keystream material for each IV. For each frame the receiver tries to decrypt, he looks at the public frame number attached to the frame of encrypted information and pre-initialises the stream cipher with the new frame number as IV and the secret key, and then decrypts the information. If synchronisation is lost for a single frame it will only affect a small amount of information, until a new frame arrives and he can resynchronise. An important practical issue is that the reinitialisation of the cipher with a new IV should be very fast, to be able to handle high information rate sources such as streaming video.

Frame number 1	Encrypted data
Frame number 2	Encrypted data
Frame number 3	Encrypted data
Frame number 4	Encrypted data
Frame number 5	Encrypted data

Figure 2.3: Data is split up into separate frames. Each frame consists of a publicly known unencrypted frame number and the encrypted payload.

The other type of stream cipher is the self-synchronous stream ciphers. These ciphers have the property that they will resynchronise after a finite number of received ciphertext symbols. Thus the state of such a cipher is only dependent on the previous generated keystream symbols. We have the following formal definition.

Definition 2.2: *A self-synchronising stream cipher is a finite state machine for which the keystream is generated as a function of the key and a fixed number of the previous ciphertext symbols.*

It can be described at time $t \geq 0$ by the equations

$$\begin{aligned} \sigma_t &= (c_{t-v}, c_{t-v+1}, \ldots, c_{t-1}), \\ z_t &= g(\sigma_t, k), \\ c_t &= h(z_t, m_t), \end{aligned}$$

where σ_0 is the *initial state*, k is the key, g is the function which produces the *keystream* z_t, and h is the *output function* which combines the keystream and the plaintext to produce the ciphertext c_t. The initial state $\sigma_0 = (c_{-v}, c_{-v+1}, \ldots, c_{-1})$ may be a publicly known value. The encryption and decryption processes differ in contrast to

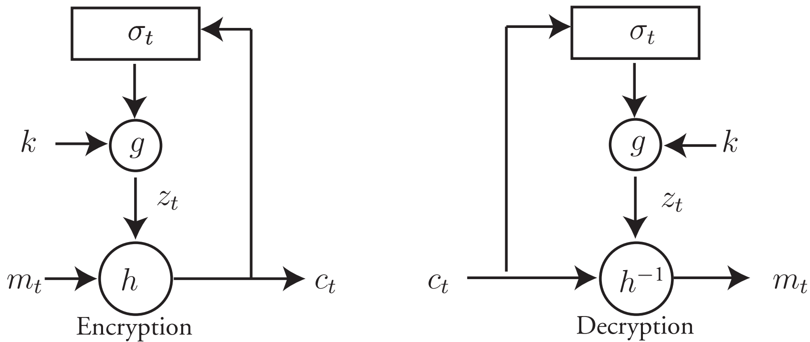


Figure 2.4: General structure of a self-synchronising stream cipher.

the synchronous stream ciphers and are pictured in Figure 2.4, where h^{-1} is the inverse of h.

Since the keystream depends on the last v ciphertext symbols, the cipher will resynchronise after v symbols if there is a transmission error. If that happens, the next v symbols will be erroneous and the error propagation is thus worse than for a synchronous stream cipher. However, if some ciphertext symbols are deleted or inserted during transmission, the self-synchronising cipher will recover after v correct ciphertext symbols, whereas the synchronous ciphers will never regain synchronisation.

Before we present some classical and interesting stream cipher designs, we shall introduce some of the basic structures often found inside stream ciphers.

2.2 Linear feedback shift registers

Recalling the keystream generator and its similarity to the OTP, we state that the fundamental property of a keystream generator is to produce as random looking symbols as possible. The distribution of symbols should be uniform and unpredictable. A good start is to use a Linear Feedback Shift Register (LFSR) for achieving a good distribution. As we will see later, the direct output of an LFSR is not a good keystream generator since each symbol produced is simply a linear combination of the previous symbols, and thus very easy to predict. Nevertheless, LFSRs are widely used components inside stream ciphers.

An LFSR is a device made up by registers, able to hold one symbol at a time. The symbols are elements from a field $\mathbb{F}_q$, over which we have chosen to define the LFSR. In stream cipher applications we often have $q = 2$ (binary field) or some extension field of the binary field $q = 2^W$, where W is the symbol size of the stream cipher. Initially we can think of an LFSR as a hardware construction, though it is very easy to implement in software as well. Thus we assume a system clock which is responsible for the timing of all events. Figure 2.5 shows a general LFSR, where the circles denote multiplication with the constant c_i and $\oplus$ is the field addition operation. At each clocking of the LFSR, the registers read a new symbol from their respective input, and as the registers are coupled in series, the symbols move forward at each clocking. The first register receives a new symbol which is a linear combination of the symbols found in the registers after the previous clocking. The exact linear combination used for producing the feedback symbol is determined by the *feedback coefficients* $c_0, c_1, \ldots, c_l$ shown in Figure 2.5. Since we need the actual feedback connection c_0 to get any symbols into the register, one normally assumes $c_0 = 1$. As we do not need more registers than necessary to make the feedback connection work, we also assume $c_l \neq 0$ and define the *length* of the LFSR to be l. At each time $t \geq 0$ the device is clocked,

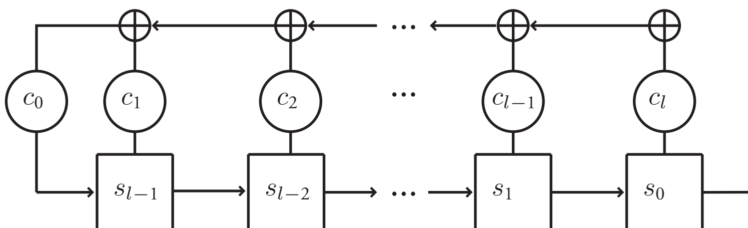


Figure 2.5: General form of a Linear Feedback Shift Register (LFSR) of length l.

and we obtain a new symbol $s_t \in \mathbb{F}_q$ at the output of the device. Due to the linear

feedback, the symbols s_t will always fulfil the *linear recurrence equation*

$$c_0^{-1} s_{t+l} - c_1 s_{t+l-1} - c_2 s_{t+l-2} - \ldots - c_l s_t = 0, \quad t \geq 0.$$

If the field characteristic is 2, and we assume $c_0 = 1$, we get the simpler form

$$\sum_{j=0}^{l} c_j s_{t+l-j} = 0, \quad t \geq 0. \tag{2.1}$$

Using the feedback coefficients, we can give a compact description of an LFSR through its *feedback polynomial*.

Definition 2.3: *The feedback polynomial g is the polynomial $g(x) = c_0 - c_1 x - c_2 x^2 - \ldots - c_l x^l$. The feedback polynomial is an element in the ring of polynomials $\mathbb{F}_q[x]$, with x being the indeterminant.*

The connection at c_0 is called the *feedback position* and each $c_j \neq 0,\ 1 \leq j \leq l$ is called a *tap position*. An alternative description is the *characteristic polynomial* $f(x)$, which is the reciprocal polynomial of $g(x)$, $f(x) = -c_l - c_{l-1}x - \ldots - c_1 x^{l-1} + c_0 x^l$.

The content of the register at time t is $\mathbf{S}_t = (s_{t+l-1}, s_{t+l-2}, \ldots, s_t)$ and is called the *state* of the LFSR at time t. We note that this state (taken at any time), together with the feedback polynomial, completely determines the produced sequence $s_t, t \geq 0$. The first l symbols to be produced, $\mathbf{S}_0 = (s_{l-1}, s_{l-2}, \ldots, s_0)$, are loaded into the registers at the start, and we denote this state the *initial state* or the *starting state* of the LFSR.

Since there are only a finite number of possible states q^l, the sequence produced by the LFSR must repeat itself after a finite period, i.e. for every starting state we can find a T such that $s_t = s_{t+T},\ t \geq 0$. The period depends on properties of the feedback polynomial, and for our use in stream ciphers, we confine ourselves to the following definitions and theorem regarding the period.

Definition 2.4: *A polynomial $g(x) \in \mathbb{F}_q[x]$ is said to be irreducible* over $\mathbb{F}_q$ *if it cannot be factored into polynomials of smaller positive degrees in the ring of polynomials $\mathbb{F}_q[x]$.*

Let $\mathbb{F}_{q^l}^*$ be the non-zero elements of $\mathbb{F}_{q^l}$. It is well known that $\mathbb{F}_{q^l}^*$ forms a group together with the field multiplication operation. An element $a \in \mathbb{F}_{q^l}^*$ is called a *generator* if $\{a^i : 0 \leq i \leq q^l - 2\} = \mathbb{F}_{q^l}^*$, i.e. the powers of a generate all non-zero elements of $\mathbb{F}_{q^l}$.

Definition 2.5: *The extension field $\mathbb{F}_{q^l}$ of a field $\mathbb{F}_q$ is called a* splitting field *for the polynomial $g(x) \in \mathbb{F}_q[x]$ if $g(x)$ factors completely into linear factors in $\mathbb{F}_{q^l}[x]$ and $g(x)$ does not factor completely into linear factors over any proper subfield of $\mathbb{F}_{q^l}$ containing $\mathbb{F}_q$.*

Definition 2.6: *An irreducible polynomial* $g(x) \in \mathbb{F}_q[x]$ *of degree* l *is said to be primitive if the root of* $g(x)$ *in the splitting field* $\mathbb{F}_{q^l}$ *is a generator of the multiplicative group* $\mathbb{F}^*_{q^l}$*.*

The following theorem is given without proof [75].

Theorem 2.7: *The period* T *of an LFSR with a primitive feedback polynomial* $g(x) \in \mathbb{F}_q[x]$ *of degree* l *and with a non-zero starting state is* $T = q^l - 1$*.* □

Thus, if we start with a non-zero state in the LFSR and use a primitive feedback polynomial, all possible states except the all-zero state will appear during a period. An LFSR with a primitive feedback polynomial is also called a *maximum-length* LFSR, and the sequence produced is called a *maximum-length sequence*. Note that again the initial state must be non-zero for the sequence to be maximum-length, and hereafter it is assumed that the starting state is as such.

An LFSR is said to *generate* an infinite sequence $\mathbf{s} = s_0, s_1, \ldots$ if there exists an initial state for the LFSR such that the output sequence is equal to $\mathbf{s}$. For a finite sequence $\mathbf{s} = s_0, s_1, \ldots, s_{n-1}$ of length n we say that an LFSR generates $\mathbf{s}$ if there exists an initial state for the LFSR such that the vector of the first n symbols produced is equal to $\mathbf{s}$.

Next, we introduce the *linear complexity* of a sequence.

Definition 2.8: *The linear complexity of a sequence* $\mathbf{s} = s_0, s_1, \ldots, s_i \in \mathbb{F}_q$*, denoted* $L(\mathbf{s})$*, is the length of the shortest LFSR, defined over* $\mathbb{F}_q$*, that generates the sequence. If the sequence is the all-zero sequence, then the linear complexity is 0, and if no LFSR can generate the sequence, then* $L(\mathbf{s}) = \infty$*.*

The linear complexity can be determined with the Berlekamp-Massey algorithm [78] which efficiently computes the feedback polynomial of the LFSR given at least $2L(\mathbf{s})$ of output symbols. Note again that a pure LFSR is not a good keystream generator. As an example we can take a binary LFSR of length 128, which produces a maximum-length sequence of length $2^{128} - 1$. The Berlekamp-Massey algorithm needs only 256 consecutive bits in a known-plaintext attack to fully determine the feedback polynomial and thus the complete sequence.

Sequences generated by maximum-length LFSRs have good statistical properties, desirable for keystream generator construction, but we need to destroy the linearity, i.e. increase the linear complexity, before the sequence can be used. There are several methods for doing this as we will see in Section 2.5. One classical approach is to use several binary LFSRs and combine the output from each of them using a Boolean function as pictured in Figure 2.6.

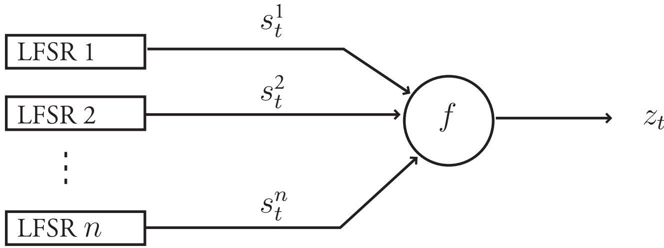


Figure 2.6: A nonlinear combiner where the output of n LFSRs are combined via the Boolean function f in order to destroy the linearity of the LFSR sequences.

2.3 Boolean functions

A Boolean function is essentially a function which maps one or more binary input variables to one binary output variable. More stringently, we write this as a mapping from a vector of binaries $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, where $x_i \in \mathbb{F}_2$, $1 \leq i \leq n$ to a single output bit

$$f : \mathbb{F}_2^n \to \mathbb{F}_2.$$

For n input variables there exist 2^{2^n} distinct Boolean functions and we denote the set of Boolean functions in n variables by $\mathcal{B}_n$. There are several ways to represent a Boolean function. If the number of input variables is small, a truth table can be constructed where all possible input vectors are listed with the corresponding output value. Thus, we can represent the Boolean function $f \in \mathcal{B}_n$ as the string of length 2^n of output values $f = [f(0, \ldots, 0, 0)\ f(0, \ldots, 0, 1)\ f(0, \ldots, 1, 0) \ldots f(1, 1, \ldots, 1)]$. Table 2.1 shows a truth table for a particular Boolean function of three variables.

If the number of input variables is large, the listing of all possible vectors is infeasible and we have to resort to a more compact description, for example the *algebraic normal form* (ANF)

$$f = \sum_{\mathbf{u} \in \mathbb{F}_2^n} \lambda_{\mathbf{u}} \left(\prod_{i=1}^{n} x_i^{u_i} \right), \quad \lambda_u \in \mathbb{F}_2,\ \mathbf{u} = (u_1, u_2, \ldots, u_n). \tag{2.2}$$

A product of m distinct variables is called an mth *order product* of the variables. To simplify (2.2), we can state that every Boolean function $f(x_1, x_2, \ldots, x_n)$ can be written as a (modulus 2) sum of distinct mth order products of its variables, $0 \leq m \leq n$.

Remark. In the context of Boolean functions, the addition sign $+$ and summation sign $\sum$ represent addition in the field $\mathbb{F}_2$, i.e. a modulus 2 sum.

x_1	x_2	x_3	$f(\mathbf{x})$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 2.1: The truth table of the Boolean function $f(x_1, x_2, x_3) = x_1x_2 + x_2x_3 + x_3$.

The *Hamming weight* of a binary vector is the number of ones in the vector, and the algebraic degree (or simply the degree) of a Boolean function f, denoted $deg(f)$, is the maximum value in (2.2) of the Hamming weight of $\mathbf{u}$ such that $\lambda_u \neq 0$, or simply the highest order of the terms in the ANF.

In the nonlinear combination generator, pictured in Figure 2.6 on page 25, it is desirable to have a high algebraic degree in the Boolean function. The following well known theorem, given without proof, explains the situation [85].

Theorem 2.9: *Assume a nonlinear combination generator with n maximum-length LFSRs, whose lengths $L_1, L_2, \ldots, L_n$ are pairwise distinct and greater than 2, combined with a Boolean function $f(x_1, x_2, \ldots, x_n)$ given in ANF. Then the linear complexity of the keystream is $f(L_1, L_2, \ldots, L_n)$ evaluated over the integers rather than over $\mathbb{F}_2$.* □

For cryptographical applications, several other interesting properties of Boolean functions have to be considered. First of all, we say that a Boolean function is *balanced* if the number of zeros in the output column of the truth table is equal to the number of ones. Equivalently we can say that f is balanced if the probability $P(f(\mathbf{x}) = 0) = P(f(\mathbf{x}) = 1) = \frac{1}{2}$, when $\mathbf{x}$ is chosen uniformly over $\mathbb{F}_2^n$.

Definition 2.10: *Functions of a degree of at most one are called affine. The set of all affine functions in n variables is denoted $\mathcal{A}_n$. We can write*

$$\mathcal{A}_n = \{a_0 + a_1x_1 + a_2x_2 + \ldots + a_nx_n : a_i \in \mathbb{F}_2,\ 0 \leq i \leq n\}. \qquad (2.3)$$

An affine function with constant term $a_0 = 0$ is called a linear function, thus the set of linear functions in n variables is a subset of $\mathcal{A}_n$.

We define the *Hamming distance* between two functions $f(\mathbf{x}), g(\mathbf{x}) \in \mathcal{B}_n$ as the

number of entries in the truth table output column where $f(\mathbf{x})$ and $g(\mathbf{x})$ differ,

$$d_H(f,g) = \#\{\mathbf{x} \in \mathbb{F}_2^n | f(\mathbf{x}) \neq g(\mathbf{x})\},$$

where $\#A$ denotes the cardinality of the set A.

Next, we introduce the measure of nonlinearity.

Definition 2.11: *The nonlinearity of a Boolean function $f \in \mathcal{B}_n$, denoted N_f, is the Hamming distance to the nearest affine function in $\mathcal{A}_n$,*

$$N_f = \min_{g \in \mathcal{A}_n} d_H(f,g).$$

A high nonlinearity is a desirable property since it will decrease the correlation between the output and the input variables or a linear combination of input variables. Many of the known attacks on stream ciphers utilise the weakness of such a correlation between the combining Boolean function and some affine function.

Another important measure is the *correlation immunity* of the Boolean function. The concept was first introduced by Siegenthaler [112] from an information theory point of view.

Definition 2.12: *Let $X_1, X_2, \ldots, X_n$ be independent binary random variables, each taking the values 0 or 1 with probability $\frac{1}{2}$. A Boolean function $f(x_1, x_2, \ldots, x_n)$ is said to be t-th order correlation immune, if for each subset of t variables $X_{i_1}, X_{i_2}, \ldots,$ X_{i_t} with $1 \leq i_1 \leq i_2 \leq \cdots \leq i_t \leq n$, the random variable $Z = f(X_1, X_2, \ldots, X_n)$ is statistically independent of the random vector $(X_{i_1}, X_{i_2}, \ldots, X_{i_t})$.*

A Boolean function which is both balanced and t-th order correlation immune is said to be *t-resilient*.

Siegenthaler [112] showed that there is a tradeoff between the algebraic degree and the order of correlation immunity.

Theorem 2.13: *Let $f(\mathbf{x})$ be a balanced Boolean function in n variables of algebraic degree d which is t-th order correlation immune. Then the following upper bound must hold*

$$\begin{cases} d + t \leq n - 1 & \text{if } 1 \leq t \leq n-2, \\ d + t \leq n & \text{if } t = n-1. \end{cases} \tag{2.4}$$

□

To summarise the properties of Boolean functions and the implications of their value when used as a combining function in a keystream generator we conclude that

Algebraic degree A high algebraic degree is desirable since it increases the linear complexity of the resulting keystream.

Nonlinearity A high nonlinearity gives a weaker correlation between the input variables and the output variable and increases the resistance to correlation attacks.

Correlation immunity High correlation immunity forces the attacker to consider several input variables jointly and thus decreases the vulnerability of divide-and-conquer attacks.

The specifics of the generic attacks *correlation attacks* and *divide-and-conquer attacks* will be given in Section 2.6. The very basic properties of Boolean functions have been introduced, and a thorough discussion on more advanced properties and details on the construction of good Boolean functions is given in [94].

2.4 S-Boxes

An S-Box (Substitution Box) can be considered as a vector output Boolean function. Formally we have the following definition.

Definition 2.14: *A $n \times m$ S-Box is a mapping*

$$S : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m.$$

We write $S[\mathbf{x}] = \mathbf{y}$ to denote the output value $\mathbf{y} \in \mathbb{F}_2^m$ of the S-Box S on input $\mathbf{x} \in \mathbb{F}_2^n$. Writing the S-Box as a function of a binary vector, we have $S[\mathbf{x}] = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x}))$ where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is an n bit binary vector and $f_i(\mathbf{x}), i = 1 \ldots m$ are Boolean functions of n bit inputs.

To specify an S-Box we could of course give the component Boolean functions $f_i, 1 \leq i \leq m$. However, this is normally cumbersome since Boolean functions in the ANF representation often have many terms and would be impractical to write out. Nevertheless, for a hardware realisation of an S-Box, this might be the preferred description. For S-Boxes where n is not too large, one could specify it by a table of 2^n entries where each entry is a m bit number. This is the preferred way of dealing with smaller S-Boxes in software, implementing them as table lookups.

A third possibility is to consider the S-Box an algebraic mapping and give the algebraic expression of the mapping. Naturally this is only feasible if the S-Box was *defined* using that mapping. Otherwise it is a hopeless task of finding a simple algebraic expression for a random mapping. One example of an algebraic description is the inverse mapping $x \rightarrow x^{-1}$ in $\mathbb{F}_{2^n}$. Here $\mathbf{x}$ is considered an element in $\mathbb{F}_{2^n}$ rather than an element in the vector field $\mathbb{F}_2^n$. The inverse mapping is used, for example, in the new AES block cipher.

The interesting properties of S-Boxes for stream ciphers are very much the same as those for Boolean functions, but now extended to a vector domain. This extension is done by considering the minimum value over all linear combinations of the component output functions $f_i, 1 \leq i \leq m$. We have the following formal definitions and lemma, where $\mathbb{F}^\star$ is the multiplicative subgroup of $\mathbb{F}$ and S is an $n \times m$ S-Box.

Lemma 2.15 (from [110]): *A function $S = (f_1, f_2, \ldots, f_m)$, where each f_i, $1 \leq i \leq m$, is a Boolean mapping $\mathbb{F}_2^n \to \mathbb{F}_2$, is uniformly distributed (balanced) if and only if all nonzero linear combinations of $f_1, f_2, \ldots, f_m$ are balanced.* □

The definition of algebraic degree also considers linear combinations of the component functions.

Definition 2.16 (from [94]): *The algebraic degree of S is defined as the minimum degree of all nonzero linear combinations of the component functions of S, i.e.*

$$deg(S) = \min_{\tau \in \mathbb{F}_2^{m*}} deg(\sum_{i=1}^{m} \tau_i f_i), \text{ where } \tau = (\tau_1, \ldots, \tau_m) \in \mathbb{F}_2^{m*}.$$

The nonlinearity follows in a similar manner.

Definition 2.17 (from [93]): *The nonlinearity of $S = (f_1, f_2, \ldots, f_m)$, denoted N_S, is defined as the minimum among the nonlinearities of all nonzero linear combinations of the component functions of S, i.e.*

$$N_S = \min_{\tau \in \mathbb{F}_2^{m*}} nl(\sum_{i=1}^{m} \tau_i f_i), \text{ where } \tau = (\tau_1, \ldots, \tau_m) \in \mathbb{F}_2^{m*},$$

where $nl(f)$ denotes the nonlinearity of f.

S-Boxes are very common in block cipher designs but have been used more and more in stream ciphers as well. Typically we see 8×8 S-Boxes or 8×32 S-Boxes conforming with a byte size or a cipher symbol size of 32 bits.

Depending on where the S-Box is employed in the stream cipher, the above properties are of more or less importance. Almost always we would like to have a balanced function, since unbalanced functions have a probabilistically biased output, given a uniform distribution on the input variables. This bias could be used in a correlation attack. The nonlinearity should be as high as possible since the primary goal of the S-Box is to destroy the linearity of the input. The algebraic degree should also be high, and recent research on algebraic attacks (see Section 2.6) might indicate that the algebraic degree must be higher than previously believed. Also low algebraic equations which relate input and output variables with high probability, should be avoided due to these algebraic attacks.

The construction of good S-Boxes is a difficult subject and an overview of results and methods, focusing on stream ciphers, is given in [94].

2.5 Some classic stream cipher designs

In this section we will present some general design techniques and also look at some specific designs. Stream ciphers based on LFSRs can be divided into three general categories; *nonlinear combination generators*, *nonlinear filter generators*, and *clock-controlled*

generators. Naturally, the classification of ciphers into these categories is not distinct as several design methods can be used simultaneously.

The nonlinear combination generator was introduced in the discussion on Boolean functions c.f. Figure 2.6 on page 25. The generator consists of n LFSRs, whose outputs are combined in a Boolean function f. The output of the Boolean function is the keystream output. The Boolean function f must have high algebraic degree, high nonlinearity and preferably a high order of correlation immunity. In view of Theorem 2.13, we know that we must employ a large number of LFSRs to get both high algebraic degree and a high order of correlation immunity.

The nonlinear filter generator takes a different approach and uses one single LFSR, from which the inputs to the Boolean function are taken, see Figure 2.7. In this case

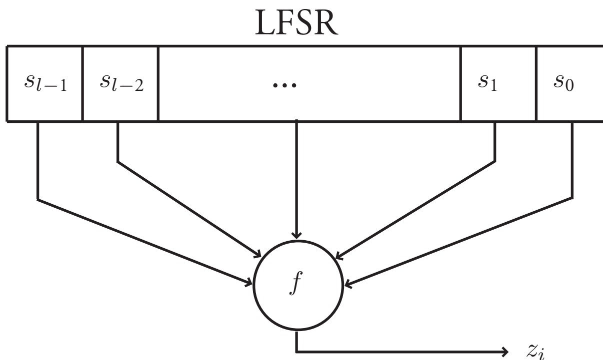


Figure 2.7: Structure of a nonlinear filter generator.

the Boolean function f is called the filtering function. Not all elements of the LFSR need to be taken as inputs to the filtering function. The nonlinear filter approach is a more efficient way of building stream ciphers in software if the LFSR is built over an extension field $\mathbb{F}_{2^w}$. This is due to the fact that shifting a maximum-length LFSR defined over $\mathbb{F}_{2^w}$ is a rather costly operation in software.

Key [69] proved an upper bound on the linear complexity $L(\mathbf{s})$ for a nonlinear filter generator of $L(\mathbf{s}) \leq \sum_{i=1}^{d} \binom{l}{i}$, where d is the algebraic degree of f, l is the length of the LFSR and $\mathbf{s}$ is the generated keystream. In [47], Golić gives a set of criteria needed for the nonlinear filter generator to be able to resist many known attacks.

In the presentation of both the nonlinear combination generator and the nonlinear filter generator given in this section, we considered only a pure Boolean function as the combiner/filter. We could however extend the ideas and use a time varying function or a combiner/filter mechanism with memory. We will use the term *Finite State Machine* (FSM) generators for this type of generator. Two recent examples of

FSM generators are the E_0 stream cipher used in the Bluetooth standard [10] for short wireless networking and SNOW, a stream cipher family presented in Chapter 8.

The principal difference from the pure combiner/filtering generator is that a part of the state space is updated via a nonlinear process. Hence, for FSM generators the properties of pure Boolean functions are not directly applicable, especially the tradeoff between algebraic degree and correlation immunity does not hold. Golić [42] has pointed out the importance of small correlations between a linear combination of input variables and output variables, also in combiners with memory.

Another interesting approach is to include key dependent S-Boxes in the combining/filtering function. This makes cryptanalysis much harder than for a pure Boolean function since effectively, the cipher is not completely specified until the key is known. Rose and Hawkes recently proposed the cipher Turing [102] based on these ideas.

The final category of stream ciphers is the clock-controlled generators. They differ from the above in the sense that the LFSRs are not clocked regularly. In the nonlinear combination generator and the nonlinear filter generator we have a system clock[2] which advances the LFSR one step for each clocking and for each clocking we produce a new keystream bit or several keystream bits. The idea in a clock-controlled generator, is to control the number of clockings of the LFSRs using some irregular signal. The clocking signal could be the output of another LFSR or some other internal variable of the cipher. The linearity of the output of the irregularly controlled LFSR is then destroyed and attacks based on a regular motion of the LFSR are harder to mount. One classical example of a clock-controlled generator is the *alternating step generator* proposed by Günther [41], pictured in Figure 2.8. The operating procedure of the

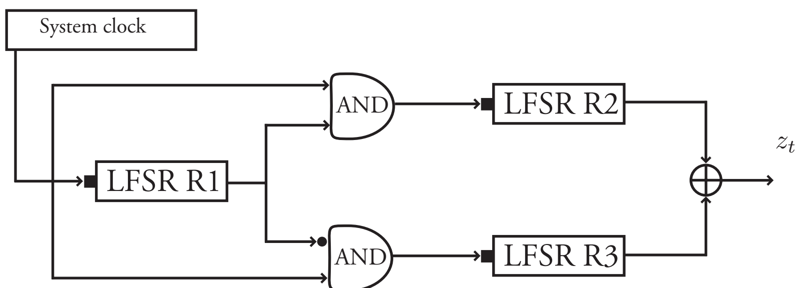


Figure 2.8: The alternating step generator. The black squares indicate the clocking input to the LFSRs and the solid circle on the lower AND-gate is an inverter.

generator is a repetition of the following steps.

[2] In a software implementation the system clock of the cipher is an imaginary or virtual clock and usually does not coincide with the CPU clock.

(*i*) Register R1 is clocked by the system clock.

(*ii*) If the output of R1 is 1 then R2 is clocked and R3 is not clocked but its previous output bit is repeated.

(*iii*) If the output of R1 is 0 then R3 is clocked and R2 is not clocked but its previous output bit is repeated.

(*iv*) The sum of the outputs from R2 and R3 is taken as the keystream output.

Two other very elegant clock-controlled generators, the *shrinking generator* and the *self-shrinking generator* are presented and analysed in Chapter 6 and 7 respectively.

Apart from the three types of stream ciphers based on LFSRs discussed here, there exists other designs that do not use an LFSR as a component. One example is the well-known cipher RC4, which is a stream cipher with a symbol size of 8 bits (1 byte). The state space consists of an array of 256 elements, containing the numbers $0 \ldots 255$. The array is slowly changing at each time instance by swapping elements in the array, and a new keystream symbol is produced by reading off different entries in the array. The index of extraction is also determined from the elements in the array. A more complete description and an attack can be found in [77].

Other interesting stream cipher designs which do not use an LFSR are the software oriented ciphers SEAL 3.0 [98, 99], SCREAM [54] and MUGI [120].

2.6 Generic attacks on stream ciphers

In this section we will introduce some well known generic attacks and point out some specific design criteria for decreasing their success probability. Most of the cryptanalysis on stream ciphers is performed under a known plaintext assumption. For the additive stream cipher, this means that the attacker has direct access to the keystream output from the generator. The attack with which we will compare other attacks, is the exhaustive key search attack. This attack can always be performed by simply trying all possible keys until the correct one is found.

When comparing attacks we have three complexity measures that are of interest.

Time complexity The required number of operations that are needed to carry out the attack. The operations counted can be either low-level operations as "CPU instructions" or "cipher clockings", or high-level operations as "table lookups" or "solving a matrix equation". The specific operation is not always specified since it often does not matter as long as we can perform the operation in polynomial time. The time complexity of the attack is normally exponential in the length of the LFSR or in the key length, and thus it is of less theoretical importance to specify which polynomial operation we are performing. However, in comparing attacks from a practical perspective, it can clearly be of interest.

We also separate the time complexity into *pre-computational* complexity and *active attack* complexity. The pre-computational part can be performed without the observed keystream and is often required to be performed only once. Then the result can be used successively for attacking the cipher with different keys. The active attack part is the complexity of the operations we need to perform while observing the keystream.

Data complexity The amount of observed keystream material that is required for the attack to be fully successful or successful with a certain probability.

Memory complexity The required amount of memory needed to perform the attack. This parameter is coupled with both the time and the data complexity. If we have a pre-computational phase in the attack the result must be stored in memory for later use in the active part. Some attacks might build a search tree in memory based on the observed keystream material. For other attacks we need the complete observed keystream available for random access and thus the memory complexity is equal to the data complexity.

When discussing complexities, it is common to use an *order* notation.

Definition 2.18 (Big Oh-notation [91]): *Let f and g be two functions mapping the natural numbers to themselves: f is $O(g)$ if and only if there exist natural numbers N and c such that, for all $n \geq N$, we have $f(n) \leq c \cdot g(n)$.*

Typically, the order notation states how the attack complexity grows depending on, for example, the length of the LFSR or the size of the key. For example, we have exponential growth $O(2^n)$, polynomial growth $O(n^3)$ or linear growth $O(n)$.

Consider the exhaustive key search attack on a nonlinear filtering generator with k bits of key, where the length of the LFSR is k bits. The key initialisation is simply done by loading the LFSR with the key bits. The attack is performed by storing the first $2k$ bits of the observed keystream. We then load each possible key into the cipher and clock it $2k$ times and compare the output of each run with the stored sequence. When we find a match, we have identified the correct key. We see that the time complexity is $O(k2^k)$, the data complexity is $O(k)$ and the memory complexity is $O(k)$.

Tradeoff attacks

For many ciphers we can do a tradeoff for the time, memory, and data complexity in the case of an exhaustive key search. Consider again the above attack on a nonlinear filter generator, but now we start with a pre-computation of the generated sequences for $2^{k/2}$ randomly selected keys. We store the first $2k$ bits of output for each chosen key together with the key used. In the active phase we observe $2^{k/2}$ bits of keystream material, generated with the unknown key. Now, scanning the observed sequence and for each position, we try to match the next $2k$ bits with the sequences we have

pre-computed. When we find a match we can directly get the state of the LFSR that generated the subsequence. Reversing the LFSR to the initial state, we have now found the correct key. This approach has time complexity $O(2^{k/2})$ for scanning the observed sequence and matching subsequences to our pre-computed database. The memory complexity is $O(k2^{k/2})$ for storing the pre-computed sequences and the respective key, and finally the data complexity is $O(2^{k/2})$.

This tradeoff between attack complexities is not only limited to an exhaustive key search, but could be employed in other attack scenarios as well, using the same basic ideas. Examples of tradeoff attacks can be found in [8, 86, 105, 7, 3].

To foil a tradeoff attack, the state space of the cipher must be at least twice the size of the key space. In the context of stream ciphers, this usually means that the combined lengths of the LFSRs in the cipher must be twice as large as the key size and during the initialisation of the cipher, the key material must be spread into the state space in a random fashion.

Guess-and-determine attacks

As the name suggests, in this attack we start by guessing some internal variables of the cipher (e.g. a part of the LFSR) and then try to determine the other variables based on the observed keystream and the evolution of the cipher in time. If our guess is correct, we can confirm it by running the cipher for some time and match the output from our trial generator with the observed sequence. If our guess is false, we simply make a new guess and start over again.

The time complexity of such an attack is $O(2^b)$, where b is the number of bits we have to guess, since in the worst case we have to try all possible combinations of the guessed bits. The difficult part of this attack is to find which part of the state space to guess in order to obtain the rest.

An example [41] of a guess-and-determine attack on the alternating step generator, presented in Section 2.5 (Figure 2.8), is to guess the initial state of the register R1. Knowing that, we can determine the clocking sequences for both R2 and R3, and the output of the generator is simply a sum of the output bits of R2 and R3. If the lengths of the registers are l_1, l_2, and l_3 respectively, we have, after $l_2 + l_3$ observed keystream bits, a system of linear equations for the initial states of R2 and R3 which is easily solved. The time complexity of this attack is $O(2^{l_1})$ for guessing the initial state of R1 correctly.

Some other examples of guess-and-determine attacks can be found in [58, 6].

Divide-and-conquer attacks and correlation attacks

The correlation attack was introduced by Siegenthaler in [112, 113]. Siegenthaler exploited the fact that in certain cases, one can find a *correlation* between some linear combination of input variables to the combining function and the output from the

combining function, and then use that correlation to extract information about the correlated input variables.

In the simplest case, a correlation means that the output is equal to one of the input variables with a probability not equal to 0.5. As a simple example we can take the nonlinear combining function to be the logical AND-gate with two inputs. For either input, we see that the output coincides with the input with probability 0.75. These correlation probabilities are usually written as $\frac{1}{2} + \varepsilon$, where ε is the correlation between the variables. For the case with an AND-gate, the correlation is $\varepsilon = 0.25$.

Now, take any nonlinear Boolean function f with some correlation property and an example can be given to illustrate Siegenthaler's attack.

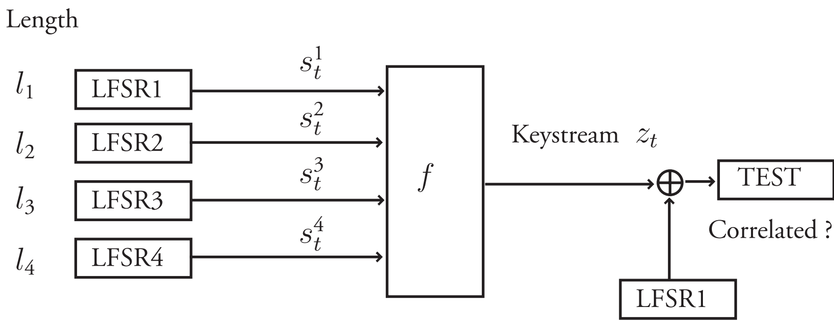


Figure 2.9: Principles of Siegenthaler's correlation attack.

Example 2.19: Assume a nonlinear combining generator with four LFSRs of lengths $l_1, \ldots, l_4$, and a given keystream $z_t,\ t \geq 0$, generated by some unknown initial state. To perform an exhaustive key search we would have to try $\prod_{i=1}^{4}(2^{l_i} - 1)$ different initial states. But, if we know that there is a correlation between the keystream output $z_t, t \geq 0$ and each of the LFSR streams $s_i^t,\ i \geq 0$, we can test each LFSR separately. So for the first LFSR1 we assume $P(s_t^1 = z_t) = \frac{1}{2} + \varepsilon$, and then try each possible initial state for LFSR1 in a separate LFSR and xor the produced sequence to the observed keystream, see Figure 2.9. For LFSR1 we have to try $(2^{l_1} - 1)$ different initial states.

The testing is done by measuring the number of zeros in the xored sequence. The zeros indicate that the keystream output and the separate LFSR have the same symbol in that position, and if the relative frequency of zeros is equal to the expected correlation $\frac{1}{2} + \varepsilon$, we have found the most probable initial state of LFSR1. We can then proceed in this fashion for each of the remaining registers, and thus reduce the complexity to merely $\sum_{i=1}^{4}(2^{l_i} - 1)$. □

In order to prevent Siegenthaler's divide-and-conquer attack, the combining function should have high correlation immunity. Assume that the Boolean function used

is correlation immune to the 1st order. Then we cannot separate the constituent LFSRs into single targets, but need to consider them pairwise, which increases the time complexity of the attack drastically.

Meier and Staffelbach refined the correlation attack in [82, 83] using a slightly different model, see Figure 2.10. The approach is known as a *fast correlation attack*. Assume that there is a correlation between one shift register (LFSR1) and the output

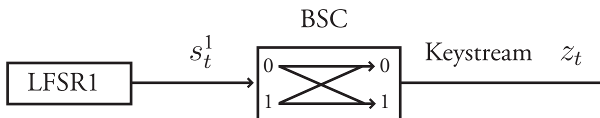


Figure 2.10: Model used by Meier and Staffelbach.

keystream z_t, such that $P(s_t^1 = z_t) = p = \frac{1}{2} + \epsilon,\ t \geq 0$. Meier and Staffelbach viewed this as if the sequence from LFSR1 was transmitted over a Binary Symmetric Channel (BSC) [24], with crossover probability $1 - p$, i.e. the BSC transmits the symbol correctly with a probability p. The combined effect of the other shift registers and the nonlinear combiner is modelled as the BSC. Since the feedback polynomial of LFSR1 is linear, each s_t for different t must satisfy a number of linear equations, based on how many taps the feedback polynomial has, and where the taps are located. If the correlation between s_t and z_t is high enough, most of the corresponding symbols in the keystream z_t must also fulfil these linear equations.

So, by attempting to slightly modify the sequence z_t to compensate for a possible crossover in the BSC model, Meier and Staffelbach could show that the sequence $\mathbf{s} = s_0^1, s_1^1, \ldots, s_N^1$ can be recovered and thus also the initial state of the shift register.

The drawback of this algorithm is that it is only successful if the feedback polynomial has very few taps. Several papers contributed minor improvements using the same technique, see [15, 88, 89, 95, 96].

Recently, the idea of a communication channel was reconsidered by Johansson and Jönsson in [65] where they identified an embedded convolutional code in the sequence $\mathbf{s}$ and could apply standard decoding techniques, e.g. the Viterbi algorithm, to recover the initial state even if the correlation probability was very close to 0.5. Typically, a shift register of length 40 with a correlation probability of 0.45 can be attacked with modest computational effort. This algorithm is independent of the number of taps of the feedback polynomial. Several papers providing improvements followed, see [64, 66, 87, 14, 13]. For a comprehensive discussion of fast correlation attacks we refer to [67].

Algebraic attacks

The algebraic attacks are a rather new approach to cryptanalysis even though the basic ideas have been known in mathematical literature for some time. The recent interest was initiated in a paper by Kipnis and Shamir [70], where they introduced a technique called *relinearisation*. The keystream generated by a keystream generator can in most cases be described by a complex system of multivariate polynomial equations, with the key bits being the indeterminants. The general solution of such a system is known to be NP-complete even in the simplest case of only quadratic equations over $\mathbb{F}_2$. The classical method for solving such systems is the algorithm by Buchberger using Gröbner bases[3]. The algorithm in [70] was designed to handle *overdefined* systems of $\epsilon \cdot n^2$ quadratic equations in n variables, where $\epsilon \leq 1/2$. Their method is successful in many cases but the criteria for success and the complexity of the approach are not well understood.

In [20], Courtois et al. gave a theoretical and practical analysis of the algorithm in [70] and extended the work with the XL-algorithm (eXtended Linearisations), based on bounded degree Gröbner bases and linearisation. The XL-algorithm handles higher degree multivariate systems of equations, but still, the criteria for success and the complexity dwell in darkness. Some authors do not accept the universality of the method and their main concern is that the number of *independent* linear equations after linearisation does not grow fast enough for the system to become solvable [90]. Nevertheless, a series of papers followed on the subject and the techniques have been successfully applied in breaking some stream ciphers [22, 20, 23, 21, 19].

Side channel attacks

Apart from the attacks discussed above, which are more direct attacks on the keystream generator algorithm, there exist others that try to attack a certain implementation of an algorithm. They are often called *side channel attacks* since these attacks utilise information leakage from other channels than the ciphertext or keystream output. Two examples of side channel attacks are *power analysis* and *timing* attacks.

The general idea in a power analysis attack is to measure the power usage of a cryptographical system, for example, when implemented on a smart card or other tamper-resistant package. Electromagnetic emissions can be used to mount similar attacks. This kind of attack has been shown to be surprisingly efficient on implementations of both block ciphers and various public-key ciphers.

In 1999, Kocher, Jaffe and Jun presented a paper on differential power analysis on DES [73]. They showed how the power usage of DES, in a certain implementation, reveals the structure of the cipher and small (6 bits) portions of the key can be guessed and verified independently. Their analysis can also locate conditional branches, for

[3] The name *Gröbner bases* was introduced by Bruno Buchberger in his Ph.D. thesis in honour of his advisor, Wolfgang Gröbner.

example, the difference between square and multiply[4] in modular exponentiation and thereby reveal information about the secret exponent in, for example, RSA.

In a timing attack, the attacker measures the execution time or the delay of various steps in the algorithm. This can reveal information on the secret key bits if they are evaluated for branching, where the branches have different execution time. A timing attack can also be applied to clock-controlled generators, that outputs keystream bits at irregular intervals. By measuring the intervals, the attacker can obtain information on the clocking sequence. Such weaknesses can be prevented by buffering the output, but still, the ideas are very relevant in an actual implementation of a cipher. Examples of timing attacks can be found in [26, 72]

2.7 Security of a stream cipher

In the design of a stream cipher, the goal is to make it act as similar to the One-Time Pad as possible, with the convenience of only using a limited number of bits for the key. The security of the OTP comes from the fact that the keystream sequence is drawn with uniform probability from the set of all possible sequences. Therefore, for the additive stream cipher, the generated keystream material should be as random looking as possible. An attacker should not be able to tell the difference between a truly random sequence and the sequence generated by the keystream generator. If such a difference could be measured, a distinguishing attack can be mounted. A distinguishing attack on an additive stream cipher could be very useful if the number of possible messages are limited. From the observed ciphertext and the guessed plaintext, the attacker can compute the keystream sequence and if that derived sequence has any statistical deviations from a truly random sequence, the attacker knows that the guessed plaintext was correct.

The generated keystream will always leak information about the key and will always have statistical deviations from a truly random sequence. Hence, if given a long enough keystream sequence, the attacker can most likely derive some useful information, but the aim for any stream cipher is to minimise that information leakage, so that the attacker cannot gain from using it, compared to exhaustively searching for the correct key.

The security of a stream cipher is thus always measured relative to the complexity of exhaustively searching for the correct key. If the complexity of an attack is less than that of the exhaustive search, the cipher is said to be *broken*.

This definition of a broken cipher is the one used by most cryptographers while discussing the theoretical aspects of cryptology. For a stream cipher in a practical application, the situation is much less clear. Consider for example a distinguishing attack on a stream cipher with a 256 bit key. It makes no sense talking about that cipher as (practically) broken if the required length of the guessed plaintext is say 2^{250}

[4] "Square-and-multiply" is a standard technique for fast exponentiation modulo an integer.

bits[5]. Apart from these insanely huge numbers (from a practical perspective), the actual security of a system using encryption is often much more dependent on other parts of the system than the cipher, e.g. protocols, users, key management/storage and implementation specific problems like software bugs [38].

However, the theoretical attacks are interesting and important for the development and understanding of the security of stream ciphers. Sometimes they reveal weaknesses that in the future can lead to practical attacks, which could concern us all. In this thesis, attacks of both theoretical and practical interest will be presented.

2.8 Summary

In this chapter we have introduced the ideas of stream ciphers and notably the additive stream cipher. We have seen that an additive stream cipher can be considered as a keystream generator, producing keystream symbols which are added (xored) to the plaintext to form the ciphertext. Some of the inner building blocks of stream ciphers were discussed, starting with the LFSR. We have recalled fundamental results on periodicity and introduced the linear complexity of a sequence, which is a measure of the minimal length of an LFSR that can generate the sequence. Next, we have introduced Boolean functions and their usefulness in destroying the inherent linearity in sequences produced by LFSRs. Also some properties of Boolean functions have been presented, including the nonlinearity and correlation immunity of a Boolean function. The ideas from Boolean functions were then extended to S-Boxes, which are vector outputs of several Boolean functions sharing the same input variables.

After the discussion on fundamental blocks used in design, we have presented some general design techniques; the nonlinear combination generator, the nonlinear filter generator, and clock-controlled generators. The last section on generic attacks on stream ciphers starts with the three most important complexity measures for an attack; time complexity, data complexity, and memory complexity. Continuing with the attacks, we have discussed different approaches to attacking a stream cipher and have given some examples on how to mount them in a general setting.

The chapter is concluded with a discussion of what it means for a stream cipher to be secure.

This completes the introduction to the subject of this thesis and in the following chapters we will discuss more specific aspects of cryptology.

[5] Current estimates of the number of electrons in the universe is roughly 2^{266} [76].

3

Cryptanalysis of SOBER-t16 and SOBER-t32

At a recent evaluation procedure in Europe for new cryptographical primitives called the NESSIE[1] project [92], two similar stream ciphers were submitted by Hawkes and Rose from Qualcomm Australia, called SOBER-t16 [56] and SOBER-t32 [57]. These are two LFSR based stream ciphers developed from previous versions of the stream cipher named SOBER.

The SOBER-t16 and SOBER-t32 stream ciphers consist of a keyed generator producing a pseudo-random sequence that is added to the plaintext. The generators can roughly be described as being nonlinear filter generators with an additional "stuttering" step before producing the output. The stuttering will irregularly decimate the keystream output sequence according to values in the keystream sequence.

It is known that as a result of this irregularity, one can potentially use a power analysis attack or a timing attack to recover the input to the stuttering step [106].

In this chapter we consider several ways of mounting distinguishing attacks on SOBER-t16 and SOBER-t32. The attacks are based on combining linear approximations of the nonlinear filter with the linear recurrence, defined through the feedback polynomial. Linear approximations have previously been used in e.g. the BAA attack on stream ciphers [28] and linear cryptanalysis on block ciphers [79] and have subsequently been used in the more recent linear masking technique [17]. In our case we derive the distribution of the noise introduced through the linear approximations from simulations. We consider attacks on SOBER-t16 both including and excluding the stuttering step, and an attack on SOBER-t32 excluding the stuttering step. The results presented in this chapter were originally presented in [32].

[1] New European Schemes for Signatures, Integrity and Encryption

The chapter is organised as follows. In Section 3.1 the stream ciphers SOBER-t16 and SOBER-t32 are described. Then in Section 3.2, we sidetrack slightly and recall some basic results regarding hypothesis testing, including the Neyman-Pearson lemma and the Chernoff information measure. Again focusing on the cipher at hand, we start by explaining the attack on SOBER-t16 without stuttering in Section 3.3. This is generalised to an attack on the full SOBER-t16 in Section 3.4. In Section 3.5 we describe a simple attack on SOBER-t32 without stuttering. In Section 3.6 we then present some recent related work on an attack on the full SOBER-t32. Finally, a summary is given in Section 3.7.

3.1 A description of SOBER-t16 and t32

Both SOBER-t16 and SOBER-t32 are word oriented stream ciphers. The symbol size is 16 bits for t16 and 32 bits for t32. The structures of t16 and t32 are very similar and we will here describe them as one cipher. The specific parameters for both t16 and t32 will be given as required. To simplify the description of the common parts of t16 and t32, we will use the notation W to denote the symbol size. Thus, W is either 16 or 32 bits, depending on which cipher we are considering. The operations in the ciphers include both addition in an extension field $\mathbb{F}_{2^W}$ and addition modulo 2^W.

Remark. Throughout this thesis, we will do addition in finite fields, finite rings and the infinite real field. The notation $\oplus$ will be used for the finite field addition, $\boxplus$ for the finite ring addition and $+$ for the real field addition. In the case where there is no risk of confusion, we will simply use $+$ for readability.

There are three main building blocks for the SOBER stream ciphers. The first is a word oriented LFSR that produces a sequence denoted $s_t,\ t \geq 0$. Secondly, a nonlinear filter (NLF) takes some of these symbols as inputs and produces a new sequence $v_t,\ t \geq 0$. Finally, there is a so-called stuttering unit. The stuttering unit takes $v_t,\ t \geq 0$, as input and produces an irregular output $z_n, n \geq 0$. The overall structure is pictured in Figure 3.1.

The LFSR

The LFSR is a length 17 shift register, where each register element contains one word of W bits. Each word is considered as an element in an extension field $\mathbb{F}_{2^W}$. The state of the LFSR at time t will be denoted by a vector $\mathbf{S}_t = (s_t, s_{t+1}, \ldots, s_{t+16})$. The next state of the LFSR is obtained by shifting the previous state one step, and calculating a new word s_{t+17}. The new word is calculated as a certain linear combination of the contents of the previous state. Thus the next state will be $\mathbf{S}_{t+1} =$

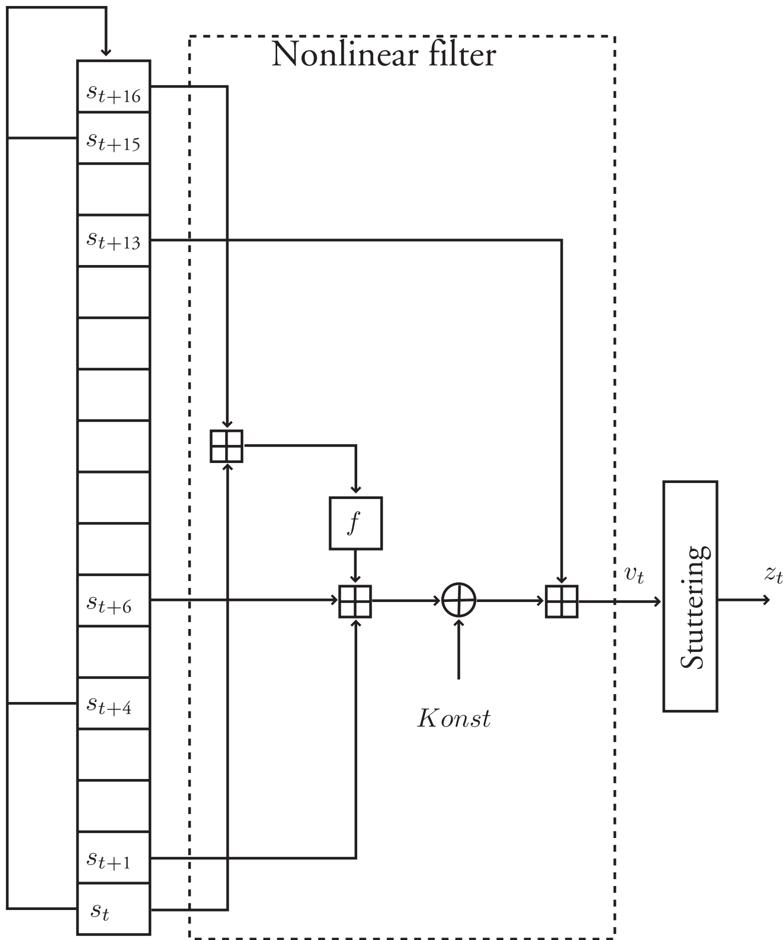


Figure 3.1: Overall structure of SOBER-t16 and SOBER-t32.

$(s_{t+1}, s_{t+2}, \ldots, s_{t+17})$ where

$$s_{t+17} = \sum_{i=0}^{16} c_i s_{t+i}, \tag{3.1}$$

for some known constants $c_i \in \mathbb{F}_{2^W}, i = 0, 1, \ldots, 16$. The arithmetic in (3.1) is performed in the extension field $\mathbb{F}_{2^W}$. The specific extension fields and recurrence equations for t16 and t32 are summarised as follows:

SOBER-t16

Defining polynomial for $\mathbb{F}_{2^{16}}$: $x^{16} + x^{14} + x^7 + x^6 + x^4 + x^2 + x + 1$
Linear recurrence equation: $s_{t+17} \oplus \alpha s_{t+15} \oplus s_{t+4} \oplus \beta s_t = 0,$
where $\alpha = 0xE382$ and $\beta = 0x67C3$.

SOBER-t32

Defining polynomial for $\mathbb{F}_{2^{32}}$: $x^{32} + (x^{24} + x^{16} + x^8 + 1)(x^6 + x^5 + x^2 + 1)$
Linear recurrence equation: $s_{t+17} \oplus s_{t+15} \oplus s_{t+4} \oplus \alpha s_t = 0,$
where $\alpha = 0xC2DB2AA3$.

The field elements α and β have been given in a hexadecimal form, corresponding to a polynomial basis. See [56, 57] for more details.

The NLF function

At time t, the NLF function takes five words $(s_t, s_{t+1}, s_{t+6}, s_{t+13}, s_{t+16})$ from the LFSR state and one constant value ($Konst \in \mathbb{F}_{2^W}$) as input, and produces, through a nonlinear function, an output word denoted by v_t. The value of $Konst$ is determined during the initialisation phase of the LFSR and is kept constant throughout the entire session. The operations involved in the NLF function are xor (denoted $\oplus$), addition modulo 2^W (denoted $\boxplus$) and application of an S-Box.

The output of the NLF function, v_t, at time t, can be written as

$$v_t = ((s_{t+1} \boxplus s_{t+6} \boxplus f(s_t \boxplus s_{t+16})) \oplus Konst) \boxplus s_{t+13}, \tag{3.2}$$

where $f(x)$ is a function, different for SOBER-t16 and SOBER-t32, and which in both cases involves an S-Box application. The interior design of the function f is pictured in Figure 3.2. At first, the input is partitioned into a high part containing the 8 most significant bits and a low part containing the remaining bits. The high part then addresses an S-Box with W bits of output. The 8 most significant bits are taken directly as the f-function output, whereas the least significant part of the S-Box output is first xored to the low part from the input, see Figure 3.2.

Stuttering

Before producing the running key, the output from the NLF is passed through a stuttering unit. The stuttering decimates the NLF output, thus making a correlation attack harder. The first output from the NLF, v_0, is taken as the first *stutter control word* (SCW). The SCW is divided into pairs of bits (called dibits). Starting with the least significant dibit, the stuttering is determined from the value of these dibits. Actions are taken according to the value of the dibit, as listed in Table 3.1. The constant

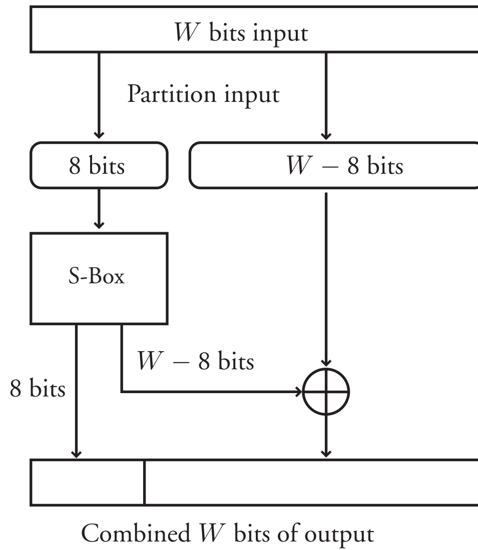


Figure 3.2: The structure of the f-function in SOBER-t16 and SOBER-t32.

C has value 0x6996 for t16 and 0x6996C53A for t32, and $\sim C$ denotes the bitwise complement of C.

Dibit		Action
00	1.	Clock the LFSR, but do not output anything.
01	1.	Clock the LFSR.
	2.	Set the value of the next keystream word to be the xor between C and the NLF output.
	3.	Clock the LFSR again, but do not output anything.
10	1.	Clock the LFSR, but do not output anything.
	2.	Clock the LFSR.
	3.	Set the value of the next keystream word to be the value of the NLF output.
11	1.	Clock the LFSR.
	2.	Set the value of the next keystream word to be the xor between $\sim C$ and the NLF output.

Table 3.1: The possible actions taken in the stuttering unit depending on the value of the dibit.

When all dibits in the SCW have been used, the LFSR is clocked once and a new SCW is read from the output of the NLF. This word determines the next 8 or 16 actions, depending on whether we are looking at SOBER-t16 or SOBER-t32. The resulting stream from the stuttering unit, denoted z_n, is the running key.

This concludes the brief description of SOBER-t16 and SOBER-t32. For a more detailed description, especially regarding the key initialisation, we refer to [56] and [57]. Before we present the attacks, we will recall some basic tools in hypothesis testing.

3.2 Hypothesis testing—the short story

A frequently occurring problem in cryptanalysis is the determination of whether a sequence of observations is more likely to be sampled from a device having output distribution P_0, or from a device having output distribution P_1. This problem arises in many other areas as well (e.g. radar technology and automatic medical surveillance) and the tools and methods are well known. A thorough introduction to the subject can be found in [24, 81, 117].

Here we will confine ourselves to a discussion of the following three issues:

- The form of the optimum test.
- The probability of making a wrong decision.
- The number of samples needed in order to obtain a certain level of confidence in the decision.

Assume that we have a sequence of n independent and identically distributed (*i.i.d.*) random variables $X_1, X_2, \ldots, X_n$ over an alphabet $\aleph$. The distribution is denoted $Q(x) = Pr(X_i = x)$, $1 \leq i \leq n$ and the sampled values are denoted $\mathbf{x} = x_1, x_2, \ldots, x_n$, where $x_i \in \aleph$, $1 \leq i \leq n$. We consider two hypotheses:

- $H_0 : Q = P_0$.
- $H_1 : Q = P_1$.

Let $\phi(\mathbf{x})$ be a decision function where $\phi(\mathbf{x}) = 0$ implies that H_0 is accepted and $\phi(\mathbf{x}) = 1$ implies that H_1 is accepted. Furthermore, let $P_0^n(\mathbf{x})$ denote the simultaneous probability $\prod_{i=1}^n P_0(x_i)$, and similarly we have $P_1^n(\mathbf{x}) = \prod_{i=1}^n P_1(x_i)$. Since $\phi(\mathbf{x})$ only takes two possible values, we can specify a set $A \in \{\aleph\}^n$, over which $\phi(A) = 0$ and the complementary set $A^* \in \{\aleph\}^n$, over which $\phi(A^*) = 1$.

We can now specify the two types of error that can occur:

$$P_F = Pr(\phi(\mathbf{x} = 1) | H_0 \text{ is true }) = P_0^n(A^*), \tag{3.3}$$

$$P_M = Pr(\phi(\mathbf{x} = 0) | H_1 \text{ is true }) = P_1^n(A). \tag{3.4}$$

Remark. The notation P_F and P_M comes from old radar terminology [117]. P_F means *probability of a false alarm* (i.e. a blip on the radar screen just being noise) and P_M means *probability of a miss* (i.e. there is a target but the radar did not detect it).

Ideally, we would like to minimise both P_F and P_M but normally there is a trade-off. Sometimes the implications of a miss are more severe than a false alarm and the probabilities are not as equally important to minimise. An example could be a cardiac monitoring system at a hospital; an alarm signifying a cardiac arrest cannot be missed but a rare false alarm is probably acceptable.

The optimum test between the two hypotheses is given by the *Neyman-Pearson lemma*, given here without a proof.

Lemma 3.1 (Neyman-Pearson [24]): *Let $X_1, X_2, \ldots, X_n$ be drawn i.i.d. according to the mass function Q. Consider the decision problem corresponding to the hypotheses $Q = P_0$ vs. $Q = P_1$. For $T \geq 0$ define a region*

$$A_n(T) = \left\{ \frac{P_0(x_1, x_2, \ldots, x_n)}{P_1(x_1, x_2, \ldots, x_n)} > T \right\}.$$

Let $P_F = P_0^n(A_n^*(T))$ and $P_M = P_1^n(A_n(T))$ be the probabilities of error corresponding to the decision region $A_n(T)$. Let B_n be any other decision region with associated probabilities of error P_F^B and P_M^B. If $P_F^B \leq P_F$ then $P_M^B \geq P_M$. □

The Neyman-Pearson lemma tells us that the region $A_n(T)$, determined by the *likelihood ratio* $\frac{P_0(\mathbf{x})}{P_1(\mathbf{x})} \geq T$, is the one that (jointly) minimises P_F and P_M.

If we have symmetrical distributions of equal shape and would like to have the probabilities of error P_F and P_M equally large, we should choose $T = 1$. When computing the likelihood ratio for a large sample, both the numerator and the denominator tend to become very small and if a computer is used we could run into serious numerical problems. Recalling that X_i, $1 \leq i \leq n$ are assumed to be independent, we can rewrite the test using a 2-logarithmic measure and $T = 1$ as

$$\begin{aligned}
\frac{P_0(x_1, x_2, \ldots, x_n)}{P_1(x_1, x_2, \ldots, x_n)} &> 1, \\
\frac{\prod_{i=1}^n P_0(x_i)}{\prod_{i=1}^n P_1(x_i)} &> 1, \\
\log_2 \left(\frac{\prod_{i=1}^n P_0(x_i)}{\prod_{i=1}^n P_1(x_i)} \right) &> 0, \\
\sum_{i=1}^n \left(\log_2 \frac{P_0(x_i)}{P_1(x_i)} \right) &> 0. \qquad (3.5)
\end{aligned}$$

In (3.5) we have a simple, computationally robust test, which is easy to implement and tells us which of the two hypotheses H_0 or H_1 is the most likely. The ratio is called a *log-likelihood ratio*, and the test is called a *log-likelihood test*.

The Neyman-Pearson lemma tells us how to carry out the actual test given a sample of size n. In addition, (3.3) and (3.4) state the two probabilities of error. We could also try to estimate the number of samples that we need in order to achieve a certain level of confidence in the test, i.e. how large n must be so that the overall probability of error is smaller than a given value.

Assigning *a priori* probabilities to the two hypotheses, we can write the overall probability of error as

$$P_e = \pi_0 P_F + \pi_1 P_M, \tag{3.6}$$

where π_0 is the prior probability of H_0 and π_1 is the prior probability of H_1 and $\pi_0 + \pi_1 = 1$. It can then be shown [24] that P_e is essentially equal to the larger of P_F and P_M and the total error is given by

$$P_e = 2^{-nC(P_0,P_1)}, \tag{3.7}$$

where $C(P_0, P_1)$ is the *Chernoff information*

$$C(P_0, P_1) = -\min_{0\leq\lambda\leq 1} \log_2 \left(\sum_{x\in\aleph} (P_0(x))^{\lambda} (P_1(x))^{1-\lambda} \right), \tag{3.8}$$

and n is the number of samples. The exact value of λ in (3.8) can be difficult to obtain, but using e.g. $\lambda = 0.5$ we get an upper bound in (3.7) on the probability of error and that is sufficient in many cases.

So, firstly we need to calculate the Chernoff information between P_0 and P_1 using (3.8). Then we settle for an appropriate error probability P_e and from (3.7) we can derive the required number of samples n needed.

Example 3.2: Consider the problem of determining if a random variable X is drawn from a distribution $P_\varepsilon(X = 0) = \frac{1}{2} + \varepsilon$ or if X is uniform with $P_U(X = 0) = \frac{1}{2}$. Calculating the Chernoff information between P_ε and P_U using $\lambda = 0.5$ we get

$$\begin{aligned} C(P_\varepsilon, P_U) &\geq -\log_2 \left(\sqrt{\frac{1}{2}}\sqrt{\frac{1}{2}+\varepsilon} + \sqrt{\frac{1}{2}}\sqrt{\frac{1}{2}-\varepsilon} \right) \\ &= -\log_2 \left(\frac{1}{2}\sqrt{1+2\varepsilon} + \frac{1}{2}\sqrt{1-2\varepsilon} \right). \end{aligned}$$

Using the Taylor expansion of $\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8}$ for small x we get

$$\begin{aligned} C(P_\varepsilon, P_U) &\approx -\log_2 \left(\frac{1}{2}\left(1 + \frac{2\varepsilon}{2} - \frac{4\varepsilon^2}{8}\right) + \frac{1}{2}\left(1 - \frac{2\varepsilon}{2} - \frac{4\varepsilon^2}{8}\right) \right) \\ &= -\log_2 \left(1 - \frac{\varepsilon^2}{2}\right) \approx \frac{\varepsilon^2}{2\ln(2)}, \end{aligned}$$

where in the last approximation we have used the Taylor expansion of $\log_2(1+x) = x/\ln(2)$ for small x. Thus, using $P_e = 0.5$ we need about $n = 1.4/\varepsilon^2$ samples to separate the two distributions. □

In Example 3.2 we have derived a well-known "rule of thumb", namely; To separate the two binary distributions $\frac{1}{2}+\varepsilon$ and $\frac{1}{2}$ we need approximately $1/\varepsilon^2$ samples.

We have reviewed some of the fundamentals of hypothesis testing, and will now return to the analysis of SOBER-t16 and SOBER-t32.

3.3 A distinguishing attack on SOBER-t16 without stuttering

We start by analysing a version of SOBER-t16 where the stuttering unit has been removed. In this scenario each NLF output word is taken as a running key word. Thus we have $z_t = v_t$ for all $t \geq 0$. We also assume that we are given N words of the output key stream, so we have access to $z_0, z_1, \ldots, z_{N-1}$.

The first step in our attack is to approximate the NLF-function with a linear function and then argue that the noise introduced by the approximation possesses a nonuniform distribution. Recall the expression for the NLF output

$$v_t = ((s_{t+1} \boxplus s_{t+6} \boxplus f(s_t \boxplus s_{t+16})) \oplus Konst) \boxplus s_{t+13}. \tag{3.9}$$

We now approximate this function with a linear function by replacing $\boxplus$ with $\oplus$, and f by the identity map. When we do this approximation we introduce a noise (an error), which is denoted by w_t. We also incorporate the value of $Konst$ into the noise w_t. Then

$$v_t = s_{t+1} \oplus s_{t+6} \oplus s_t \oplus s_{t+16} \oplus s_{t+13} \oplus w_t, \tag{3.10}$$

where $w_t, t \geq 0$ denotes a random variable that represents the error we get in the approximation at each time t. The distribution of w_t will be dependent on the value $Konst$. However, w_t will have the same distribution for all t, and this distribution is denoted Ψ.

Introduce the notation $\Omega_t = s_t \oplus s_{t+1} \oplus s_{t+6} \oplus s_{t+13} \oplus s_{t+16}$ for the xor of the words from the LFSR that are inputs to the NLF function. We can then write the output word v_t as

$$v_t = \Omega_t \oplus w_t. \tag{3.11}$$

By looking at the running key at times $t, t+4, t+15$ and $t+17$ in combination with (3.11) we can express $z_{t+17} \oplus \alpha z_{t+15} \oplus z_{t+4} \oplus \beta z_t$ in the following way

$$\begin{aligned} z_{t+17} \oplus \alpha z_{t+15} \oplus z_{t+4} \oplus \beta z_t &= v_{t+17} \oplus \alpha v_{t+15} \oplus v_{t+4} \oplus \beta v_t \\ &= (\Omega_{t+17} \oplus w_{t+17}) \oplus \alpha(\Omega_{t+15} \oplus w_{t+15}) \oplus (\Omega_{t+4} \oplus w_{t+4}) \oplus \beta(\Omega_t \oplus w_t). \end{aligned} \tag{3.12}$$

Rearranging the terms on the right hand side of (3.12) we get

$$z_{t+17} \oplus \alpha z_{t+15} \oplus z_{t+4} \oplus \beta z_t = \Omega_{t+17} \oplus \alpha\Omega_{t+15} \oplus \Omega_{t+4} \oplus \beta\Omega_t \oplus w_{t+17} \oplus \alpha w_{t+15} \oplus w_{t+4} \oplus \beta w_t. \quad (3.13)$$

Then, recalling the linear recurrence relation for SOBER-t16

$$s_{t+17} \oplus \alpha s_{t+15} \oplus s_{t+4} \oplus \beta s_t = 0, \quad (3.14)$$

we see that $\Omega_{t+17} \oplus \alpha\Omega_{t+15} \oplus \Omega_{t+4} \oplus \beta\Omega_t = 0$ and we can reduce (3.12) to

$$z_{t+17} \oplus \alpha z_{t+15} \oplus z_{t+4} \oplus \beta z_t = w_{t+17} \oplus \alpha w_{t+15} \oplus w_{t+4} \oplus \beta w_t, \quad (3.15)$$

where the multiplications with α and β are in the extension field $\mathbb{F}_{2^W}$.

Estimating the distribution of w_t

The noise variables $w_t, t \geq 0$ are random variables taken from $\mathbb{F}_{2^{16}}$ with a nonuniform but unknown distribution Ψ. Let us write the distribution Ψ in the form

$$\Psi = \begin{bmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{2^{16}-1} \end{bmatrix},$$

where $Pr(w_t = x) = \psi_x$. We cannot hope to find a closed expression for the distribution Ψ, since it is computationally too complex to derive. However, we can run the cipher and estimate the distribution Ψ.
In the simulations, we measure the frequency of different values for the noise w_t, calculated as

$$w_t = (((s_{t+1} \boxplus s_{t+6} \boxplus f(s_t \boxplus s_{t+16})) \oplus Konst) \boxplus s_{t+13}) \oplus \Omega_t. \quad (3.16)$$

Assume that we sample 2^ν values of w_t according to (3.16) by running the cipher, and denote the measured frequencies by the vector $\hat{\Psi} = (\hat{\psi}_0, \hat{\psi}_1, \ldots, \hat{\psi}_{2^{16}-1})$. $\hat{\Psi}$ is an estimation of Ψ and we can write $\Psi = \hat{\Psi} + \mathbf{E}$, where $\mathbf{E}$ is a vector representing the error in the estimation. Focusing on one single component of $\mathbf{E}$, it will be approximately Gaussian distributed with zero mean and a standard deviation of $2^{-(\nu/2+8)}$. Simulations show that Ψ is quite nonuniform. For example, in simulation with $Konst = 0$ and $\nu = 38$, the maximum value is $2^{-16} + 2^{-17.6}$. The error in this estimation is of order 2^{-28}. The distribution Ψ has been tabulated for a number of different values of $Konst$.

Calculating the full noise distribution

Let us define

$$w_t^F = w_{t+17} \oplus \alpha w_{t+15} \oplus w_{t+4} \oplus \beta w_t. \tag{3.17}$$

This means that w_t^F, $t \geq 0$ are the random variables corresponding to the full noise (superscript F for *Full*) that we can sample from the running key. Let the distribution of w_t^F be Ψ^F with components

$$\Psi^F = \begin{bmatrix} \psi_0^F \\ \psi_1^F \\ \vdots \\ \psi_{2^{16}-1}^F \end{bmatrix}.$$

That is, we have $Pr(w_t^F = x) = \psi_x^F$.

In (3.17) we see that we must combine four Ψ distributions to get the overall noise distribution, Ψ^F, and it is assumed that $w_t, t \geq 0$, are independent variables. Furthermore, we make the distinct simplification that w_t^F, $t \geq 0$, are also independent variables.

The distribution $\Psi' = [\psi_i']$ of the xor of two random variables with distribution $\Psi^1 = [\psi_i^1]$ and $\Psi^2 = [\psi_i^2]$ respectively, is obtained by

$$\psi_l' = \sum_{i \oplus j = l} \psi_i^1 \psi_j^2. \tag{3.18}$$

The distribution of αw_t is simply a permutation of the distribution Ψ. It can be shown [4] that when combining distributions as done in (3.18), we sustain significance in the resulting distribution. So by estimating the Ψ distribution by simulation and then combining the probabilities according to (3.18), we can estimate the distribution Ψ^F, of the right hand side of (3.15) for different values of $Konst$.

To be able to distinguish the full noise distribution, Ψ^F, from the uniform distribution (denoted P_U) we need have some N different keystream symbols. Let $z_t^* = z_{t+17} \oplus \alpha z_{t+15} \oplus z_{t+4} \oplus \beta z_t$, $t = 0, \ldots, N-1$. From (3.5) we know that we need to test the log-likelihood ratio

$$\sum_{t=0}^{N-1} \log \left(\psi_{z_t^*}^F / 2^{-16} \right) > 0. \tag{3.19}$$

Then from (3.7) we have the relationship between the probability of an incorrect decision, denoted P_e, the number of required samples N, and the Chernoff information. We fix the probability of error to $P_e = 2^{-32}$ and we need to choose $N \geq 32C(P_1, P_2)^{-1}$.

Summarising the results

The distribution Ψ^F has been determined through simulation as previously described. We summarise the results given in this section in the following attack.

Set $\hat{f}_i = 0,\ i = 0 \ldots 2^{16} - 1$.
For $t = 1 \ldots N$ do

(*i*) Calculate $z_t^* = z_{t+17} \oplus \alpha z_{t+15} \oplus z_{t+4} \oplus \beta z_t$.

(*ii*) Let $\hat{f}_{z_t^*} = \hat{f}_{z_t^*} + 1$.

end for.

Calculate $I = \sum_{x \in \mathbb{F}_{2^{16}}} \hat{f}_x \log_2 \left[\frac{\psi_x^F}{2^{-16}} \right]$.

If $I > 0$ then output **SOBER** otherwise output **random**

We have calculated the combined Ψ^F distribution using $2^{38} (\nu = 38)$ sampled outputs to generate the Ψ distribution. Note that since Ψ (and thus Ψ^F) is dependent on the unknown value of $Konst$, we actually need to determine the Ψ^F distribution for all 2^{16} possible values of $Konst$.

The resulting Chernoff information between Ψ^F and the uniform distribution has been derived for 50 random values of $Konst$, and were all between 2^{-84} and 2^{-87}. We assume that calculating the Chernoff information for other values of $Konst$ will give similar results. In the worst case, we need at least $N = 32 \cdot 2^{87} = 2^{92}$ words from the running key to be able to distinguish a SOBER-t16 output sequence without stuttering from a uniform distribution with a probability of error $P_e = 2^{-32}$. The computational complexity of the attack is roughly 2^{92}.

Finally, the Neyman-Pearson test must be performed for each of the 2^{16} possible values of $Konst$. Still, the probability of error is smaller than 2^{-16}, which is small enough. Note that this step does not change the overall complexity.

3.4 A distinguishing attack on SOBER-t16 with stuttering

When the stuttering unit is present, not every NLF output is used to produce a keystream symbol. Recalling the functionality of the stuttering unit, we see that each v_t can be either discarded, used as a new SCW, or used (possibly xored with a constant) as a keystream symbol z_n. To be able to use the results from Section 3.3, we must have access to the NLF output quadruple $(v_t, v_{t+4}, v_{t+15}, v_{t+17})$.

Assume that we look at one output symbol $z_n = \mathcal{C}_0 \oplus v_t$, where $\mathcal{C}_0 \in \{0, C, \sim C\}$ is the constant that is xored to v_t in the stuttering unit to form z_n. Simulations show that the most probable position in the keystream for v_{t+4} to appear in is z_{n+2}. Similar simulations to determine the most probable position for v_{t+15} and v_{t+17} give the following results

$$\begin{aligned} Pr(\mathcal{C}_1 \oplus v_{t+4} \rightarrow z_{n+2} | \mathcal{C}_0 \oplus v_t \rightarrow z_n) &= 0.31, \\ Pr(\mathcal{C}_2 \oplus v_{t+15} \rightarrow z_{n+7} | \mathcal{C}_1 \oplus v_{t+4} \rightarrow z_{n+2}) &= 0.19, \\ Pr(\mathcal{C}_3 \oplus v_{t+17} \rightarrow z_{n+8} | \mathcal{C}_2 \oplus v_{t+15} \rightarrow z_{n+7}) &= 0.40. \end{aligned}$$

Having established the most probable positions in the keystream for $(v_{t+4}, v_{t+15}, v_{t+17})$, given an output $z_n = \mathcal{C}_0 \oplus v_t$, we are still faced with the problem of determining which constants $\mathcal{C}_i, i = 0, 1, 2, 3$ each NLF output is xored with.

Denote by $\mathcal{E}$ the event that, given $\mathcal{C}_0 \oplus v_t \rightarrow z_n$, we have $\mathcal{C}_1 \oplus v_{t+4} \rightarrow z_{n+2}$, $\mathcal{C}_2 \oplus v_{t+15} \rightarrow z_{n+7}$ and $\mathcal{C}_3 \oplus v_{t+17} \rightarrow z_{n+8}$. The probability of event $\mathcal{E}$, denoted p_0, is $p_0 \approx 2^{-5.5}$.

By looking at Table 3.1 we note that certain combinations of $(\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3)$ cannot occur under the assumption $\mathcal{E}$. In general, the distribution is nonuniform and for example, the five combinations of $(\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3)$ given by

$$\begin{aligned} &(0, 0, 0, 0), \\ &(C, C, C, \sim C), \\ &(C, C, \sim C, 0), \\ &(C, \sim C, 0, 0), \\ &(\sim C, 0, 0, 0), \end{aligned}$$

are more likely to occur than others.

From (3.12) and (3.15) in Section 3.3, we know that

$$v_{t+17} \oplus \alpha v_{t+15} \oplus v_{t+4} \oplus \beta v_t = w_{t+17} \oplus \alpha w_{t+15} \oplus w_{t+4} \oplus \beta w_t. \qquad (3.20)$$

Given event $\mathcal{E}$, we can write

$$z_{n+8} \oplus \alpha z_{n+7} \oplus z_{n+2} \oplus \beta z_n = w_t^F \oplus \mathcal{C}_3 \oplus \alpha \mathcal{C}_2 \oplus \mathcal{C}_1 \oplus \beta \mathcal{C}_0, \qquad (3.21)$$

where $w_t^F = w_{t+17} \oplus \alpha w_{t+15} \oplus w_{t+4} \oplus \beta w_t$ and has a known distribution, Ψ^F.

Again, we derive the distribution of the right hand side of (3.21) and denote this distribution by Ψ^{FC} (superscript FC for *Full with Constants*). Assuming $w_t^{FC} = w_t^F \oplus \mathcal{C}_3 \oplus \alpha \mathcal{C}_2 \oplus \mathcal{C}_1 \oplus \beta \mathcal{C}_0$, we can derive Ψ^{FC} from Ψ^F by considering all possible combinations of $(\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3)$ and their respective probability. The Chernoff information between Ψ^{FC} and the uniform distribution P_U is then calculated to be $C(\Psi^{FC}, P_U) \approx 2^{-95}$.

Sampling the keystream output sequence at $(z_n, z_{n+2}, z_{n+7}, z_{n+8})$ will give us a sample of the noise from the distribution Ψ^{FC} with probability $p_0 = 2^{-5.5}$. With probability $(1 - p_0)$ the assumption was wrong and it is reasonable to assume that we then get a uniform distribution. Write the distribution Ψ^{FC} as a vector

$$\Psi^{FC} = \begin{bmatrix} 2^{-16} + \xi_0 \\ 2^{-16} + \xi_1 \\ \vdots \\ 2^{-16} + \xi_{2^{16}-1} \end{bmatrix}, \tag{3.22}$$

where each element $2^{-16} + \xi_x$ represents $Pr(w_t^{FC} = x) = 2^{-16} + \xi_x$.

Let $z_n^* = z_{n+8} \oplus \alpha z_{n+7} \oplus z_{n+2} \oplus \beta z_n$. The distribution of z_n^*, $n \geq 0$, denoted P_{Z^*}, can then be calculated to be

$$P_{Z^*} = \begin{bmatrix} 2^{-16} + \xi_0 p_0 \\ 2^{-16} + \xi_1 p_0 \\ \vdots \\ 2^{-16} + \xi_{2^{16}-1} p_0 \end{bmatrix}, \tag{3.23}$$

i.e. we have $Pr(Z^* = z^*) = P_{Z^*}(z^*)$. The resulting Chernoff information between P_{Z^*} and the uniform distribution P_U can be derived from the $C(\Psi^{FC}, P_U)$ calculated in Section 3.3. We have $P_U(x) = 2^{-16}$, $0 \leq x \leq (2^{-16} - 1)$, and calculating a lower bound on the Chernoff information by using $\lambda = 0.5$ (see (3.8)) we get

$$\begin{aligned} C(P_{Z^*}, P_U) &= -\log_2 \sum_{i=0}^{2^{16}-1} \sqrt{2^{-16} + \xi_i p_0} \sqrt{2^{-16}} \\ &= -\log_2 \sum_i 2^{-16} \sqrt{1 + \frac{\xi_i p_0}{2^{-16}}}. \end{aligned} \tag{3.24}$$

By the Taylor expansion of (3.24), similarly to the calculation in Example 3.2, it follows that $C(P_{Z^*}, P_U) \approx p_0^2 C(\Psi^{FC}, P_U)$ and hence, from the result in Section 3.3 for $C(\Psi^{FC}, P_U)$, we conclude that we need at most $N = 32 \cdot p_0^{-2} 2^{95} \approx 2^{111}$ keystream symbols to be able to distinguish the output of SOBER-t16 with stuttering from a uniform source. The complexity is of the same size. We summarise the results given in this section in the following attack.

Let $\hat{f}_i = 0,\ i = 0\ldots 2^{16}-1$.
For $t = 1\ldots N$ do

(i) Calculate $z_n^* = z_{n+8} \oplus \alpha z_{n+7} \oplus z_{n+2} \oplus \beta z_n$.

(ii) Let $\hat{f}_{z_n^*} = \hat{f}_{z_n^*} + 1$.

end for.

Calculate $I = \sum_{x\in\mathbb{F}_{2^{16}}} \hat{f}_x \log_2 \left[\frac{P_{Z^*}(x)}{2^{-16}}\right]$.

If $I > 0$, then output **SOBER** otherwise output **random**

Again, we should note that P_{Z^*} is dependent on $Konst$, and a full attack includes testing against 2^{16} different distributions.

3.5 A distinguishing attack on SOBER-t32 without stuttering

The attack on SOBER-t16 was possible since we could compute the noise distribution Ψ by simulation. From Ψ we could derive Ψ^F.

Obtaining significance in simulation was possible due to the small word size of 16 bits. In SOBER-t32 we cannot directly use the same method to obtain a similar distribution Ψ, due to computational limitations. We note, however, that *if* we could simulate and find a noise distribution, then the attack on t32 would probably be strong. This is due to the fact that the linear recurrence relation in t32 has only one constant not equal to one, α, whereas t16 has two, α and β. The multiplications by these constants tend to smooth out the distribution.

In this section we present another attack, based on a bitwise linear approximation through the NLF function. Using the same notation as before, we denote the xor of the input words to the NLF at time t, by $\Omega_t = s_t \oplus s_{t+1} \oplus s_{t+6} \oplus s_{t+13} \oplus s_{t+16}$. The output from the NLF at time t, is denoted v_t. Since the stuttering unit is removed, we have $z_t = v_t$ for all $t \geq 0$. Each word is 32 bits and we will denote a specific bit i, $0 \leq i \leq 31$, in a word x, with $x[i]$. Let k denote the value of $Konst$.

We start by considering the linear recurrence relation of t32 given by

$$s_{t+17} \oplus s_{t+15} \oplus s_{t+4} \oplus \alpha s_t = 0, \tag{3.25}$$

and the corresponding characteristic polynomial for the recurrence

$$x^{17} + x^{15} + x^4 + \alpha. \tag{3.26}$$

Repeated squaring of this polynomial will still yield a valid linear recurrence equation for the considered linear recurrence of t32. Specifically, exponentiation with 2^{32} gives

$$x^{17\cdot 2^{32}} + x^{15\cdot 2^{32}} + x^{4\cdot 2^{32}} + \alpha^{2^{32}}. \tag{3.27}$$

Since $\alpha \in \mathbb{F}_{2^{32}}$ we have $\alpha^{2^{32}} = \alpha$ and addition of (3.26) and (3.27) gives

$$x^{17} + x^{15} + x^4 + x^{17\cdot 2^{32}} + x^{15\cdot 2^{32}} + x^{4\cdot 2^{32}}. \tag{3.28}$$

Here we can divide with x^4, and the resulting linear recurrence is given by

$$s_{t+17\cdot 2^{32}-4} \oplus s_{t+15\cdot 2^{32}-4} \oplus s_{t+4\cdot 2^{32}-4} \oplus s_{t+13} \oplus s_{t+11} \oplus s_t = 0,$$

which is written

$$s_{t+\tau_5} \oplus s_{t+\tau_4} \oplus s_{t+\tau_3} \oplus s_{t+\tau_2} \oplus s_{t+\tau_1} \oplus s_t = 0, \tag{3.29}$$

by introducing the constants $\tau_1 = 11$, $\tau_2 = 13$, $\tau_3 = 4\cdot 2^{32} - 4$, $\tau_4 = 15\cdot 2^{32} - 4$ and $\tau_5 = 7\cdot 2^{32} - 4$. Note that in (3.29) we have derived a linear recurrence equation that holds *for each single bit position.*

Consider the xor between two adjacent bits, i and $i-1$, $i \geq 1$, in the running key z_t. As before, we use a linear approximation of the NLF function, $z_t = \Omega_t \oplus w_t$, where the value of $Konst$ is incorporated into the binary random variable w_t representing the noise. We can now write

$$z_t[i] \oplus z_t[i-1] = \Omega_t[i] \oplus \Omega_t[i-1] \oplus w_t[i] \oplus w_t[i-1], \tag{3.30}$$

where $w_t[i]$ denotes the noise in bit position i introduced by the linear approximation. Let $\Psi^B[i]$ (superscript B for bit) be the distribution of $w_t[i] \oplus w_t[i-1]$. We can estimate the distribution $\Psi^B[i]$ by simulation and the result shows that the distribution is quite nonuniform for many positions of $1 \leq i \leq 31$. We can write the correlation between the xor of bit i and $i-1$ of the input and output as

$$\begin{aligned} Pr(z_t[i] \oplus z_t[i-1] = \Omega_t[i] \oplus \Omega_t[i-1]) &= \\ Pr(\Psi^B[i] = 0) &= \frac{1}{2} + \varepsilon_i, \end{aligned} \tag{3.31}$$

for each bit position $1 < i \leq 31$.

The largest correlation we have found is for the xor of bit 29 and bit 30 (i.e. $\Psi^B[30]$) in the input and output words. Simulations with 2^{30} samples for 100 random values of k, indicate that the correlation in (3.31) for $i = 30$ is only dependent on the two corresponding bits in k, i.e. $k[30]$ and $k[29]$. We have the following result

$$\varepsilon_{30} \approx \begin{cases} -0.0086 & \text{if } k[30] = 0 \text{ and } k[29] = 0, \\ -0.0052 & \text{if } k[30] = 1 \text{ and } k[29] = 1, \\ +0.0086 & \text{if } k[30] = 1 \text{ and } k[29] = 0, \\ +0.0052 & \text{if } k[30] = 0 \text{ and } k[29] = 1. \end{cases} \tag{3.32}$$

Now, given a keystream output, $z_0, z_1, \ldots, z_{N-1}$, of length N, we can use the linear recurrence relation (3.29) to calculate

$$\begin{aligned} z_{t+\tau_5} \oplus z_{t+\tau_4} \oplus z_{t+\tau_3} \oplus z_{t+\tau_2} \oplus z_{t+\tau_1} \oplus z_t = \\ \Omega_{t+\tau_5} \oplus w_{t+\tau_5} \oplus \Omega_{t+\tau_4} \oplus w_{t+\tau_4} \oplus \Omega_{t+\tau_3} \oplus w_{t+\tau_3} \oplus \\ \Omega_{t+\tau_2} \oplus w_{t+\tau_2} \oplus \Omega_{t+\tau_1} \oplus w_{t+\tau_1} \oplus \Omega_t \oplus w_t \end{aligned} \tag{3.33}$$

where the sum of all the Ω_j terms will equal zero as a result of (3.29). Thus, we have

$$z_{t+\tau_5} \oplus z_{t+\tau_4} \oplus z_{t+\tau_3} \oplus z_{t+\tau_2} \oplus z_{t+\tau_1} \oplus z_t = \bigoplus_{j=0}^{5} w_{t+\tau_j}. \tag{3.34}$$

Introduce the notation $z_t^* = z_{t+\tau_5} \oplus z_{t+\tau_4} \oplus z_{t+\tau_3} \oplus z_{t+\tau_2} \oplus z_{t+\tau_1} \oplus z_t$ for the left hand side of (3.34), and $w_t^F = \bigoplus_{j=0}^{5} w_{t+\tau_j}$ for the right hand side. We can calculate the probability that

$$\begin{aligned} Pr(z_t^*[i] \oplus z_t^*[i-1] = 0) = \\ Pr(w_t^F[i] \oplus w_t^F[i-1] = 0) = \frac{1}{2} + 2^5 \varepsilon_i^6, \end{aligned} \tag{3.35}$$

where the last equality comes from combining the six independent noise distributions Ψ^B, each with probability $1/2 + \varepsilon_i$ of being zero.

Recalling the measured correlation for bits $29 \oplus 30$ from (3.32), we see that ε_{30} takes four possible values. If we want to distinguish the distribution of w from a uniform source, the worst case occurs when ε_{30} takes the smallest value. Thus, using $\varepsilon_{30} = 0.0052$ and combining the six noise distributions according to (3.35) we derive the final correlation probability for the six independent keystream positions as

$$p_0 = Pr(z_t^*[i] \oplus z_t^*[i-1] = 0) = \frac{1}{2} + 2^5(0.0052)^6 \approx \frac{1}{2} + 2^{-40.5}. \tag{3.36}$$

Summarising the results

To be able to distinguish this nonuniform distribution, denoted P_0, from a uniform source, denoted P_U, we again calculate the Chernoff information between the two distributions as

$$C(P_0, P_U) = -\min_{0 \leq \lambda \leq 1} \log_2 \sum_x P_0^{\lambda}(x) P_U^{1-\lambda}(x) \approx 2^{-81.5}. \tag{3.37}$$

Settling for an error probability of $P_e = 2^{-32}$ we see that we need $N = 2^{86.5}$ samples from the keystream. Each sample spans a distance of $\tau_5 = 17 \cdot 2^{32} - 4 \approx 2^{36}$ positions, so in total, we need $N + \tau_5 \leq 2^{87}$ keystream output words to distinguish an output sequence from SOBER-t32 without stuttering unit from a uniform source. We summarise the results given in this section in the following attack.

Let $\hat{f} = 0$.
For $t = 1 \ldots N$ do

(*i*) Calculate $z_t^* = z_{t+\tau_5} \oplus z_{t+\tau_4} \oplus z_{t+\tau_3} \oplus z_{t+\tau_2} \oplus z_{t+\tau_1} \oplus z_t$.

(*ii*) Let $\hat{f} = \hat{f} + (1 - (z_t^*[i] \oplus z_t^*[i-1]))$.

end for.
Calculate $I = \hat{f} \log_2 \left[\frac{\frac{1}{2}+2^{-40.5}}{1/2}\right] + (N - \hat{f}) \log_2 \left[\frac{\frac{1}{2}-2^{-40.5}}{1/2}\right]$.
If $I > 0$, then output **SOBER** otherwise output **random**

3.6 Related results on SOBER-t32 with stuttering

An obvious extension of the attack in the previous section would be to guess the most probable keystream positions for $v_{t+\tau_1}, \ldots, v_{t+\tau_5}$, given $z_n = v_t$. For $v_{t+\tau_1}$, simulations show that the most probable position in the keystream is z_{n+6}, and that this event occurs with probability 0.217. Similarly, the most probable position for $v_{t+\tau_2}$ is z_{n+7} and the probability of this event is 0.198. But, since τ_3, τ_4, τ_5 are all in the order of 2^{32}, the probability of guessing the positions of $v_{t+\tau_3}, \ldots, v_{t+\tau_5}$ in the output will be very small, but will still give an attack which is better than an exhaustive key search. This approach has been exploited in a paper by Babbage et al. [4]. They derive the probability that the NLF output v_t, at time t, appears at its most probable position as

$$Pr(v_t \to \frac{n - \lfloor \frac{n}{25} \rfloor}{2}) = \frac{\lambda}{\sqrt{\frac{8\pi n}{25}}}, \tag{3.38}$$

where λ is a constant determined by simulation to be $\lambda \approx 0.84$. The expression for the most probable position is determined by the fact that on average every 25th word coming from the NLF is a SCW and thus removed from the possible output words. Then, of all the remaining words, half of them are expected to be output as keystream material, and the other half are expected to be discarded.

Now the probability that $v_{t+\tau_3}, \ldots, v_{t+\tau_5}$ appears at its most probable position in the keystream can be calculated using (3.38). That, together with the simulated probabilities for the mappings $v_{t+\tau_1} \to z_{n+6}$ and $v_{t+\tau_2} \to z_{n+7}$ gives a total probability p_0, that each of $v_{t+\tau_i}$, $i = 1, \ldots, 5$ are at their most probable positions, of

$$p_0 = 0.217 \cdot 0.198 \cdot 2^{-17.3} \cdot 2^{-18.0} \cdot 2^{-16.8} = 2^{-56.6}. \tag{3.39}$$

Using the value obtained in (3.37) together with (3.39) they calculate the Chernoff information between the distribution of the stuttered output P_Y and the uniform distribution to be

$$C(P_Y, P_U) \approx p_0^2 C(P_0, P_U) = 2^{2 \cdot -56.6} \cdot 2^{-81.5} = 2^{-194.7}, \tag{3.40}$$

where P_0 is the distribution from (3.37). The total attack needs about $32 \cdot 2^{194.7} + 17 \cdot 2^{32} < 2^{200}$ words of keystream output and about the same computational complexity.

In the same paper [4], Babbage et al. also give estimates on how efficient the attack presented in Section 3.4 on full SOBER-t16 with stuttering would be, if extended to full SOBER-t32. This approach gives an estimated data complexity of 2^{153} required words and a similar computational complexity.

3.7 Summary

In this chapter we have considered the stream ciphers SOBER-t16 and SOBER-t32. We have derived a distinguishing attack on SOBER-t16 with and without a stuttering unit, based on a linear approximation of the NLF function. We can distinguish the output sequence from a random source using at most 2^{92} keystream words and with the same time complexity in the case of no stuttering. For full SOBER-t16 we can distinguishing it using at most 2^{111} keystream words and the same time complexity.

For SOBER-t32 without the stuttering unit we can, due to a fairly strong bit correlation in the NLF function, distinguish the output from a random source using 2^{87} keystream output words and the same time complexity. Results from [4] show that the full SOBER-t32 can be distinguished using at most 2^{153} keystream symbols and the same time complexity.

We have also reviewed some fundamentals of hypothesis testing, including the Neyman-Pearson lemma and Chernoff information.

4
Cryptanalysis of $A5/1$

The stream cipher $A5/1$ is the strong version of the encryption algorithm used in the GSM standard to provide privacy for more than 130 million customers in the air link of their voice and data communication. GSM is designed to work seamlessly over national borders and different network operators. This fact, together with the cryptography services provided, makes GSM one of the first major international cryptographical challenges. The history of GSM began in the early 1980's, when a greater variety of analog mobile telephone systems were being used to a much larger extent by the public, and the problems of inter-operability and demand were becoming more apparent. Imagine a car, speeding down the Autobahn in Germany, that suddenly stops dead when crossing the border to France. This was almost the situation in the early 1980's for mobile telecom users. The leading telecom companies focused solely on local national networks and were blind to the increasing international business.

In 1985 the GSM project was launched with support from the European Commission with the aim of providing a digital system. The development in Very Large Scale Integrated (VLSI) circuits promised a future with small hand-held mobile devices, even though the first phones seemed to target body-building customers only, in terms of size and weight. In 1987 most of the details were finalised, and the launch year for a limited set of services was set for 1991, followed by a coverage of the major European cities in 1993.

The attack presented in this chapter targets the cipher $A5/1$ used for encrypting the voice and data over GSM. The attack is based on a bad initialisation of the cipher; the fact that the key and the frame counter are initialised in a linear fashion. This "bad property" enables us to launch a kind of correlation attack, which is quite powerful.

As opposed to all other attacks presented so far, this attack is (almost) independent of the shift register lengths. Instead, it depends on the number of times that the cipher

is clocked before it starts producing the output bits. In $A5/1$ this number is 100. If this number is increased, the attack becomes weaker and vice versa. The resulting attack recovers the initial state of $A5/1$ in a few minutes without requiring any notable pre-computation and without extensive memory requirements. The results in this chapter were originally presented in [31, 34].

The chapter is organised as follows. In Section 4.1 we first present how security is maintained in GSM, including the authentication of the mobile device and the computation of the secret encryption key. Next, we give a more detailed description of the stream cipher $A5/1$ in Section 4.2. Some related work on $A5/1$ is presented in Section 4.3 and the basis of our attack is presented in Section 4.4. In Section 4.5 we give a refined attack and show some simulation results from the attack in Section 4.6. The chapter is concluded with a summary in Section 4.7.

4.1 An overview of the security management in GSM

In this section we will present a general overview of the security management in GSM. We will focus on the authentication procedure and on how the encryption key for the stream cipher, responsible for the encryption of both voice and data, is derived. All air link security in GSM stems from a single secret key, the Subscriber Authentication Key K_i, fixed (but individual) for each user. This key is a 128 bit number stored only in the Subscriber Identity Module (the SIM card) of the user and in the Home Location Register (HLR) of the network operator. The SIM card is the small plastic card each user plugs into their phone. This SIM also contains the user's International Mobile Subscriber Identity (IMSI), which uniquely identifies each user. The IMSI is linked with the user's phone number and used for billing purposes. Thus, a user can change the physical phone, but continue using the same SIM card and have the same phone number as previously. The International Mobile Equipment Identity (IMEI) identifies the Mobile Equipment (ME) to confirm it is allowed to operate within the network.
The physical phone, together with the SIM card, constitute a Mobile Station (MS). Figure 4.1 shows a schematic picture of the GSM network architecture. The MS communicates over the air with the Base Transceiver Station (BTS). Each BTS covers a geographical area called a *cell*, and serves all mobile phones within that cell (if they are allowed on the operator network). The BTS forwards the call via the Base Station Controller (BSC) to the network backbone and particularly to the Mobile Switching Controller (MSC). The MSC controls the traffic among several cells and serves as an interface to the regular public switched telephone network.

$A3$	The algorithm used to produce the SRES. Resides on the SIM and in the AuC.	$A5$	The algorithm used to encrypt voice and data over the GSM air link. Resides in the ME. There are various different implementations; $A5/0$ which is no encryption at all, $A5/1$ which is the strong version, and $A5/2$ which is the weaker version, targeting the market outside Europe.
$A8$	The algorithm used to produce the session key K_c. Resides on the SIM and in the AuC.	AuC	The Authentication Centre. Generates the $RAND$ and computes the correct $SRES$ and the session key K_c.
BSC	The Base Station Controller. The common node between several BTSs and the MSC.	BTS	The Base Transceiver Station, a base station with which the MS communicates.
HLR	The Home Location Register. A database that stores the secret key for each user.	IMEI	International Mobile Equipment Identity. Uniquely identifies the ME.
IMSI	International Mobile Subscriber Identity. Uniquely identifies the user and binds them to their phone number and billing address.	K_c	The 64 bit session key. Used for encrypting the voice and data in the $A5$ algorithm.
K_i	The 128 bit secret key shared between the SIM and the HLR of the home network.	ME	The Mobile Equipment. The phone or the computer.
MS	The Mobile Station, i.e. the ME and the SIM.	MSC	The Mobile Switching Centre. Provides access to other networks and the PSTN.
PSTN	The Public Switch Telephone Network.	$RAND$	A 128 bit random number generated in the AuC.
SIM	Subscriber Identity Module. Contains the secret key K_i, the IMSI and the algorithms A3 and A8.	$SRES$	The Signed Response. The result of applying the $A3$ algorithm to $RAND$ and K_i.
VLR	The Visitor Location Register. Stores C/R triples for MSs that are visiting the network.		

Table 4.1: Explanation of frequently used acronyms.

Authenticating the mobile station

When a user powers up their phone (or MS), it starts transmitting a call signal to attempt to communicate with a BTS. The MS identifies itself by its IMSI[1] and IMEI. The BSC for the area contacts the MSC and asks for a challenge/response triple ($RAND$, $SRES$, K_c) for user IMSI using a ME with identification IMEI. The number $RAND$ is a 128 bit random number used to challenge the MS. The number $SRES$ is the correct answer to the challenge, used to check the response from the MS, and K_c is the *session key*, later to be used in the encryption of the voice data.

We will return to what the MSC needs to do to generate the triple, and for a moment assume that it simply sends back an answer to the BSC. When the BSC has received the triple, it sends the random number $RAND$ to the MS, and challenges the MS to calculate a response. The purpose of this challenge/response scheme is to

[1] Normally the IMSI is only sent the very first time the SIM is used. After that, a frequently altered, temporary number is used to protect the identity of the user to eavesdroppers.

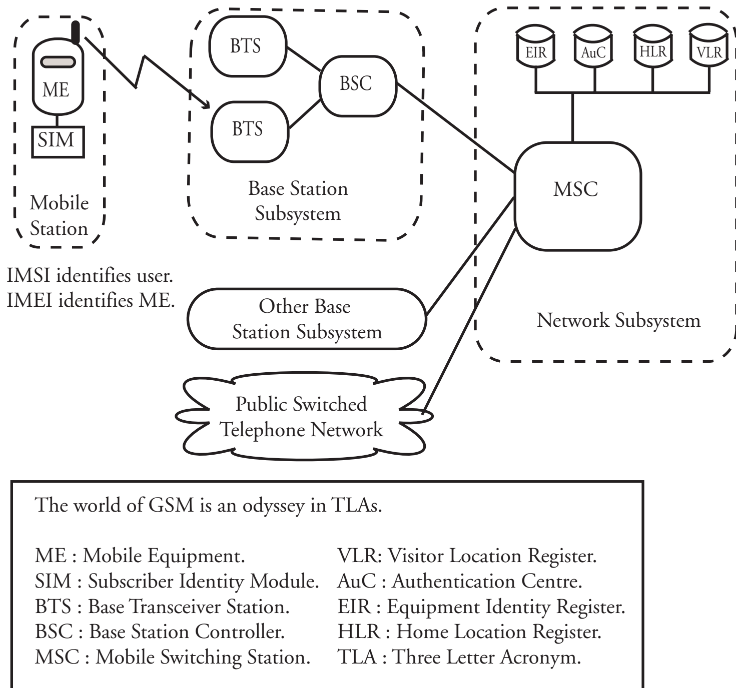


Figure 4.1: GSM network architecture.

authenticate the MS to the network, i.e. the network should be certain that the MS is a legitimate device. The important point from a security perspective, is to verify that the MS has the correct secret key K_i.

In calculating the response, the MS takes the received random number $RAND$ and its secret key K_i and uses those values as input to a function called $A3$, see Figure 4.2. The output $SRES'$, calculated by the MS, is then sent back to the BSC, which compares it to the correct answer, given by the MSC, i.e. the BSC checks if $SRES' = SRES$. If the response is correct, the MS has authenticated itself to the network and is allowed to operate within the network. The steps are pictured in Figure 4.3.

We now return to the MSC and the procedure for obtaining the challenge/response triple. Firstly, the MSC consults the Equipment Identity Register (EIR), which is a database of IMEI values. In this database there are three lists maintained by the operator; the white list, the grey list, and the black list. The white list contains all the

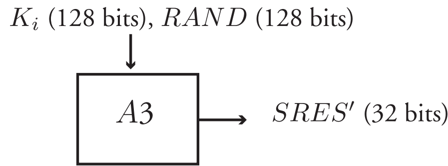


Figure 4.2: Signed response (SRES) calculation.

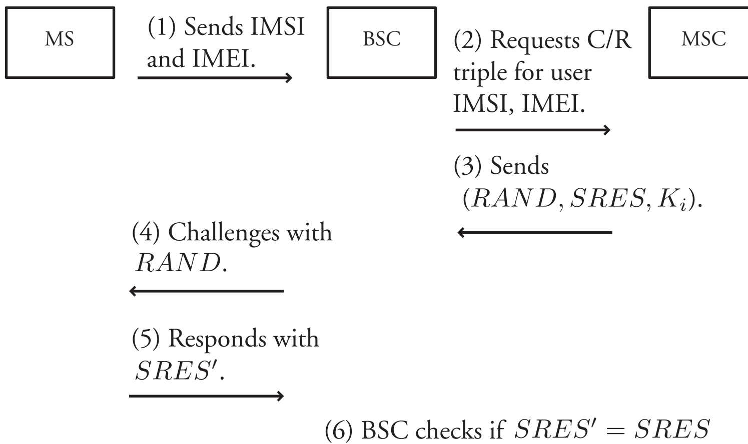


Figure 4.3: Flowchart for the challenge/response scheme used in authenticating the MS.

IMEI numbers of the MEs that are allowed to operate within the network. The grey list contains the IMEI of the MEs that are under observation for possible problems, and the black list contains the IMEI of the MEs that are prohibited to operate within the network. A black listed ME could, for example, be a stolen phone or a cloned phone.

After the equipment is verified the MSC proceeds to verify the user (the IMSI). Now, if the MS is operating in its home network, the MSC has all the information about the user stored in a database called the Home Location Register (HLR) and the MSC consults the HLR for the user's secret key, K_i. The HLR forwards the request to the Authentication Centre (AuC), which generates a 128 bit random number $RAND$, and inputs $RAND$ and K_i to the $A3$ algorithm, as shown in Figure 4.2. The result from the calculations in $A3$ is the true answer to the challenge, $SRES$. The AuC now also computes the last item in the triple, the session key K_c, which is explained below. The final step for the AuC is to return the computed triple to the HLR, which then

forwards it to the MSC.

If a user is trying to operate in another network than their home network, that visited network has no information on that user's secret key, and needs to contact the home network of the user to retrieve that information. The home network does not send the secret key to the visited network, but performs the challenge/response triple computation itself and sends the triple to the visited network. This information is stored in the Visitor Location Register (VLR) in the visited network. The VLR acts as a combination of the HLR and the AuC for the MSC in the visited network.

Calculation of the session key K_c

The session key, K_c, is the key used for the encryption of the voice and data in GSM. It is also derived through the challenge/response scheme employed during authentication, and is changed only when the network decides to re-authenticate a user. The network operator can control how often the user is to be authenticated. From a security perspective, one would like the user to be authenticated before each call, but that increases the signalling load in the network and lengthens the call setup time.

To generate the session key, both the MS and the AuC execute an algorithm called $A8$. The inputs to $A8$ are the random number $RAND$ (used in the challenge/response) and the secret key K_i, see Figure 4.4. The output from $A8$ is the 64 bit session key K_c used for encryption[2].

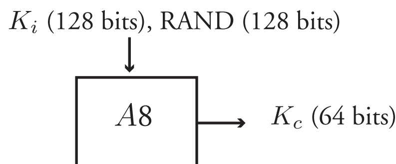


Figure 4.4: Session key (K_c) calculation.

An interesting fact is that neither of the algorithms, $A3$ or $A8$, are specified in the GSM standard. Only the input and output of the algorithms are specified, but exactly how the calculations are done is up to the network operator. The algorithms both reside, and are executed, on the SIM card in the MS, making it easy for the network operator to issue SIM cards with their own algorithms. In the AuC it is easy for the operator to implement any algorithms of their choice.

Even though the algorithms $A3$ and $A8$ are operator specific, almost all operators have initially chosen an algorithm called COMP128 for both the $A3$ and the $A8$ functions. The COMP128 algorithm can produce both the $SRES$ and the K_c in

[2]However, frequently in the past the last 10 bits of K_c were set to zero, making the effective key size only 54 bits.

one execution, which saves memory and execution time in the SIM card. However, the COMP128 algorithm has been broken [11], and if an attacker can get physical access to the SIM card, and mounts a chosen-plaintext attack on the challenge/response scheme, she can recover the secret key K_i in a couple of hours using a SIM card reader connected to an ordinary PC. The implications are that an attacker can clone a user's SIM card and have all phone bills paid by someone else.

The derived session key is then used in the encryption algorithm $A5$, which is actually a family of ciphers. The first, $A5/0$, is a dummy cipher, providing no encryption. This version allows the network operator to force the communication into plaintext. The next version, $A5/1$, is the strong version of the algorithm, employed in most operator networks. This will be attacked in this chapter. There is also a weaker version, $A5/2$, originally intended for a non-Western market. The reason to also specify a weaker algorithm was that the export restrictions on strong cryptographical devices and algorithms were much tougher 15 years ago, and European governments did not want to provide certain countries with strong encryption algorithms. Recently, a new version of the cipher called $A5/3$ was introduced, which is an even stronger version than $A5/1$.

The designs of $A5/1$ and $A5/2$ were initially kept secret from public scrutiny, but a sketch of the design of $A5/1$ was leaked in 1994 [1], and the exact design was reverse engineered in 1999 by Briceno el al. [12] from an actual GSM telephone. The design was later confirmed by the telecom industry. Both $A5/1$ and $A5/2$ are stream ciphers based on irregular clocking of three linear feedback shift registers. The keystream is produced by xoring the output from the three registers. The difference between $A5/1$ and $A5/2$ is the length of the LFSRs and the size of the key, as previously stated. The development of $A5/3$ has been public from the start, and the recently decided standard is a stream cipher based on a block cipher called Kasumi.

We will now focus on the strong version, $A5/1$, and give a detailed description of the cipher and the inadequate initialisation procedure enabling our attack.

4.2 A description of $A5/1$

A GSM conversation is sent as a sequence of frames, where one frame is sent every 4.6 milliseconds. Each frame contains 114 bits representing the communication from the MS to the BTS, and another 114 bits representing the return communication. Each conversation is encrypted by the session key K_c. For each frame to be sent, the session key K_c is mixed with a publicly known *frame counter*, denoted F_n, where n indicates different frames, and the result serves as the initial states of the shift registers in the $A5/1$ generator. The generator then produces 228 bits of keystream, which are xored with the 228 bits of plaintext to produce the ciphertext.

$A5/1$ consists of three short binary LFSRs of lengths 19, 22, 23, denoted by R1, R2, R3, respectively. These three LFSRs all have primitive feedback polynomials. The

Condition			Registers clocked		
$C1$	$= C2$	$= C3 \oplus 1$	R1	R2	
$C1$	$= C2 \oplus 1$	$= C3$	R1		R3
$C1 \oplus 1$	$= C2$	$= C3$		R2	R3
$C1$	$= C2$	$= C3$	R1	R2	R3

Table 4.2: The different clocking tap conditions and corresponding registers that are clocked.

keystream of $A5/1$ is given as the xor of the output of the three LFSRs, as illustrated in Figure 4.5.

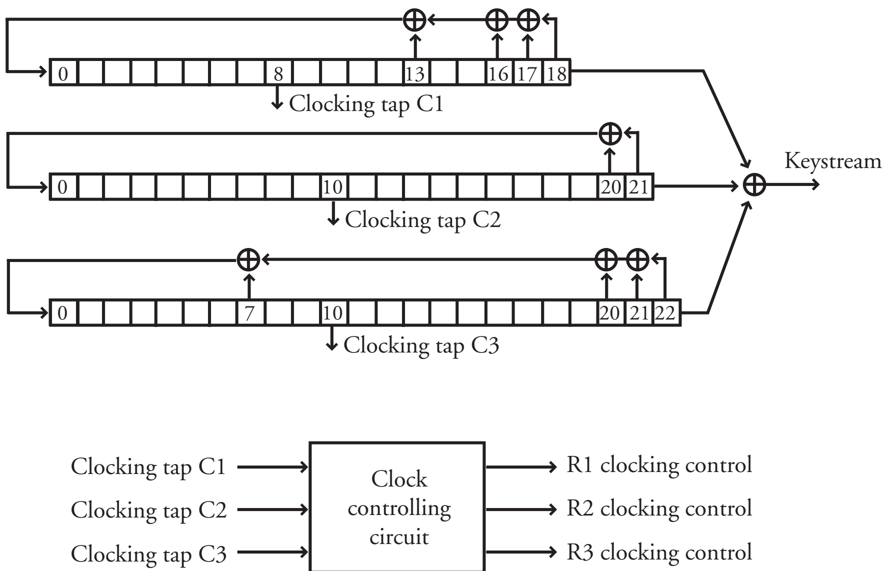


Figure 4.5: Schematic description of $A5/1$.

The LFSRs are clocked in an irregular fashion. It is a type of stop/go clocking with a majority rule as follows. Each register has a certain clocking tap, denoted $C1$, $C2$, $C3$, respectively. Each time the LFSRs are to be clocked, the three clocking taps $C1$, $C2$, $C3$ determine which of the LFSRs that are clocked according to Table 4.2. Note that at each step at least two LFSRs are clocked, and that the probability of an individual LFSR being clocked is $3/4$.

Finally, we describe the key initialisation. Firstly, the LFSRs are initialised to zero.

They are then all clocked 64 times, ignoring the irregular clocking, and the key bits of K_c are consecutively xored in parallel to the feedback of each of the registers.

In the second step the LFSRs are clocked 22 times, ignoring the irregular clocking, and the successive bits from F_n are again xored in parallel to the feedback of each of the registers. Let us call the contents of the LFSRs at this time the *initial state of the frame*.

In the third step, the three registers are clocked for 100 additional clock cycles *with* the irregular clocking, but ignoring the output. Then finally, the three registers are clocked for 228 additional clock cycles with the irregular clocking, producing the 228 bits that form the keystream. As $A5/1$ is an additive stream cipher, the keystream is xored to the plaintext to form the ciphertext. We denote the keystream output by $\mathbf{z} = z_1, z_2, \ldots, z_{228}$.

The proposed attack is a known-plaintext attack, and since the digital representation of the conversation is split into frames of length 228 bits, the corresponding known-plaintext assumption is now that the attacker is given access to the keystream from m different frames, each keystream of length 228 bits. Note that the LFSRs for each frame are initialised with the same session key K_c but with different frame counters F_n.

Given the keystream, the goal of the proposed attack is to recover the initial state of the keystream generator. In the case of the GSM system the initial state of the shift registers is a linear combination of the publicly known frame counter and the secret session key. By deducing the initial state, the secret session key K_c can be recovered.

4.3 Related work

As stated previously, the exact design of the $A5/1$ algorithm was secret until 1999, so the early analysis on the cipher was done on an alleged version. An erroneous description of the cipher arrived anonymously to a research group in Cambridge in 1994, and Anderson and Roe [1] then described a simple guess-and-determine attack involving guessing the initial states of registers R1 and R2, then deriving the contents of register R3 using the observed keystream. The initial estimate of the complexity was 2^{41}, but later altered to approximately 2^{45}, since the contents of R3 cannot be directly obtained and additional guessing is required.

In 1997, Golić [48] described two attacks on the alleged $A5/1$. The first is an attack based on solving systems of linear equations and requires about 2^{40} operations, where each operation involves finding a solution to a system of linear equations. The attack, however, needs only 64 bits of keystream material for setting up the equation systems and verification. The second attack is a general trade-off attack on stream ciphers (which was independently published by Babbage [3]). This time-memory trade-off attack can find the initial state of the cipher using a pre-computed table of 2^{42} 128 bit entries, and probing the table with about 2^{22} queries during the active phase of the

attack.

In 2000, Biryukov, Shamir, and Wagner [8] refined the time-memory trade-off ideas of Golić to a very impressive level. They present two attacks, both based on highly optimised and cipher specific search algorithms. The first, called the *biased birthday attack*, considers special 16 bits patterns in the keystream and stores their initial state on a hard disk. Whenever the attacker then sees the specific pattern in the keystream, she can perform a lookup on the correct initial state. The attack reduces the original complexity of the attack by Golić by observing that some of the subsequences where the special 16 bit pattern are occurring are more likely to appear in the keystream of $A5/1$ then others (in fact, some subsequences are not possible at all). By only storing a fraction of the data, but still having a large possibility of occurrence, their first attack needs approximately 2^{48} operations in pre-computation time, 150 gigabytes of storage, roughly 2 minutes of known plaintext and 1 second of active attack time. The second attack in [8], called the *random subgraph attack*, reduces the required observed keystream to about 2 seconds of known plaintext at the cost of increased storage (about 300 gigabytes) and the active attack time is 2–3 minutes.

Also in 2000, Biham and Dunkelman [6] presented a guess-and-determine attack, combined with some additional table lookup. The attack starts by assuming a position in the keystream in which the third register R3 is not clocked for 10 clockings. Then they guess 12 bits and can recover the content of both R1 and R2. To recover the bits in R3, they build a table indexed by some bits in the output stream and the (partial) contents in R1 and R2. The entries in the tables are the possible values of the bits in R3 which generated the output. Some additional observations and tricks reduce the time complexity to $2^{39.91}$ $A5/1$ clockings with $2^{20.8}$ bits of known plaintext. The pre-computational requirements are 2^{38} operations with 32 gigabytes of memory. A second version increases the time complexity of the attack to $2^{40.97}$, while gaining in pre-computation time ($2^{33.6}$) and memory (2 gigabytes).

Very recently, Barkan, Biham and Keller [5] presented a collection of attacks on $A5/2$, including a ciphertext only attack. The attack is based on the highly redundant error-correcting code employed during call set up, together with a weakness in the protocol during set up. With their attack, the session key can be recovered in one second using a regular PC. If a fraudulent base station is used, they note that an attacker can force the MS into using $A5/2$ and then recover the session key. When the MS later contacts the original operator network and switches cipher to the stronger $A5/1$, the session key is not necessarily changed and subsequent calls can easily be decrypted.

All the previous attacks on $A5/1$ are based on a time-memory trade-off, and are exponentially expensive in the length of the shift registers. Thus, if the lengths were increased, say by a factor of 2, the above attacks would become impractical. A new approach will now be presented, where the crucial parameter is the number of premix clockings rather than the length of the shift registers.

4.4 A basic correlation attack

Let us start by presenting some fundamental observations. Firstly, from the key initialisation description we note that the initial state is a linear function of K_c and F_n. Let $K_c = (k_1, k_2, \ldots, k_{64})$ and $F_n = (f_1, f_2, \ldots, f_{22})$, where $k_i, f_i \in \mathbb{F}_2$. Let $u_0^1, u_1^1, \ldots$ be the LFSR output sequence produced by *regular* clocking of R1 after the key and frame number initialisation (starting with the initial state). Similarly, let $u_0^2, u_1^2, \ldots$ be the LFSR output sequence produced by regular clocking of R2, and finally, let $u_0^3, u_1^3, \ldots$ be the LFSR output sequence of R3 when clocked regularly. This means that $(u_0^1, u_1^1, \ldots, u_{18}^1)$ forms the initial state of R1 for the given frame and similarly for R2 and R3.

Recall from Section 4.2 the linear fashion in which the key K_c and the frame number F_n together form the initial state of the frame. Given this observation, we can write each LFSR output symbol from R1 as

$$u_t^1 = \sum_{i=1}^{64} c_{it}^1 k_i \oplus \sum_{i=1}^{22} d_{it}^1 f_i, \tag{4.1}$$

for some known binary constants c_{it}^1, $i = 1, \ldots, 64$, $t \geq 0$, and d_{it}^1, $i = 1, \ldots, 22$, $t \geq 0$. We introduce the notation $s_t^1 = \sum_{i=1}^{64} c_{it}^1 k_i$ and $\hat{f}_t^1 = \sum_{i=1}^{22} d_{it}^1 f_i$, $t \geq 0$. Then we can write

$$u_t^1 = s_t^1 + \hat{f}_t^1, \quad t \geq 0. \tag{4.2}$$

We call the sequence s_t^1 the *key part* of sequence u_t^1 and the sequence $\hat{f}_t^1$ is called the *frame number part* of u_t^1. Of course, we can also write

$$s_t^1 = \sum_{i=0}^{18} \hat{c}_{it}^1 s_i^1, \tag{4.3}$$

for some known binary constants $\hat{c}_{it}^1$, $i = 0, \ldots, 18$, $t \geq 0$.

Note that $s_0^1, s_1^1, s_2^1, \ldots$ is an unknown binary sequence (2^{19} possible sequences) that remains the same for all frames within a conversation (it depends only on K_c, which is fixed during a conversation). Furthermore, $\hat{f}_1^1, \hat{f}_2^1, \ldots,$ is a known contribution from the frame counter that differs for each frame. Since the frame number is always known, the above sequence $\hat{f}_1^1, \hat{f}_2^1, \ldots,$ can be calculated for each frame. For registers R2 and R3 we can, in a similar way, write the output symbols as

$$u_t^2 = \sum_{i=1}^{64} c_{it}^2 k_i + \sum_{i=1}^{22} d_{it}^2 f_i, \tag{4.4}$$

$$u_t^3 = \sum_{i=1}^{64} c_{it}^3 k_i + \sum_{i=1}^{22} d_{it}^3 f_i, \tag{4.5}$$

with known binary constants $c_{it}^2, c_{it}^3, i = 1, \ldots, 64, t \geq 0$ and $d_{it}^2, d_{it}^3, i = 1, \ldots, 22$, $t \geq 0$. Following the notation for R1, we introduce

$$s_t^2 = \sum_{i=1}^{64} c_{it}^2 k_i, \quad \text{and} \quad \hat{f}_t^2 = \sum_{i=1}^{22} d_{it}^2 f_i, t \geq 0,$$
$$s_t^3 = \sum_{i=1}^{64} c_{it}^3 k_i, \quad \text{and} \quad \hat{f}_t^3 = \sum_{i=1}^{22} d_{it}^3 f_i, t \geq 0,$$

for the key part and frame number part of the output sequences u_t^2 and u_t^3 of registers R2 and R3. Similarly to (4.2) we also write

$$u_t^2 = s_t^2 + \hat{f}_t^2, \quad t \geq 0, \tag{4.6}$$
$$u_t^3 = s_t^3 + \hat{f}_t^3, \quad t \geq 0. \tag{4.7}$$

Let us now present the very basic idea for a correlation attack on $A5/1$. It will later be considerably refined. Let $z_1, z_2, \ldots, z_{228}$ denote the observed keystream from $A5/1$ in a certain frame. Let us consider what happens after the LFSRs have received their initial values. Firstly, the registers are clocked irregularly 100 times, producing no output, then they are clocked once and the first output symbol is produced. Since each of the shift registers will clock on average three times out of four, we can expect that after 101 irregular clockings, each LFSR will have been clocked about 76 times. Assume for a moment that each of the three LFSRs has been clocked *exactly* 76 times. Then the produced bit z_1 is the xor of the output of the three LFSRs

$$u_{76}^1 + u_{76}^2 + u_{76}^3 = z_1. \tag{4.8}$$

Since $\hat{f}_1, \hat{f}_2, \ldots$ is a known quantity in each frame, we can simply calculate its contribution to the output bit. From (4.2), (4.6) and (4.7) we can rewrite (4.8) as

$$s_{76}^1 + s_{76}^2 + s_{76}^3 = \hat{f}_{76}^1 + \hat{f}_{76}^2 + \hat{f}_{76}^3 + z_1. \tag{4.9}$$

Note that the right hand side of (4.9) contains known quantities only. Denote the right hand side of (4.9) in frame j with $O_{(76,76,76,1)}^j$. Under the assumption that each LFSR has been clocked exactly 76 times, we get one bit of information about the key in frame j, since

$$s_{76}^1 + s_{76}^2 + s_{76}^3 = O_{(76,76,76,1)}^j. \tag{4.10}$$

Of course, if the assumption is incorrect we can expect (4.10) to hold with probability $1/2$. Hence, we have identified a correlation by calculating

$$P(s_{76}^1 + s_{76}^2 + s_{76}^3 = O_{(76,76,76,1)}^j) = P(\text{assumption correct}) \cdot 1 + P(\text{assumption incorrect}) \cdot 1/2. \tag{4.11}$$

In this case the probability of all three LFSRs being clocked exactly 76 times is calculated to be about 10^{-3}. Hence

$$P(s^1_{76} + s^2_{76} + s^3_{76} = O^j_{(76,76,76,1)}) = 1/2 + 1/2 \cdot 10^{-3}. \qquad (4.12)$$

The left-hand-side of (4.10), $s^1_{76} + s^2_{76} + s^3_{76}$, remains constant over all frames. It is not hard to show that if we have access to a few million frames and thus can calculate $O^j_{(76,76,76,1)}$ for each frame, then $s^1_{76} + s^2_{76} + s^3_{76}$ can be determined with high confidence.

The value of $s^1_{76} + s^2_{76} + s^3_{76}$ gives us one bit of information about the key. By considering other assumed triples for the number of clockings of the three LFSRs, we can derive more information about the key and eventually recover it.

4.5 A refinement of the attack

The previously described attack is very simple, but has the drawback that it requires many frames. In this section we show how to refine the attack.

Denote the produced keystream after the initialisation with the key K_c and the frame counter F_n by $w_0, w_1, \ldots, w_{328}$. Recall that the first 101 symbols, $w_0, w_1, \ldots,$ w_{100} are discarded during the initialisation and the keystream z_i is given by $z_1 = w_{101}, z_2 = w_{102}, \ldots, z_{228} = w_{328}$. Consider a certain assumed clocking triple (cl_1, cl_2, cl_3) of registers R1, R2 and R3. This clocking might occur in several keystream positions, e.g. the clocking (79,79,79) might not only appear at position 101 but also at positions 102, 103, ..., et cetera. Keystream positions before 101 are not considered since they are discarded and are not accessible. Let

$$P((cl_1, cl_2, cl_3) \text{ in } v\text{th position}), \qquad (4.13)$$

denote the probability of clocking (cl_1, cl_2, cl_3) occurring at position v (i.e. keystream symbol w_v).

Given a specific clocking triple (cl_1, cl_2, cl_3), we can calculate an interval $\mathcal{I}$ for v, where that clocking triple has a non-negligible probability of occurring. So, instead of using only *one* keystream position when calculating the correlation probability as done in (4.11) and (4.12), we can use *all* positions ($v \geq 101$) where there is a non-negligible probability of occurrence. In frame j we calculate a correlation probability (implicitly conditioned on $\mathcal{I}$ in the jth frame), denoted $p^j_{(cl_1,cl_2,cl_3)} = P(s^1_{cl_1} + s^2_{cl_2} + s^3_{cl_3} = 0)$, as a weighted voting over several positions, using the formula

$$p^j_{(cl_1,cl_2,cl_3)} = \sum_{v \in \mathcal{I}} P((cl_1, cl_2, cl_3) \text{ in } v\text{th position}) \cdot [O^j_{(cl_1,cl_2,cl_3,v-100)} = 0]$$
$$+ 1/2 \cdot (1 - \sum_{v \in \mathcal{I}} P((cl_1, cl_2, cl_3) \text{ in } v\text{th position})), \qquad (4.14)$$

where $[x = 0]$ is the indicator function which equals 1 if $x = 0$ and 0 otherwise. Also, $O^j_{(cl_1,cl_2,cl_3,v-100)} = \hat{f}^1_{cl_1} + \hat{f}^2_{cl_2} + \hat{f}^3_{cl_3} + z_{v-100}$ for frame j, as in (4.9) and (4.10). Finally, $\mathcal{I}$ is the interval over which there is a non-negligible probability of occurrence for the clocking triple (cl_1, cl_2, cl_3).

If we assume that the bits entering the clock controlling device of $A5/1$ are uniformly distributed independent bits, we can write the probability (4.13) as a recursive formula

$$P((cl_1, cl_2, cl_3) \text{ in } v\text{th position}) = F(cl_1, cl_2, cl_3, v), \tag{4.15}$$

where

$$\begin{aligned}
F(cl_1, cl_2, cl_3, 0) &= 1 \text{ if } cl_1 = 0, cl_2 = 0 \text{ and } cl_3 = 0,\\
F(cl_1, cl_2, cl_3, v) &= 0 \text{ if } cl_1 < 0 \text{ or } cl_2 < 0 \text{ or } cl_3 < 0,\\
F(cl_1, cl_2, cl_3, v) &= 0 \text{ if } cl_1 > v \text{ or } cl_2 > v \text{ or } cl_3 > v,\\
F(cl_1, cl_2, cl_3, v) &= 0.25F(cl_1 - 1, cl_2 - 1, cl_3 - 1, v - 1)\\
&\quad + 0.25F(cl_1, cl_2 - 1, cl_3 - 1, v - 1)\\
&\quad + 0.25F(cl_1 - 1, cl_2, cl_3 - 1, v - 1)\\
&\quad + 0.25F(cl_1 - 1, cl_2 - 1, cl_3, v - 1), \text{ otherwise.}
\end{aligned}$$

This formula will give an exact probability under the assumption of independent uniformly distributed clocking bits. We will use these probabilities to approximate the actual $A5/1$ case. The approximation works well when the probability is fairly high (as in cases considered in this attack), since there are several different initial states that give the desired (cl_1, cl_2, cl_3, v).

Under the same assumptions as above, we can give a somewhat easier formula which gives a closed expression for the probability in (4.13) as

$$P((cl_1, cl_2, cl_3) \text{ in } v\text{th position}) = \frac{\binom{v}{v-cl_1}\binom{v-(v-cl_1)}{v-cl_2}\binom{v-(v-cl_1)-(v-cl_2)}{v-cl_3}}{4^v}. \tag{4.16}$$

The formula in (4.16) can be derived using the following arguments. The first LFSR has to *not* be clocked $v - cl_1$ times. The number of possible positions for this to happen in is $\binom{v}{v-cl_1}$. For the remaining positions, the second LFSR has to *not* be clocked $v - cl_2$ times. These positions must not coincide with the positions in which the first LFSR is halted, since at most one register can be halted at each clocking. Similarly for the third LFSR and at the remaining positions, all three registers need to be clocked. This results in the multinomial distribution given in (4.16).

The expression in (4.16) gives the same value as (4.15) for any *valid* clocking triple (cl_1, cl_2, cl_3) in position v, e.g. it will fail for $(0, 0, 0)$ in position 10 which cannot

$P \cdot 10^4$	Eq. (4.15) or Eq. (4.16)	Estimation by simulation
P(76,76,76,101)	9.7434	9.7331
P(79,79,79,105)	9.2012	9.2033
P(80,80,80,105)	6.6388	6.6388
P(79,80,81,106)	8.3858	8.4126
P(82,82,82,109)	8.7076	8.7269

Table 4.3: Comparison of the approximated value given by (4.15) or (4.16) and the estimated value from simulations of the $A5/1$ cipher. All values should be multiplied by 10^{-4}.

happen in $A5/1$. Table 4.3 gives an indication of the validity of the approximation compared to an estimated value based on 100 million simulations of the $A5/1$ cipher.

We give a small fictitious example to clarify (4.14) for three different sequences of values $O^j_{(cl_1,cl_2,cl_3,v-100)}$, $v = 101, 102, ..., 106$. Note that the probabilities given in Example 4.1 are unrelated to the actual $A5/1$ case.

Example 4.1: As presented in Figure 4.6, we have chosen $\mathcal{I} = \{101, \ldots, 106\}$. We see from the tabulated example that if we calculate the sequence $O^j_{(cl_1,cl_2,cl_3,v-100)}$, $v = 101, 102, ..., 106$ to be only zeros in the interval $\mathcal{I}$, using the known frame number and the keystream $\mathbf{z}$, then the probability that the key part $s^1_{cl_1} + s^2_{cl_2} + s^3_{cl_3}$ for this specific clocking is zero is fairly high (0.9). If we calculate the same sequence to be only ones, the probability of the key part being zero is low (0.1). Finally, if we have a mix of ones and zeros we see that the zeros are observed at positions where there are (in total) a higher probability of occurrence, so in this case we vote for $s^1_{cl_1} + s^2_{cl_2} + s^3_{cl_3}$ being zero, due to a slightly higher probability (0.62). □

In order to use the information in all the available frames to estimate the value of the linear combination $s^1_{cl_1} + s^2_{cl_2} + s^3_{cl_3}$ we will use a log-likelihood ratio. Firstly, define $\hat{p}_{(cl_1,cl_2,cl_3)} = P(s^1_{cl_1} + s^2_{cl_2} + s^3_{cl_3} = 0)$ as the total probability that $s^1_{cl_1} + s^2_{cl_2} + s^3_{cl_3} = 0$, taken over all frames. Recall that $p^j_{(cl_1,cl_2,cl_3)}$ denoted the same for the jth frame only. Then define the log-likelihood ratio $\Lambda_{(cl_1,cl_2,cl_3)}$ of $\hat{p}_{(cl_1,cl_2,cl_3)}$ as

$$\Lambda_{(cl_1,cl_2,cl_3)} = \ln \frac{\hat{p}_{(cl_1,cl_2,cl_3)}}{1 - \hat{p}_{(cl_1,cl_2,cl_3)}}, \tag{4.17}$$

where $\ln$ is the natural logarithm. We can now find an estimate of $\Lambda_{(cl_1,cl_2,cl_3)}$ over all

Keystream pos. v

100	101	102	103	104	105	106	107
..	z_1	z_2	z_3	z_4	z_5	z_6	...

$P((cl_1, cl_2, cl_3)$ in vth position$) =$

	0.04	0.16	0.20	0.20	0.16	0.04	
$O^j =$	0	0	0	0	0	0	
					$p^j_{(cl_1,cl_2,cl_3)} = 0.9$		
$O^j =$	1	1	1	1	1	1	
					$p^j_{(cl_1,cl_2,cl_3)} = 0.1$		
$O^j =$	1	0	1	0	0	1	
					$p^j_{(cl_1,cl_2,cl_3)} = 0.62$		

Figure 4.6: Example of three different sequences $O^j_{(cl_1,cl_2,cl_3,v)}$ and the corresponding $p^j_{(cl_1,cl_2,cl_3)}$ probabilities calculated according to (4.14).

frames by calculating

$$\Lambda_{(cl_1,cl_2,cl_3)} = \sum_{j=1}^{m} \ln \frac{p^j_{(cl_1,cl_2,cl_3)}}{1 - p^j_{(cl_1,cl_2,cl_3)}}, \tag{4.18}$$

where m is the number of available frames. For a log-likelihood ratio Λ defined as in (4.17) we know that

$$\begin{array}{ll} \Lambda = 0 & \text{if } P(s^1_{cl_1} + s^2_{cl_2} + s^3_{cl_3} = 0) = 1/2, \\ \Lambda > 0 & \text{if } P(s^1_{cl_1} + s^2_{cl_2} + s^3_{cl_3} = 0) > 1/2, \\ \Lambda < 0 & \text{if } P(s^1_{cl_1} + s^2_{cl_2} + s^3_{cl_3} = 0) < 1/2. \end{array}$$

We will now turn to specific parameter choices as we describe the final phase of the attack. Starting at position 79, we pick a suitable interval of length 8, $\mathcal{C}_1 = \{79, \ldots, 86\}$, and look at all linear combinations of $s^1_{cl_1} + s^2_{cl_2} + s^3_{cl_3}$ where each of cl_1, cl_2, cl_3 runs in the interval $\mathcal{C}_1$. For each such value of (cl_1, cl_2, cl_3) and for each frame $j = 1, \ldots, m$, we calculate $p^j_{(cl_1,cl_2,cl_3)}$ and use (4.18) to calculate $\Lambda_{(cl_1,cl_2,cl_3)}$. Using $\Lambda_{(cl_1,cl_2,cl_3)}$ we finally estimate the linear combination of key bits with a simple hard decision. For example, if $\Lambda_{(79,79,79)} = 2.56$ we estimate $s^1_{79} + s^2_{79} + s^3_{79} = 0$, if $\Lambda_{(79,79,80)} = -0.93$ then $s^1_{79} + s^2_{79} + s^3_{80} = 1$, etc.

We note that when (cl_1, cl_2, cl_3) run through all possible values in the specified interval of size 8, this gives a system of $8^3 = 512$ linear equations with $8+8+8 = 24$ unknown variables. The problem of finding the correct values of the 24 unknown variables is now equivalent to the problem of decoding a length 512 linear code of dimension 24. The estimated bits can be viewed as a received word of length 512, and the corresponding equations are parity check equations for the code. If we have enough frames to make the estimates reasonably accurate, we can decode the received word and find 24 bits from the key.

Since the log-likelihood ratio Λ represents a soft value of the probability, it is also possible to use a soft decoding algorithm. This algorithm would be expected to perform better than a hard decoding algorithm, since it takes good advantage of the given information. However, we have tried soft decoding and it did not improve the attack notably. When the number of received frames increases, the probability tends to either 0 or 1 quickly, thus reducing the advantage of soft decoding. The reduced complexity of the hard decoding algorithm seems to be a better choice in this case.

Table 4.4 shows the average, maximum and minimum number of correct estimates for the 512 equations in a run of 60 simulations, using the procedure described.

	Number of thousand frames in the estimation (m).					
	10	30	50	70	100	200
Average	283	293	309	320	326	354
Max	308	307	330	347	346	378
Min	259	283	288	303	301	339

Table 4.4: Number of correct estimates for a system of 512 equations.

Using the interval $\mathcal{C}_1 = \{79, \ldots, 86\}$ we solve for the bits $s^1_{79}, \ldots, s^1_{86}$, $s^2_{79}, \ldots,$ s^2_{86} and $s^3_{79}, \ldots, s^3_{86}$ from the key part of the registers. Using (4.3) this will give us $8 + 8 + 8 = 24$ bits of information about the key K_c (in the form of linear combinations of key bits). To fully recover the key (64 bits) we can increase the length of the interval to 22, such that we get $22+22+22 > 64$ bits of information about the key, making the decoding much harder. Instead, we propose to pick a new subinterval $\mathcal{C}_2 = \{87, \ldots, 94\}$, thus recovering another 24 bits from the key. Finally, we do the same for the subinterval $\mathcal{C}_3 = \{95, \ldots, 102\}$. Then we have recovered 24 bits from each shift register output and have a total of 72 bits. This is more than required for solving for the key K_c.

The computational work to check a solution consists of first loading the estimated bits into the register, then running the cipher backwards 79 clocks plus an additional 22 clocks for the frame number loading. Then loading the frame number in the usual

way and running the 100 premix clocks and finally checking the generated keystream output against the received keystream. The maximum number of output bits that we need to check is approximately 64, since the state space is 64 bits. The resulting computational work for checking a solution is therefore one register loading plus about $223 + 64 = 287$ cipher clockings.

4.6 Simulations of the attack

Firstly, the probabilities $p^j_{(cl_1,cl_2,cl_3)}$ were calculated for each frame $j = 1, \ldots, m$ and (cl_1, cl_2, cl_3) in the interval $\mathcal{I}$. Then the log-likelihood ratios $\Lambda_{(cl_1,cl_2,cl_3)}$ were calculated based on the received keystream.

Secondly, the decoding in our simulations was done by exhaustive search over all possible values of $s^1_{t_1}, s^2_{t_2}, s^3_{t_3}$, where t_1, t_2, t_3 each run through the interval $\mathcal{I}$. The solution which gave the closest Hamming distance to the received codeword was taken as the correct solution. However, in order to have a high probability that the correct solution is the codeword closest in Hamming distance to the received word, we needed a large number of frames. Simulations have shown that when we have fewer than about 100,000 frames, there are often other (erroneous) solutions to the system of equations that give a closer distance. To overcome this problem we save a list of the $T \approx 1000$ closest solutions for each subinterval. Picking one solution from each list (subinterval), we can combine them into three 24 bit LFSR sequences as allegedly produced by the shift registers. These sequences are then verified by running the cipher backwards as described in Section 4.5.

In the case of using an interval length of 8, we need three subintervals and the number of combinations to verify amounts to T^3, which is rather expensive. A more efficient way is to use overlapping intervals where each subinterval overlaps the previous subinterval with 2 or 3 bits. Now we only have to verify combinations that agree in the overlapping bits. We can also use the fact that the sequences we want to verify (using overlapping intervals) are 23 bits long and two of the shift registers are shorter. Thus, a first test of the correctness of the combined sequence is to check whether the last bits of the sequence fulfil the feedback polynomial for the two shortest shift registers.

Using these techniques, simulations with $T = 1000$ have shown that we can reduce the number of verifications from 1000^3 to between 5000 and 50000 for the case when the interval size is 8 and number of overlapping bits is 3. The different configurations used in our simulations are shown in Table 4.5.

Table 4.6 shows the success rate for different configurations and different number of received frames. The entries are the number of successful attacks from a batch of 100 runs and in parentheses are the attack times for each configuration. The corresponding length of the GSM conversation is also given (although this is a known plaintext attack and would not apply directly to the GSM system). The simulations were run on a PC

Interval size	Overlapping bits	Intervals
7	3	$[79,85],[83,89],[87,93],[91,97],[95,101]$
8	3	$[79,86],[84,91],[89,96],[94,101]$
9	2	$[79,87],[86,94],[93,101]$

Table 4.5: Configurations used in our simulations.

with an Intel Pentium 4 processor, running at 1.8 GHz, with 512 Mb of memory using a Linux operating system.

	Number of received frames (time of GSM conversation)		
Configuration	30000 (2m30s)	50000 (3m45s)	70000 (5m20s)
7/3	2 (1)	13 (2)	49 (3)
8/3	2 (2)	20 (3)	57 (4)
9/2	3 (3)	33 (4)	76 (5)

Table 4.6: Simulation results using a list size of $T = 1000$. Entries show the number of successes out of 100 runs. Time of attack in minutes is given in parentheses.

The pre-computation phase in the presented attack amounts to calculating and storing the probabilities that a certain number of clockings of the registers appear in a certain keystream position. These are the probabilities used in (4.14). Using the same hardware as in our simulations, it takes about 15 minutes to calculate the required tables and less that 2 Mb to store them. We have summarised the implemented attack in Figure 4.7.

Recalling (4.14), the summation is taken over an interval $\mathcal{I}$ where there is a non-negligible probability of occurrence of clocking (cl_1, cl_2, cl_3). Calculating the probabilities for the highest clocking in the simulations, $(101, 101, 101)$, shows that this clocking has a very small probability of occurring beyond the $v = 140$th keystream position (the 40th position in the output keystream since the first 100 are discarded). Therefore, the attack only needs the first 40 bits of the keystream in each frame. Furthermore, the frames used in the attack need not be consecutive.

(i) Choose a subinterval $\mathcal{C}_1$ (e.g. $\mathcal{C}_1 = [79, 86]$)

(ii) Let (cl_1, cl_2, cl_3) run through the interval $\mathcal{C}_1$. For each frame $j = 1, \ldots, m$ calculate

$$p^j_{(cl_1,cl_2,cl_3)} = \sum_{v=\mathcal{I}} P((cl_1, cl_2, cl_3) \text{ in } v\text{th pos.}) \cdot [O^j_{(cl_1,cl_2,cl_3,v-100)} = 0] + 1/2 \cdot (1 - \sum_{v=\mathcal{I}} P((cl_1, cl_2, cl_3) \text{ in } v\text{th pos.}))$$

Calculate the log-likelihood ratio of the weighted probability over all frames

$$\Lambda_{(cl_1,cl_2,cl_3)} = \sum_{j=1}^{m} \ln \frac{p^j_{(cl_1,cl_2,cl_3)}}{1 - p^j_{(cl_1,cl_2,cl_3)}}$$

Estimate the linear combination

$$s^1_{cl_1} + s^2_{cl_2} + s^3_{cl_3} = \text{HD}(\Lambda_{(cl_1,cl_2,cl_3)})$$

using a hard decision (HD) on the value of $\Lambda_{(cl_1,cl_2,cl_3)}$.

(iii) Decode the generated linear code

$$s^1_{cl_1} + s^2_{cl_2} + s^3_{cl_3} = \text{HD}(\Lambda_{(cl_1,cl_2,cl_3)})$$

for (cl_1, cl_2, cl_3) in interval $\mathcal{C}_1$ using an ML decoding through exhaustive search. Save the T closest solutions.

(iv) Repeat steps 1 to 3 for each new subinterval $\mathcal{C}_2, \mathcal{C}_3, \ldots$ until a total of 64 bits of the shift register sequences are recovered.

(v) Combine the solutions from each subinterval and check the validity of the solutions.

Figure 4.7: A summary of the proposed attack.

4.7 Summary

In this chapter, we have proposed a new attack on the $A5/1$ stream cipher, based on an identified correlation. In contrast to previously known attacks, this is not a time-

memory trade-off attack, but uses completely different properties of the cipher. It explores the weak key initialisation which allows us to separate the session key from the frame number in binary linear expressions.

The complexity of the attack is only linear in the length of the shift registers and depends instead on the number of irregular clockings before the keystream is produced. The implemented attack needs the 40 first bits from about 2^{16} (possible non-consecutive) frames, which corresponds to about 5 minutes of GSM conversation. Our implementation of the attack shows that we have a high success rate of more than 70%. This can be improved by using a larger list size and/or larger interval sizes. The complexity of the attack using the parameters presented here is quite low and the attack can be carried out on a modern PC in less than 5 minutes using very little pre-computational time and memory.

The improvements compared to previous work are the following. All previous attacks have a complexity exponential in the shift register length. The complexity of the attack presented in this paper is roughly linear in the shift register lengths.

Previously known attacks also need either high pre-computational time and/or memory complexity or they have a high active attack time complexity. The proposed attack is simple to implement, has been successfully implemented, and completes its task in less than 5 minutes.

Finally, the presented attack also identifies interesting new design weaknesses in $A5/1$ that should be considered when constructing new stream ciphers.

5

Cryptanalysis of E_0

In 1999, five major telecom corporations together published the specifications for a new radio network link called *Bluetooth* [10]. Bluetooth was developed to replace cables in short range communication. Examples of usage include synchronising a handheld computer or PDA (Personal Digital Assistant) to a stationary computer, wireless payments in shops with electronic cash, wireless handsfree equipment for mobile phones, and removing some of the cable mess found behind most computers by using a wireless keyboard, mouse and printer connection, et cetera.

Considering the proposed areas of usage, it is clear that the specification must also include cryptographic functionalities. Bluetooth provides both authentication and encryption. Authentication to ensure the correct sender and receiver and encryption to ensure privacy in the communication. Both these functions are provided at the link layer, but higher protocols can of course provide their own cryptographic layers.

Two ciphers are specified; *SAFER+*, a block cipher used in the authentication protocol, and E_0, a stream cipher used for the link encryption. In this analysis we will focus on the stream cipher and the encryption. The attack on E_0 is based on a strong correlation discovered within the cipher, and we use this correlation in a divide-and-conquer style correlation attack. The results in this chapter were first presented in [30].

This chapter is organised as follows. Section 5.1 gives an introduction to Bluetooth and a system overview. Section 5.2 describes the security system in Bluetooth, focusing on the stream cipher E_0. In Section 5.3 the discovered correlation is presented and the proposed attack on E_0 is introduced. Section 5.4 presents previously known results and newer results on E_0 by other authors, and finally a summary is given in Section 5.5.

5.1 Bluetooth—system overview

Bluetooth operates in the unlicensed 2.4 GHz frequency band. The exact parameters differ between countries but in the US and most of Europe, there are 79 RF bands defined, each of 1 MHz bandwidth. The transceiver uses frequency hopping among these 79 RF bands to achieve a communication link of 10^6 symbols per second. The channel is divided into time slots of 625 μs each. A Time-Division Duplex scheme is used to get full duplex in the channel. The link is packet based, where each packet normally covers a single time slot and is transmitted on a different hop frequency. The maximum distance between two communicating Bluetooth units is in the range of 10 to 100 metres, and the Bluetooth device can adjust the transmission power to minimise energy consumption.

Each of the previously mentioned services require different modes of operation. The PDA synchronisation should preferably be fast, but has no time critical requirements. On the other hand, the handsfree equipment uses a fairly low information bit rate, but needs a synchronous connection to the mobile phone to ensure low latency in the voice transmission. So, even if the radio link is packet based, Bluetooth can operate in both circuit and packet switching mode by reserving time slots for the synchronous transmission. It is possible to have up to three synchronous voice transmissions with 64 kb/s in each (voice) channel, or an asynchronous data transfer at maximum speed 723.2 kb/s in one direction with 57.6 kb/s in the return channel. If a symmetric asynchronous channel is desired, Bluetooth can provide 433.9 kb/s in each direction. The services are summarised in Table 5.1.

Packet type	Symmetric rate (kb/s)	Asymmetric rate (kb/s)	
		Forward	Reverse
Asynchronous services			
DM1	108.8	108.8	108.8
DH1	172.8	172.8	172.8
DM3	258.1	387.2	54.4
DH3	390.4	585.6	86.4
DM5	286.7	477.8	36.3
DH5	433.9	723.2	57.6
Synchronous services			
HV1	64.0	-	-
HV2	64.0	-	-
HV3	64.0	-	-
DV	64.0(V)+57.6(D)	-	-

Table 5.1: Summary of services. DM: Data-Medium rate, DH: Data-High rate, HV: High quality Voice and DV: Data+Voice.

The difference between DM and DH is that in DM an error correcting code of rate 2/3 is employed whereas in DH, the extra bits are used for additional data bits. DM3 (DM5) occupies 3 (5) consecutive frames to extend the payload size, thereby reducing the overhead from frame headers. The HV1-3 packet types have different payload sizes, and can be used to get up to three simultaneous voice connections. These packet types are never retransmitted in order to ensure streaming (voice) data. The DV packet provides both a voice and a data payload where only the data field can be retransmitted. An important observation, from the point of view of attacking the cipher in Bluetooth, is that the longest frame contains 2745 encrypted bits.

A Bluetooth session can be either a point-to-point connection, where only two Bluetooth units are engaged, or a point-to-multipoint connection, where two or more units are engaged. In a point-to-multipoint setting, each participating unit shares the same channel. This is called a *piconet*. In each piconet there must exist a single master unit, with all other units being slaves. The master is responsible for synchronisation and also defines the frequency hop sequence to which all slaves will listen. A Bluetooth unit can be engaged in several overlapping piconets and these piconets form a *scatternet*. Each unit can only be a master of one piconet at each time, but can participate as a slave in another piconet. Different piconets are not time nor frequency synchronised, instead defining their own hopping sequences.

5.2 Security in Bluetooth

As Bluetooth may be used for highly private and confidential communication, e.g. computer passwords, the system needs to provide basic cryptographic functionalities; Authentication and identification for ensuring the correct sender and receiver, and encryption to ensure privacy. This is provided through two cryptographic primitives. For authentication and identification, Bluetooth uses a block cipher (*SAFER*+) with a keysize of 128 bits. For link encryption, the stream cipher E_0 is used. Due to export regulations, the keysize of the encryption function is variable and configured during the manufacturing process, after which it cannot be changed. The key setup is a complex protocol depending on the type of connection, i.e. point-to-point or piconet, but in short it is derived as follows. Each device uses a PIN-code, which can be supplied to the device by the user, for example from a keypad, or can be supplied by higher layers, for example by the application layer as a result of a Diffie-Hellman key exchange. This PIN-code can be of variable length, from 1 to 16 bytes. In addition, each unit has an unique address, BD_ADDR (Bluetooth device address), which is a publicly known 48 bit value. The BD_ADDR can be obtained for each device by a query, either from the user or from another Bluetooth unit. When a Bluetooth unit is first used, it computes a *unit key* which is stored in non-volatile memory and is almost never changed.

Firstly, for a point-to-point communication setup, a 128 bit *initialisation key* is

derived in both units based on the PIN and (depending on the size of the PIN) the BD_ADDR of the claimant unit. This key is used for a few transactions to establish a new 128 bit key called the *link key* K_{link}. The reason for this step is that units with small memory resources are able to provide their own unit key as the link key, and thus save memory if they are engaged in several piconets. Then, a mutual authentication scheme is performed to ensure the correctness of the link key. From the link key, the *cipher key* K_c is derived. This key has a factory preset length, due to the export regulations. The link key is only used for authentication and is not as strictly regulated as the encryption keys, thus K_{link} is always 128 bits.

The stream cipher E_0

The cipher key K_c, together with a 48 bit BD_ADDR, a 128 bit publicly known random value, and the 26 least significant bits from the master clock are used as initial values for the link encryption algorithm, E_0. This is a stream cipher with LFSRs feeding a finite state machine (FSM). The state machine is an elaboration on the summation combiner introduced by Rueppel [103]. The binary output from the state machine is the keystream, which is xored to the plaintext to form the ciphertext. The cipher is pictured in Figure 5.1, where the numbers to the left of the LFSRs are the length of the registers.

Remark. The notation used in this chapter conforms to the one used in the specification [10], and might thus differ from the rest of this thesis.

The boxes labelled z^{-1} are delay elements holding two bits each. T_1 and T_2 are two different linear bijections over $\mathbb{F}_2^2$, $(x_1, x_0) \rightarrow (y_1, y_0)$ where $T_1(x_1, x_0) \rightarrow (x_1, x_0)$ and $T_2(x_1, x_0) \rightarrow (x_0, x_1 \oplus x_0)$. Let x_t^i denote the output from $LFSR_i$ at time t. The output from the keystream generator, z_t, can now be written as

$$z_t = x_t^1 \oplus x_t^2 \oplus x_t^3 \oplus x_t^4 \oplus c_t^0. \tag{5.1}$$

Furthermore, the following relations hold (Notations are from Figure 5.1).

$$s_{t+1} = (s_{t+1}^1, s_{t+1}^0) = \lfloor \frac{y_t + c_t}{2} \rfloor, \tag{5.2}$$

$$y_t = x_t^1 + x_t^2 + x_t^3 + x_t^4, \tag{5.3}$$

$$c_{t+1} = (c_{t+1}^1, c_{t+1}^0) = (s_{t+1}^1, s_{t+1}^0) \oplus T_1(c_t) \oplus T_2(c_{t-1}). \tag{5.4}$$

Since the addition in (5.2) and (5.3) is over the integers, we have the possible values $y_t \in \{0, 1, 2, 3, 4\}$ and $s_t \in \{0, 1, 2, 3\}$. Furthermore, (s_t^1, s_t^0) is the binary vector representation of s_t with the natural mapping $0 \rightarrow (0, 0)$, $1 \rightarrow (0, 1)$ et cetera. The four feedback polynomials used for the LFSRs are given in Table 5.2. To complete the operative description of the cipher, we note that the LFSR output, x_t^i, is not taken from the end of the shift register, but from the taps given in Table 5.2.

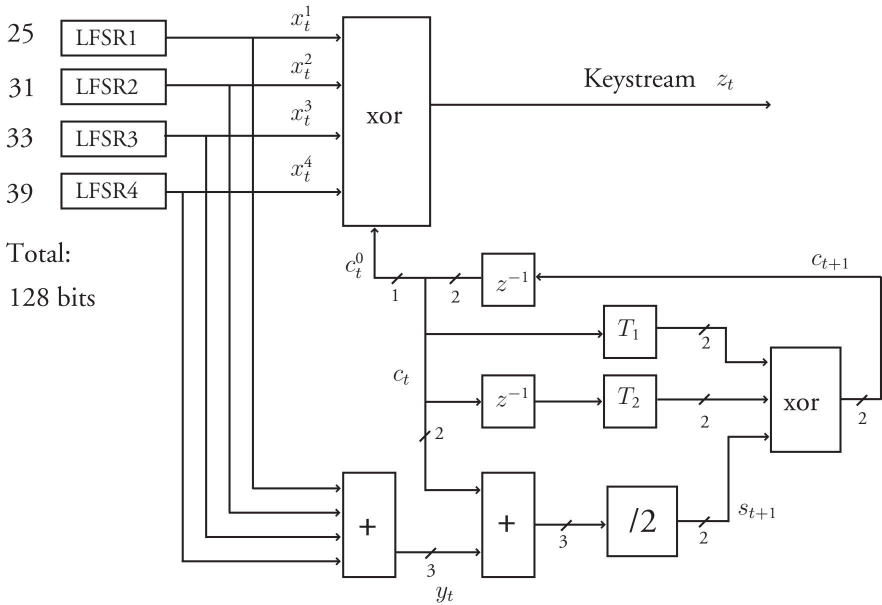


Figure 5.1: Bluetooth stream cipher E_0. The small numbers indicate the number of bits in the "wire".

LFSR	feedback polynomial	output tap
1	$t^{25} + t^{20} + t^{12} + t^8 + 1$	24
2	$t^{31} + t^{24} + t^{16} + t^{12} + 1$	24
3	$t^{33} + t^{28} + t^{24} + t^4 + 1$	32
4	$t^{39} + t^{36} + t^{28} + t^4 + 1$	32

Table 5.2: Feedback polynomials for the four LFSR in E_0.

The key initialisation in E_0 is somewhat more complicated and involves a premixing of the initially loaded key material, the details of which can be found in the Bluetooth specification documents [10]. However, it is important that the initial values of the LFSRs are dependent on the master clock, and that the registers are reinitialised and premixed for each frame. Two consecutive frames with little difference in the master clock will *not* generate initial states with little difference due to this premixing. To simplify matters, we will refer to the state of the LFSRs *after* this premixing as the

initial state. Recall that the largest frame contains 2745 encrypted bits, which is the longest keystream we can expect to see generated by a particular initial state.

5.3 Attacking E_0

Since the nonlinear function in E_0 has memory, we cannot directly apply the correlation attacks from Section 2.6, but need to remodel the cipher, using the well-known method of replacing the nonlinear part with a sequence of random variables having some correlation probability. For this, we start by searching for correlations in the c_t^0 sequence, which is the only nonlinear part of the keystream.

This approach to attacking E_0 is not new, but was first considered by Hermelin and Nyberg [59]. They discovered a correlation

$$P(c_t \oplus c_{t-1} \oplus c_{t-3} = 0) = \frac{1}{2} - 0.03125, \tag{5.5}$$

valid for all $t \geq 0$. By viewing the nonlinear function as a Finite State Machine (FSM), we can analyse all possible state transitions and find two stronger correlations in the c_t^0 sequence,

$$P(c_t \oplus c_{t-1} \oplus c_{t-2} \oplus c_{t-3} \oplus c_{t-4} = 0) = \frac{1}{2} - 0.04883, \tag{5.6}$$

$$P(c_t \oplus c_{t-5} = 0) = \frac{1}{2} + 0.04883. \tag{5.7}$$

These new correlations were independently discovered by Fluhrer [39]. Correlation probabilities are normally written as $\frac{1}{2} + \epsilon$, so in (5.7) we have $\epsilon = 0.04883$.

Our proposed attack is a divide-and-conquer style of attack, primarily targeting the initial state of LFSR1. Assume that we have a given received keystream z_t of length N. Firstly, we note that the three other LFSRs can be combined into a single equivalent LFSR. Denote the output sequence from this equivalent LFSR by u_t, $0 \leq t \leq (N-1)$. Furthermore, we assume that the sequence c_t^0 is a random noise sequence with the correlation property described by (5.7). We can now model a simplified version of E_0 as in Figure 5.2. Next, we try to guess the initial state of LFSR1, and add the presumed produced sequence, x'_t, to z_t. If the guess is correct, the resulting sequence can be written as

$$v_t = z_t + x'_t = u_t + c_t, \tag{5.8}$$

where we have dropped the 0 superscript of c_t. From coding theory [100] we know that the sequence $u_0, u_1, \ldots u_{N-1}$ is a linear (N, l)-block code $\mathcal{C}$, with generator matrix G, which is straightforward to calculate. The number of information symbols in $\mathcal{C}$ is l, which is equal to the length of the equivalent shift register, i.e. the sum of the length of LFSR2, LFSR3 and LFSR4. If we write the sequence u_t as a row vector

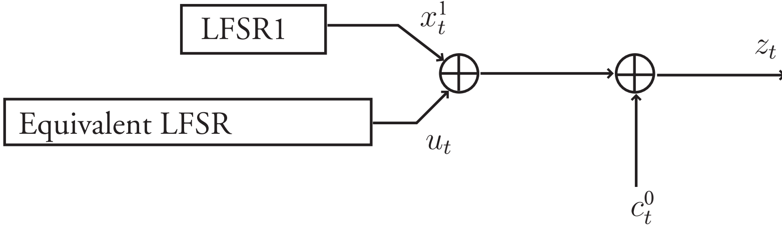


Figure 5.2: Simplified model of the E_0 stream cipher.

$\mathbf{u} = (u_0, u_1, \ldots, u_{N-1})$, we have $\mathbf{u} = \mathbf{u_0}G$, where $\mathbf{u_0}$ is the initial state of the equivalent shift register. Suppose we can find k columns in G such that

$$G_{i_1} + G_{i_2} + \cdots + G_{i_k} = 0, \tag{5.9}$$

then we must have $u_{i_1} + u_{i_2} + \cdots + u_{i_k} = 0$ for the sequence u_t. Since the code $\mathcal{C}$ is *cyclic*, we can write

$$\sum_{i \in \mathcal{I}} u_{t+i} = 0, \tag{5.10}$$

for any time index $t \geq 0$, where $\mathcal{I}$ is the set of indices in (5.9). We can now remove the influence of u_t in v_t by summing over indices in $\mathcal{I}$, indicated by (5.10). Considering the correlation in c_t given in (5.7), we can write

$$\sum_{i \in \mathcal{I}} v_{t+i} + v_{t+i-5} = (c_{t+i_1} + c_{t+i_1-5}) + (c_{t+i_2} + c_{t+i_2-5}) + \cdots + (c_{t+i_k} + c_{t+i_k-5}), \tag{5.11}$$

and we get

$$\begin{aligned} &P(\sum_{i \in \mathcal{I}} v_{t+i} + v_{t+i-5} = 0) = \\ &P((c_{t+i_1} + c_{t+i_1-5}) + (c_{t+i_2} + c_{t+i_2-5}) + \cdots + (c_{t+i_k} + c_{t+i_k-5}) = 0) = \\ &\qquad \frac{1}{2} + 2^{k-1}\epsilon^k. \end{aligned} \tag{5.12}$$

This equation is used to verify that the initial state of LFSR1 was indeed guessed correctly. If this is the case, the correlation in (5.12) can be detected, and if we *did not* guess LFSR1 correctly, we have only added more noise to the sequence v_t, and will not be able to detect any correlation. The model of the attack is pictured in Figure 5.3.

Depending on the magnitude of ϵ, we have to sample v_t according to (5.11) at many different time instances in order to get statistical significance in the hypothesis that LFSR1 was guessed correctly. If we guess right, the distribution of the sum in

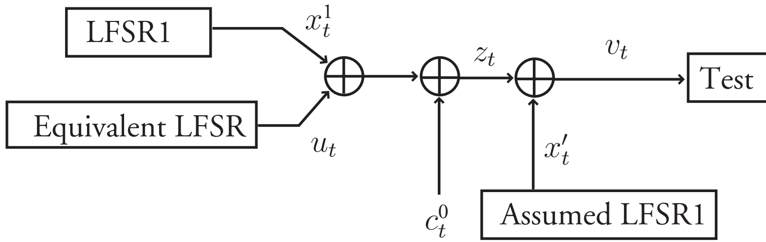


Figure 5.3: Model of the attack.

(5.11) tends to $\frac{1}{2} + 2^{k-1}\epsilon^k$, and if we are incorrect it tends simply to $\frac{1}{2}$, and another initial state of LFSR1 must be trialled.

Required length of the received sequence

The attack has two parameters that will influence the required length, N, of the received sequence z_t. Firstly, the value of the highest index in $\mathcal{I}$. This comes from the fact that we need to look at a span of z_t such that indices can be found that satisfy (5.10). Secondly, we need to shift (5.11) in time, say m shifts, to gain statistical significance. We will start with the first problem of finding an estimate of the highest index in $\mathcal{I}$. The problem is restated as a theorem for a random generator matrix.

Theorem 5.1: *Assume a cyclic code $\mathcal{C}$ with a random generator matrix* $\mathbf{G}$. *The total number of columns, w, in* $\mathbf{G}$ *required to find k columns that add to the all-zero column is approximately $2^{l/(k-1)}$, where l is the number of rows in* $\mathbf{G}$. □

A proof of Theorem 5.1 can be found in [44].

Even if the true generator matrix is not random, the above theorem gives a good approximation of the required length of the received sequence in order to find k columns that add up to the all-zero column.

We now turn to the other question of how many samples, m, we need in order to separate the uniform distribution $P_U(X = 0) = \frac{1}{2}$ from our indicator distribution $P_{E_0}(X = 0) = \frac{1}{2} + 2^{k-1}\epsilon^k$. From the theory of hypothesis testing in Section 3.2, we know that this is a well-studied problem and the distributions can be separated using approximately $1/(2^{k-1}\epsilon^k)^2$ samples.

Since $P_{E_0}(X = 0)$ gets closer to $1/2$ with increasing k, the Chernoff information $C(P_U, P_{E_0})$ is decreasing. Hence, the required number of samples, m, increases with increasing k, for a fixed error probability. From Theorem 5.1 we know that $w \approx 2^{l/(k-1)}$, so w decreases with increasing k. The total required number of observed keystream bits, N, is the sum $N = m + w$, and naturally we should choose k such

that we minimise N. We look at a brief example of attacking LFSR1, and compare the results for different k.

EXAMPLE 5.2: For the attack on the first shift register in E_0, we have $l = 103$ and $\epsilon = 0.04883$. Firstly we need $w = 2^{l/(k-1)}$ bits to find k columns in the generator matrix that add to zero. Aiming for an error probability of 2^{-6}, we then need $m = 6/C(P_U, P_{E_0})$ samples for the hypothesis testing. The results for different k are tabulated in Table 5.3. We see that $k = 4$ is the best choice for attacking LFSR1. □

k	m	w	$N = m + w$
3	$2^{25.2}$	$2^{51.5}$	$2^{51.5}$
4	$2^{32.0}$	$2^{34.3}$	$2^{34.6}$
5	$2^{35.3}$	$2^{25.8}$	$2^{35.3}$

Table 5.3: Different choices of k for attacking LFSR1 in E_0.

When performing the attack, we sum the sequence v_t according to (5.11) and count the number of times it equals zero. Let n_0 be the number of times it equals zero and n_1 be the number of times it equals one, and we have $m = n_0 + n_1$. Furthermore, let $\hat{\varepsilon} = 2^{k-1}\varepsilon^k$ such that we can write $P_{E_0} = 1/2 + \hat{\varepsilon}$. According to the Neyman-Pearson lemma, we perform the test between the two hypotheses $H_0 : P_U$ and $H_1 : P_{E_0}$ as

$$\frac{(\frac{1}{2})^m}{(\frac{1}{2} + \hat{\varepsilon})^{n_0}(\frac{1}{2} - \hat{\varepsilon})^{n_1}} > T, \tag{5.13}$$

with decision threshold $T \geq 0$. If we choose $T = 1$, and use $k = 4$ and $m = 2^{32}$, we will have equally high probability for the two error events, $P_F = P_M = 2^{-6}$. However, we could instead use an unsymmetrical threshold and decrease P_F at the expense of P_M. It can be shown [24], that the probabilities of error can be expressed as

$$P_M = 2^{-mD(P_\lambda || P_{E_0})}, \tag{5.14}$$

$$P_F = 2^{-mD(P_\lambda || P_U)}, \tag{5.15}$$

where P_λ is the probability distribution on the boundary between the two decision regions determined by T and $D(P_0||P_1)$ is the *relative entropy* defined as

$$D(P_0||P_1) = \sum_{x \in \aleph} P_0(x) \log_2 \frac{P_0(x)}{P_1(x)}. \tag{5.16}$$

The boundary distribution P_λ is determined by the chosen threshold such that

$$D(P_\lambda || P_{E_0}) - D(P_\lambda || P_U) = \frac{\log_2 T}{m}. \tag{5.17}$$

The exact value of the threshold is not crucial, but we would like to have $P_F << P_M$. Since the length of LFSR1 is 25, and if we choose a threshold such that $P_F \approx 2^{-10}$ we will then get about 2^{15} false alarms, i.e. we have reduced the number of secret bits in LFSR1 from 25 to 15. Having P_M of the same small order would cost too much in terms of required observed keystream, therefore we use an unsymmetrical threshold of $T = 25$, resulting in $P_M \approx 2^{-4}$.

The results for attacking the full E_0 are presented in Table 5.4.

LFSR	k	m	w	N	T	P_F	P_M	Comp. complexity
1	4	2^{32}	2^{34}	2^{34}	25	2^{-10}	2^{-4}	$2^{25}2^{32} = 2^{57}$
2	4	2^{32}	2^{24}	2^{32}	25	2^{-10}	2^{-4}	$2^{31}2^{32} = 2^{63}$
3	3	2^{28}	2^{19}	2^{28}	30	2^{-32}	2^{-28}	$2^{33}2^{28} = 2^{61}$
4	-	-	-	-	-	-	-	$2^{39}2^{4} = 2^{43}$

Table 5.4: Computational complexity of attacking the E_0 stream cipher. All exponents are rounded to the nearest integer.

For LFSR3, the error probabilities are very low because $C(P_U, P_{E_0})$ is relatively large, and hence we can allow a longer received sequence, giving a very low error probability. The attack on LFSR4 is not carried out as on the other LFSRs, since we only need to guess the state of LFSR4 together with the 4 bits in the FSM, and compare the resulting keystream with the observed one.

The full attack requires approximately 2^{34} bits in received length and the computational complexity is about 2^{63}. However, the primary target for this attack is the initial state of LFSR1, and after that is found, one might (as indicated in [51]) find stronger correlations in the FSM, and thus increase the performance of the attack for the other shift registers.

Finally, we note that the required length of the observed keystream is much larger than the largest frame used in Bluetooth. Thus, we cannot hope to apply this attack on the actual Bluetooth encryption scheme.

5.4 Previous and newer attacks on E_0

The first attack on E_0 was presented 1999 by Hermelin and Nyberg [59]. They discovered a weaker correlation than presented here, and used that in an attack similar

to that presented in this thesis. Their attack can recover the initial state of the shift registers with a given keystream length of 2^{64} and a computational complexity of 2^{64}.

The attack discussed in this chapter was presented in 2000 [30] and more recent proposals include the following attacks.

In 2001, Fluhrer and Lucks [40] presented an attack which does not only recover the initial state of the shift register, but also reverses the premixing step, and thus the attack can recover the session key K_c. This is done within the available amount of keystream, using several frames. The computational complexity is between 2^{76} and 2^{84}, depending on the amount of known keystream material. Their attack starts by guessing the initial state of LFSR1 and LFSR2 together with the state of the FSM. By observing the keystream, they can then decide whether the xor of the outputs from LFSR3 and LFSR4 is one or zero and build a search tree with linear equations. Whenever there is an inconsistency in the path, they backtrack and try a different branch.

Golić, Bagini, and Morgari [51] presented an attack based on linear correlations between the output sequence of the LFSRs and the keystream sequence. A correlation conditioned on a known LFSR1 sequence is used, together with decoding techniques, to obtain a computational complexity of 2^{70} for recovering the session key K_c. The attack can be carried out with a very short observed keystream, i.e. within the limits of the Bluetooth frame length.

In a very recent unpublished report, Armknecht [2] proposes an algebraic approach to solve for the initial state of E_0. Armknecht identifies an equation of degree 4, which holds with probability 1 at each clocking. By linearisation, the system becomes solvable, assuming that enough independent equations can be collected. From the size of the state space, it is concluded that the number of possible terms in the linearised system is $T \approx 2^{24.056}$ and by employing Strassen's algorithm for solving systems of linear equations, the complexity of this approach is concluded to be about $2^{70.341}$. It is also conjectured in [2] that, in order to get enough independent linear equations, the number of observed keystream bits must be approximately $T = 2^{24.056}$, but the question is regarded as an open problem.

5.5 Summary

In this chapter we have given a brief introduction to the Bluetooth wireless network system and the services it can provide in terms of information transfer rates. We have presented the stream cipher E_0, used in Bluetooth for link encryption, and given the operational details for keystream generation.

We have demonstrated a strong correlation within the nonlinear output sequence from the FSM, found by examining all possible FSM state transitions of depth 8. This correlation has then been used in a divide-and-conquer style correlation attack, targeting the three smallest LFSRs. The attack needs about 2^{34} known keystream bits

and requires a computational complexity of about 2^{63} to fully recover the initial state of E_0.

Further work on Bluetooth might include a deeper analysis of the FSM correlations if one or two registers are known. Also, a super-tuned time-memory tradeoff attack could be considered, similar to the one performed by Biryukov, Shamir, and Wagner [8] on $A5/1$, discussed in Chapter 4.

6

Cryptanalysis of the shrinking generator

The shrinking generator (SG) is a well-known pseudo random keystream generator, proposed in 1993 by Coppersmith, Krawczyk, and Mansour [18]. It is intended for use in stream cipher applications and is of interest due to its conceptual simplicity. It combines only two LFSRs in a very simple way. One of the LFSRs controls the output of the other, and the produced keystream is an irregularly decimated version of a maximum-length sequence.

Denote the two independent binary LFSRs by A and S. The pseudo-random bits are produced by shrinking the output sequence of the generating LFSR A under the control of the selecting LFSR S as follows. The output bit of LFSR A is taken if the current output bit of LFSR S is 1, otherwise it is discarded. It is recommended in [18] that, besides the initial states, the feedback polynomials are also to be defined by the secret key and thus kept unknown to the attacker. The SG is pictured in Figure 6.1.

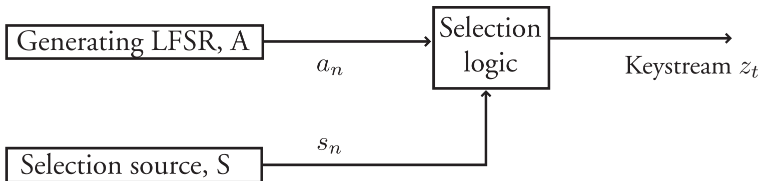


Figure 6.1: Model of the shrinking generator.

The attack discussed in this chapter is a known-plaintext distinguishing attack and it was first presented in [36]. The attack exploits a newly detected non-randomness in the distribution of output blocks in the generated keystream. The attack requires the feedback polynomial of A to be known, thus it is not directly applicable to the SG if secret feedback polynomials are used.

To describe the approach, denote by *a-stream* the output sequence generated by LFSR A and by *z-stream* the output sequence generated by the SG. The bits in the a-stream are denoted $a_n, n \geq 0$ and the bits in the z-stream are denoted $z_t, t \geq 0$. Rather than single bits, bit strings (blocks) in the a-stream are considered and compared with suitable strings in the z-stream.

Consider a block of odd length, centred at the position of a_n in the a-stream. If this block is xored with similar blocks (of equal length) centred at all other tap positions in the LFSR-recursion, including the feedback position, the sum is (trivially) the all-zero block. For blocks of odd length, the majority bit is set to 1 if the number of ones is larger than the number of zeroes, and 0 otherwise. Then a key observation is that the majority bits of such blocks fulfil the linear recursion of LFSR A with a probability larger than 1/2.

Through the shrinking process, the exact positions of the undiscarded a-stream bits in the z-stream are lost. However, the deletion rate is 1/2, and we can guess the average *shrunken tap positions* by halving the distances of the tap positions in the a-stream.

The main idea of our attack is to consider samples in the z-stream where the blocks near the shrunken tap positions are all imbalanced (a block is imbalanced if it has a different number of ones than zeroes). The bits in these blocks have, with high probability, been generated by bits in the neighbourhood of the original tap positions. If a z-block (i.e. a block in the z-stream) with high imbalance is found, the probability that the corresponding a-block is imbalanced, is quite high. By estimating the imbalance in the a-blocks using the measured imbalance in the z-blocks, we can estimate the majority bit of the a-blocks.

If these estimated majority bits fulfil the linear recursion of LFSR A with a probability larger than 1/2, we can distinguish the z-stream generated by the SG from a truly random sequence. Theoretical estimates as well as extensive experiments have shown that we are able to reliably distinguish the SG from random for low-weight recursions of LFSR A. For example, for a weight 4 recursion of length 10000, the attack needs approximately 2^{32} output bits, and for a weight 3 recursion of length 40000, the attack needs approximately 2^{23} output bits to reliably distinguish the SG.

As the feedback polynomial of an arbitrary LFSR of length l is known to have a polynomial multiple of weight 4 and length about $2^{l/3}$ [13, 44, 66, 118], our distinguisher also applies to arbitrary, shrunken LFSR-sequences of moderate length.

This chapter is organised as follows. In Section 6.1 the proposed attack is presented in more detail. In Section 6.2 the probability of success and required length of observed keystream is analysed. Simulation results are presented in Section 6.3 and an overview of related work is given in Section 6.4. In Section 6.5 a summary of the chapter is

given and finally, some technical details of the analysis are presented in Section 6.6.

6.1 Description of the attack

For the description of the attack, we assume the feedback connection of the generating LFSR A to be known and given by the linear recurrence

$$a_n + a_{n+n_2} + a_{n+n_3} + \ldots + a_{n+n_W} = 0, \quad n \geq 0, \tag{6.1}$$

whereas the selecting sequence can be any random sequence with independent and equally distributed bit probabilities. See Figure 6.2.

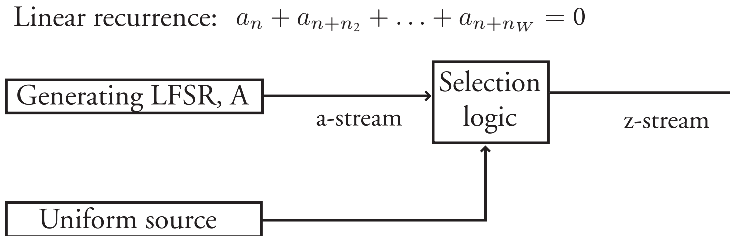


Figure 6.2: Model of the shrinking generator used in the attack.

We do not distinguish the feedback position from the other tap positions, so for a certain bit a_n, the tap positions are $n, n+n_2, n+n_3, \ldots, n+n_W$. Now, consider a bit string (block) surrounding a position a_n in the a-stream, i.e. a string $(a_{n-k}, \ldots, a_n, \ldots, a_{n+k})$ for some k. If we xor this block with the blocks (of equal length) surrounding the other positions in the LFSR recurrence equation, $a_{n+n_2}, \ldots,$ a_{n+n_W}, the sum (trivially) is the all-zero block. If we choose the considered block length to be odd, each of the W blocks must have a unique *majority bit*. The majority bit is 1 if the number of ones in the block is larger than the number of zeros and vice versa. The main observation is that the majority bits of such blocks fulfil the linear recurrence equation of the LFSR with a probability larger than $1/2$. Firstly, we formally introduce the intuitive notion of *imbalance*.

Definition 6.1: *The imbalance of a block B, $Imb(B)$, is defined as*

$$Imb(B) = \textit{number of ones in } B - \textit{number of zeros in } B$$

Furthermore, a block B is said to be imbalanced if $Imb(B) \neq 0$.

The general idea is to search for imbalanced blocks in the z-stream at positions $z_{t_1}, z_{t_1+n_2/2}, \ldots, z_{t_1+n_W/2}$. These positions are called the shrunken tap positions,

where t_1 implicitly is the reference position. The bits in these blocks have, with high probability, been generated by bits in the neighbourhood of the unshrunken tap positions, $a_{n_1}, a_{n_1+n_2}, \ldots, a_{n_1+n_W}$, where w.l.o.g. we assume $a_{n_1} = z_{t_1}$. If we find a block with high imbalance in the z-stream, then the probability that the corresponding block in the a-stream is imbalanced is quite high. Then we can derive good approximations of the true majority bits in the a-blocks.

The attack has two phases. The first phase is a search for positions in the output sequence, where we find imbalanced blocks centred around each of the shrunken taps. Whenever we find such a position, we say we have a **hit** and invoke the second phase, which estimates the majority bits of the unshrunken segments. We then count the number of times the xor sum of the estimated majority bits equals zero, and compare with the truly random case.

First phase

Pick a block B_1 of odd length $BL_1 = E + 1$ centred around a reference position z_{t_1}, where E is an *even* parameter of the attack. This mean that we have

$$B_1 = (z_{t_1-E/2}, \ldots, z_{t_1}, \ldots, z_{t_1+E/2}), \tag{6.2}$$

and as we assume $z_{t_1} = a_{n_1}$, the bits in B_1 come from bits surrounding a_{n_1}.

The next unshrunken tap $a_{n_1+n_2}$ is, with high probability, mapped to an interval near $t_1 + n_2/2$. This interval size grows proportionally to $\sqrt{n_2}$. The same holds for the other taps. Thus at tap position $n_j, j = 2, \ldots, W$, we measure on a block centred at position $t_1 + n_j/2$ of length $BL_j \approx BL_1 + \sqrt{n_j}/2$. The $\approx$ symbol denotes here: "take the closest odd integer", since we need an odd length for having a unique imbalance.

Next, we measure the imbalance in each block, $Imb(B_j), j = 1, \ldots, W$. Whenever we have $|Imb(B_j)| > T, j = 1, \ldots, W$, where T is an imbalance threshold, we have found a *hit*, and it is likely that the a-blocks surrounding the unshrunken taps are also imbalanced. If we do not find imbalanced z-blocks at the chosen reference position we pick a new z_{t_1} and again measure the imbalance. However, if we have a *hit* in phase 1, we invoke the second phase.

Second phase

The first goal of the second phase is to try to estimate the bit probability of the a-blocks. Firstly we introduce some notations. Let S_1 be the a-block of length L_a surrounding a_{n_1}. Similarly, we denote by $S_j, j = 2, \ldots, W$ the a-block of length L_a surrounding $a_{n_1+n_j}$. Denote by $p_j, j = 1, \ldots, W$ the estimated probability that a bit in S_j equals 1.

Since we assumed that the centre bit $z_{t_1} = a_{n_1}$, we will use the first block as a reference. The first z-block is of length $E + 1$. We denote this reference block length

by $L_z = E + 1$. For our considerations we can assume that the z-block B_1 of odd length was produced by an odd length a-block. Thus we choose the a-block length as $L_a = 2L_z - 1 = 2E + 1$.

If we measure the imbalance $Imb(B_1)$ in B_1, and assume that the bits which where discarded are balanced, i.e. the $L_a - L_z = E$ bits not visible as output bits are equally distributed, we have the following estimate

$$p_1 = \frac{\text{number of ones in } S_1}{L_a} = \frac{(Imb(B_1) + L_z)/2 + E/2}{L_a} \tag{6.3}$$

$$= \frac{1}{2} + \frac{Imb(B_1)}{4E+2}. \tag{6.4}$$

For the other $W - 1$ taps we proceed in a slightly different way. Since we are not sure which z-block contains the most bits from S_j, we must consider several z-blocks centred near $z_{t_1+n_j/2}$. Thus, we pick an interval $\mathcal{I}_j$ and calculate a weighted average over $\mathcal{I}_j$, of the bit probability in S_j as

$$p_j = \frac{1}{2} + \sum_{k \in \mathcal{I}_j} \frac{imb_{j,k}}{4E+2} P(k \text{ is the best position for estimating } S_j), \quad j = 2 \ldots W, \tag{6.5}$$

where $imb_{j,k}$ is the imbalance in the z-block of length L_z surrounding $z_{t_1+n_j/2+k}$. The expression for $P(k$ is the best $\ldots)$ is discussed in Section 6.6, but the aim is to try to derive a probability that the z-block at $t_1 + n_j/2 + k$ has the most bits from S_j, and we assume that the binomial distribution

$$P(k \text{ is the best position for estimating } S_j) = Bin[n_j, 0.5](\frac{n_j}{2} + k), \tag{6.6}$$

is adequate. Figure 6.3 shows a picture of the blocks used in the weighting process for a small interval of $k \in [-1, 0, 1]$.

In (6.5), we have used similar calculations as in (6.3) with the additional assumption that the bit probability is 0.5 if the best position, k, is outside the interval $\mathcal{I}_j$.

The estimated bit probability $p_j, j = 1, \ldots, W$ also represents an estimate of the majority bit for S_j. For each hit we have in phase 1 of the attack, we can determine an estimate of the xor sum of the majority bits. The simplest decision rule is to first make a hard decision on the estimated majority bit $\hat{m}_j$ as

$$\hat{m}_j = \begin{cases} 1 & \text{if } p_j \geq \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}, \quad j = 1, \ldots, W, \tag{6.7}$$

then check if the xor sum $\sum_j \hat{m}_j = 0$ holds. By dividing the number of times the xor sum equals zero with the number of hits, we can derive a measured final probability that the sum of the true majority bits equals zero.

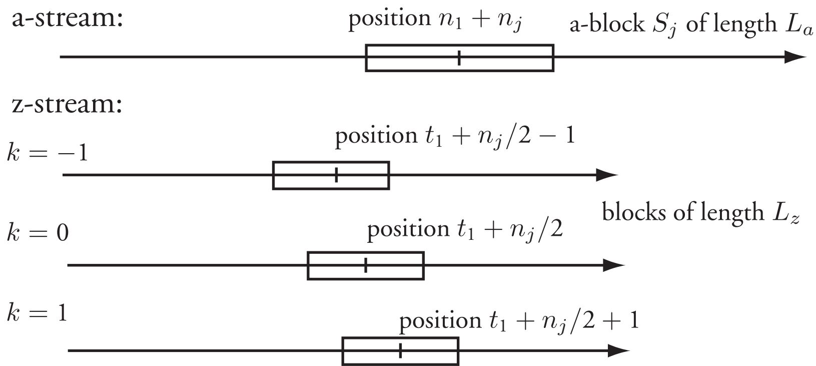


Figure 6.3: Diagram showing the blocks used in the weighting process for estimating $p_j, j = 2, \ldots, W$. The interval $\mathcal{I}_j$ shown here only includes 3 positions.

We can also employ a soft decision rule. It is well-known [61] that a soft decision rule is no worse than a hard decision rule, so employing a soft decision rule can only improve the attack. The hard decision rule, however, is simpler to analyse, and for that reason we have used a hard decision rule in the analysis and in our simulations.

If we are attacking the SG, we expect the total probability to equal

$$P = 0.5 + \varepsilon_H, \tag{6.8}$$

where ε_H is a positive value depending on the number of hits we obtain in phase 1, H. If we are attacking a truly random sequence, we expect $P = 0.5$. The optimal test to distinguish between these cases is a Maximum Likelihood (ML) test [24], where we use a threshold Γ such that if $P \geq \Gamma$ we decide the output sequence is generated from the SG, otherwise we decide it is a random source. We have summarised the proposed attack in Figure 6.4, where we have used a threshold Γ equal to approximately one standard deviation from the expected value for a truly random sequence.

Input: even integer E, imbalance threshold T, the tap positions $n_j, j = 1, \ldots, W$, where $n_1 = 0$, and a sequence $Z_t, t = 1, \ldots, N$ of received bits.

(i) Setup variables.

$$\begin{aligned} BL_1 &= E + 1, \\ BL_j &= \text{ closest odd integer } (BL_0 + \sqrt{n_j}/2), \quad j = 2, \ldots, W, \\ \mathcal{I}_j &= \text{ integers in } [-\sqrt{n_j}/3, \ldots, \sqrt{n_j}/3], \quad j = 2, \ldots, W, \\ low &= E/2, high = N - (n_W/2 + \sqrt{n_W}/3 + E/2), \\ hits &= 0, good = 0. \end{aligned}$$

(ii) For $t_1 = low$ to $high$ do

(a) For all $j = 1, \ldots, W$, let B_j be the z-block centred at $t_1 + n_j/2$ of length BL_j.

(b) Let $imb_j = Imb(B_j), j = 1, \ldots, W$.

(c) If $\forall j \; |imb_j| > T$ then

i. Increase $hits$ and set

$$\begin{aligned} p_1 &= \frac{1}{2} + \frac{imb_1}{4E+2}, \\ p_j &= \frac{1}{2} + \sum_{k \in \mathcal{I}_j} \frac{imb_{j,k}}{4E+2} \binom{n_j}{n_j/2 + k} \left(\frac{1}{2}\right)^{n_j}, j = 2, \ldots, W, \end{aligned}$$

where $imb_{j,k}$ is the measured imbalance in the z-block of length BL_1 centred at position $t_1 + n_j/2 + k$.

ii. Make a hard decision $\forall j, \hat{m}_j = 1$ if $p_j \geq 0.5$ otherwise $\hat{m}_j = 0$.

iii. if $\sum_j \hat{m}_j = 0$ increase $good$. ($\sum$ denotes the xor sum here.)

(iii) Set $\Gamma = 0.5 + \sqrt{hits}/(2 \cdot hits)$.

(iv) If $good/hits > \Gamma$ return **Shrinking** else return **random**.

Figure 6.4: A summary of the proposed attack using a hard decision rule.

6.2 Analysis of the proposed attack

We start by considering the probability that the majority bits of $S_j, j = 1, \ldots, W$ fulfil the recurrence equation. We will be using imbalance and Hamming weight (or

simply weight) interchangeably, as they measure similar quantities, where the weight of a block B is the number of ones in B. For a block length of L we have the conversion between the imbalance imb and the weight hw as

$$imb = 2hw - L,$$
$$hw = \frac{imb + L}{2}.$$

Furthermore, we will use the notation $w_H(B)$ to denote the Hamming weight of the block or string B.

The probability distribution of the sum of the majority bits

Assume we look at the direct output of an LFSR with W taps (including the feedback position). Pick a reference position a_n in the a-stream and consider the positions $a_{n+n_2}, \ldots, a_{n+n_W}$ that together with a_n sum to zero, according to the linear recurrence equation of the LFSR. At each position we take a centred block (or vector) S_j of odd length L_a. The positions and the length are assumed to be chosen such that the blocks are non-overlapping. The aim is to calculate the probability that the majority bits of the segments fulfil the recurrence equation of the LFSR. We assume the distribution of the possible vectors at tap $j, j = 1, \ldots, W-1$, only to be dependent on the weight of the vector. The vector at the final tap is totally determined by the choices at the $W-1$ previous taps since the xor of the vectors must be the all-zero vector. Introduce the notation

$$V_j(\alpha) = P(w_H(S_j) = \alpha), \tag{6.9}$$

for the probability that we have a vector of length L_a and weight α at tap $j, j = 1, \ldots, W-1$.

We can partition the possible vectors at the taps into sets of equal weight. The probability of each set is given by $V_j(\alpha)$. When we xor the first two vectors, we get a new probability distribution on the sets of different weights. In these sets, the majority bit of the vector sum may or may not agree with the sum of the majority bits of the xored vectors. So when we derive the new distribution, we also keep track of when the sum of the constituent majority bits agree with the majority bit of the resulting vector. When we have xored the $W-1$ vectors, we know that the final vector *must* agree with the sum of the first $W-1$ vectors. Thus we can sum the probabilities where the sum of the constituent majority bits agree with the majority bit of the resulting vector.

The technical details of the calculations can be found in Section 6.6. We only conclude the validity of the approach by comparing the theoretical probability to a number of simulations for different block length L_a with $W = 4$. The comparison is shown in Table 6.1. The number of trials in each simulation was $10,000,000$.

L_a	Simulations	Theory
5	0.56082 0.56050	0.56054
11	0.52254 0.52169	0.52207
17	0.51372 0.51298	0.51340
29	0.50764 0.50727	0.50748

Table 6.1: Results from two simulations versus theory for the probability that the majority bits sum to zero.

Skewing the distributions V_j

Since in phase 1 of the attack we impose the condition that the absolute value of the imbalance in the z-block should be greater than a certain threshold T, the distributions V_j for the a-blocks will not be binomial, as expected from a random source. As the probabilities $P(Imb(B_j) > T) = 0.5$ and $P(Imb(B_j) < -T) = 0.5$ are equally likely, we divide the distribution V_j into two parts

$$V_j(\alpha) = \frac{V_j^+(\alpha) + V_j^-(\alpha)}{2}, \quad j = 1, \ldots, W, \tag{6.10}$$

where

$$V_j^+(\alpha) = P(w_H(S_j) = \alpha | Imb(B_j) > T), \tag{6.11}$$

$$V_j^-(\alpha) = P(w_H(S_j) = \alpha | Imb(B_j) < -T), \tag{6.12}$$

and $w_H(S_j)$ is the weight of the a-block S_j.

We cannot hope to give an exact expression for (6.11) and (6.12) since there are too many interdependencies between adjacent $imb_{j,k}$ in (6.5). We can however derive approximate expressions. The derivation can be found in Section 6.6. Intuitively, and also from the derived approximations, we note that V_j^+ and V_j^- are "mirrored" distributions in the sense that $V_j^+(\alpha) = V_j^-(L_a - \alpha)$.

An approximation of the required number of hits

Firstly, we consider the probability that the estimated majority bit $\hat{m}_j$ is correct. We will confine ourselves to the case of a hard decision rule. Through simulations we have noted that these probabilities are not independent. The first $W - 1$ majority bits are more often correctly predicted if all or most of the $W - 1$ a-blocks are strongly imbalanced. But then also the Wth block tends to have some imbalance and thus gives

a better prediction of the majority bit. In the analysis however, we will assume that they are independent.

Assume without loss of generality that $m_j = 1$. Then we know that the vectors of S_j are drawn (on the average) from the distribution V_j^+. The probability P_{m_j} that m_j is correct is determined by the probability mass that gives a majority bit equal to 1. We have

$$P_{m_j} = \sum_{\alpha=(L_a+1)/2}^{L_a} V_j^+(\alpha), \quad j = 1, \ldots, W. \tag{6.13}$$

The same holds if we have $m_j = 0$ as a result of the mirrored properties of V_j^+ and V_j^-.

Let m_j be the true majority bit of the a-block S_j, and as before, let $\hat{m}_j$ be the hard decision estimate as derived in the attack. We have $P(m_j = \hat{m}_j) = P_{m_j}$. We can model this as if we have a noisy observation of the true majority bits, where the noise for each observation is equal to 0 with probability P_{m_j}. Furthermore, let P_n denote the probability that noise variables sum to zero. Using the independency assumption and the *piling-up-lemma* [79], we have

$$P_n = \frac{1}{2} + 2^{W-1} \prod_{j=1}^{W} (P_{m_j} - 0.5). \tag{6.14}$$

The correctness of the estimates $\hat{m}_j, j = 1, \ldots, W$ is assumed independent of the probability $P(\sum_j m_j = 0)$. Recalling (6.8) and introducing the notation P_M for the probability that the true majority bits sum to zero, we have an approximation for the total probability P, that the estimated majority bits sum to zero of

$$P = \frac{1}{2} + 2(P_M - 0.5)(P_n - 0.5) = \frac{1}{2} + \varepsilon_H. \tag{6.15}$$

Using similar arguments as in Section 3.4 (also derived in [17]) we can state an upper bound on the number of hits H we need in phase 1 to distinguish the SG from a truly random sequence as

$$H \leq \frac{1}{\varepsilon_H^2}. \tag{6.16}$$

A lower bound on the expected number of hits

In phase 1 of the attack, we search for positions in the z-stream where we find W blocks $B_j, j = 1, \ldots, W$ of length $BL_j, j = 1, \ldots, W$, such that we simultaneously have $|Imb(B_j)| > T$. If we again make the simplifying assumption that the imbalance (or Hamming weight) of the blocks are independent, we can derive a lower bound on the expected number of hits in phase 1, given a received sequence of length N from the generator.

The probability P_{B_j} that we have $|Imb(B_j)| > T$ for a block of length BL_j where we assume the keystream bits from the generator to be independent and random, is given by

$$P_{B_j} = \sum_{h=1+\frac{T+BL_j}{2}}^{BL_j} \binom{BL_j}{h} \left(\frac{1}{2}\right)^{BL_j} + \sum_{h=0}^{\frac{-T+BL_j}{2}-1} \binom{BL_j}{h} \left(\frac{1}{2}\right)^{BL_j} . \quad (6.17)$$

The approximated joint probability of finding W blocks is given by

$$P_H = \prod_{j=1}^{W} P_{B_j} \quad (6.18)$$

and thus we would expect

$$H \geq P_H N. \quad (6.19)$$

Parameter trade-offs

If we choose a large value for the parameter E we will have larger blocks B_j and the probability for the bit a_{n_j} to be mapped inside B_j increases. On the other hand the a-blocks will also be larger, resulting in a smaller probability for the true majority bits to sum to zero. If we only choose the lengths BL_j to be larger, then the distribution for the smaller block that is scanned in (6.5) tends to be closer to the binomial distribution, which gives us less non-randomness from which to extract information.

If we choose a high imbalance threshold T, the distributions of V_j will be more skewed and we will need a fewer number of hits to distinguish the sequence. But on the other hand, the probability for getting a hit in phase 1 decreases, thus requiring a longer received sequence.

6.3 Simulation results

In this section we present some simulation results and do a comparison with the derived theoretical approximation. In the attack we have used the hard decision rule and a decision threshold $\Gamma = 0.5 + \sqrt{hits}/(2 \cdot hits)$, which corresponds to approximately one standard deviation from the expected value if the sequence were truly random. We start by attacking a weight 4 LFSR given by the linear recurrence equation

$$a_n + a_{n+302} + a_{n+703} + a_{n+1000} = 0. \quad (6.20)$$

Using a threshold $T = 3$ and block size parameter $E = 14$, the theoretical calculations in (6.15) and (6.17) give $P_H = 0.02648$ and $\varepsilon_H = 0.00092$. Thus we would need about $H = 1/\varepsilon_H^2 \approx 2^{20}$ hits in phase 1 and $N = H/P_H \approx 2^{25}$ received output bits.

Running the attack with these configurations 50 times (with random initial state), we can distinguish the SG in all 50 cases.

As the degree of the feedback polynomial increases, the attack naturally needs a larger received sequence, and it becomes impractical to simulate the attack. Also, the approximations made in the analysis tend to overestimate the required N, as experiments with shorter sequences have shown. We have summarised some results from attacking weight 4 feedback polynomials in Table 6.2. We also give two examples of attacking an SG with weight 3 and weight 5 feedback polynomials in Table 6.3.

Tap positions	Theoretical parameters				N used	Successes out
(excluding 0)	P_H	ε_H	H	N	in attack	of 50 runs
$302, 733, 1000$	0.02648	$2^{-10.1}$	$2^{20.2}$	$2^{25.4}$	2^{23}	43
					2^{24}	46
					2^{25}	50
$812, 1433, 2500$	0.03586	$2^{-11.5}$	$2^{23.0}$	$2^{27.8}$	2^{25}	39
					2^{26}	46
					2^{27}	50
$2333, 5847,$ 8000	0.05542	$2^{-13.5}$	$2^{27.0}$	$2^{31.2}$	2^{28}	42
					2^{29}	48
					2^{30}	50
$3097, 6711,$ 10000	0.05989	$2^{-13.9}$	$2^{27.7}$	$2^{31.8}$	2^{28}	45
					2^{29}	45
					2^{30}	46

Table 6.2: Theoretical and simulation results from attacking the SG with various weight 4 feedback polynomials

The computational complexity of the attack is quite modest. If we use precomputed tables for (6.6), we see that we need to scan the input sequence once and whenever we have a *hit* we calculate (6.5). The size of the interval $\mathcal{I}_j$ is proportional to $\sqrt{n_j}$ and hence we have the computational complexity $O(N\sqrt{n_W})$, where N is the number of received output bits and n_W is the last tap position or the degree of the feedback polynomial.

Tap positions (excluding 0)	Theoretical parameters				N used in attack	Successes out of 50 runs
	P_H	ε_H	H	N		
$17983, 40000$	0.1414	$2^{-10.2}$	$2^{20.3}$	$2^{23.1}$	2^{21}	36
					2^{22}	46
					2^{23}	50
$73, 131,$ $219, 300$	0.0068	$2^{-11.56}$	$2^{23.1}$	$2^{30.3}$	2^{29}	48
					2^{30}	50

Table 6.3: Theoretical and simulation results from attacking the SG with weight 3 and weight 5 feedback polynomials.

6.4 Related work

Several approaches for attacking the SG have been proposed. In the original paper on the SG [18], Coppersmith, Krawczyk and Mansour considered two basic divide-and-conquer attacks. The first attack assumes known feedback polynomials for both LFSRs, A and S, of degree l_A and l_S respectively. By guessing the initial state of S, the selection sequence from S is recovered and consequently a (non-consecutive) sequence of a-stream bits can be obtained from the keystream output. The resulting system of linear equations for the initial state of A can then be solved in polynomial time. In the second attack, the feedback polynomials are assumed unknown and the attack starts by guessing the feedback polynomial and initial state of S. From the known keystream of N bits, it is then possible to obtain the so-called *product sequence* $p_n = s_n a_n,\ 0 \leq n \leq N$, which is shown in [104] to have a linear complexity of at most $l_A l_S$. By observing $N = 2l_A l_S$ bits, the complete product sequence can be constructed. This information is sufficient to reconstruct the a-stream, a_n, and hence both the initial state of A and the feedback polynomial for A. The complexity of this second attack is exponential in l_S but polynomial in l_A.

In 1994, Golić and O'Connor presented a probabilistic correlation attack on general clock-controlled generators [53]. Their approach is to consider the joint probability of an a-stream sequence $a_n,\ n \geq 0$ and the shrunken sequence $z_t\ t \geq 0$. The joint probability is considered for every initial state of A that could possibly generate the shrunken sequence z_t. For a certain number of observed bits, which is shown to be dependent on the channel capacity for a deletion channel, the correct initial state of A will lead to the highest joint probability and hence a recovery is possible. The approach in [53] was later experimentally analysed, using the SG, by Simpson, Golić and Dawson in [114]. Their analysis assumes a known feedback polynomial for A and it is stated that the attack in [53] needs approximately $20 \cdot l_A$ bits of observed keystream and a computation complexity of $2^{l_A} l_A^2$ to recover the initial state of A.

In 1995, Golić presented another technique [43]. In this paper, several attacks on a general keystream generator based on irregularly clocked shift registers are discussed. The one most relevant to this chapter, is the distinguisher based on a statistical weakness in the output sequence. The attack is based on the probability that the properly decimated linear recurrence is fulfilled in the decimated sequence. The paper also discusses a method of reconstructing an unknown feedback polynomial using this statistical weakness. The ideas from this paper were applied to the shrinking generator in [46, 50], both authored by Golić. The distinguishing attack presented there compares nicely to the one discussed in this chapter. From Table 6.2 we recall that our distinguisher needs approximately 2^{32} bits to distinguish the SG with a weight 4 polynomial of degree $10,000$ and from Table 6.3 we recall that we need approximately 2^{23} bits to distinguish the SG with a weight 3 polynomial of degree $40,000$. The distinguisher in [46] needs approximately 2^{48} and 2^{39} output bits respectively for these cases.

In 1998, Johansson [63] presented a reduced complexity correlation attack based on a suboptimal decoding procedure for the deletion channel. The attack searches for specific substrings in the output sequence such that the posteriori probabilities for the bits in the a-stream are highly biased. These probabilities are then used in the decoding algorithm to recover the initial state of LFSR A. However, the attack is still exponential in the length of LFSR A.

Very recently, in an unpublished paper by Golić and Menicocci [52], another distinguishing attack on the SG was presented. Similarly to the ideas presented in this chapter, their attack calculates the probability of the bits in the a-stream conditioned on the observed bits in the z-stream. Then, a hard decision is made on the bits in the a-stream and the bit values are used to check the validness of the (unshrunken) linear recurrence equation. A chi-square statistical test is performed on the number of valid/invalid instances and the distinguisher is successful if the statistics deviate sufficiently from the random case. This approach gives results comparable to ours. From experiments, it can be shown that our approach, using the majority bit, is superior if the weight is 3 or 4, and comparable to their attack if the weight of the feedback polynomial is 5. As stated previously, our theoretical analysis tends to overestimate the required length of the keystream while their theoretical analysis seems to be more accurate with respect to experimental data.

6.5 Summary

A practical distinguishing attack on the shrinking generator with a low-weight feedback polynomial for the generating LFSR has been proposed. The attack is based on the new observation that the majority bits of blocks in the LFSR stream fulfil the linear recursion with a probability larger than $1/2$ and a powerful method of estimating these bits, based on the received keystream, has been presented and analysed.

Approximate expressions for the required number of observed keystream symbols are given, based on an upper bound on the required number of *hits* obtained from phase 1 of the attack. Also, a lower bound on the expected number of *hits*, based on input parameters to the attack, is derived. Simulations of the attack are given to verify the derived bounds.

It can also be noted that the first segment need not be taken as the reference segment. A tap in the middle of the feedback polynomial could be used instead. The probabilities are then calculated both forward and backward in the a-stream and the probabilities of the estimates being correct would thus increase.

The proposed attack can also be used to predict the distribution of bits in the generated sequence. Assume it is known that the generating source is the SG. Firstly, the bit probabilities of the $W-1$ first segments are calculated. Then using the theoretical calculations of the probability that the majority bits sum to zero, the unknown distribution of the last segment can be derived.

6.6 Technical details of the analysis

In this section some of the more technical details of the analysis of the attack are given.

Some details of phase 2

The interval size in (6.5) is a tunable parameter of the attack and we have chosen it to be the integers in the range $[-\sqrt{n_j}/3, \ldots, \sqrt{n_j}/3]$. Next we consider the probability that the z-block centred at $z_{t_1+n_j/2+k}$, $k \in \mathcal{I}_j$, is the best block to choose. An exact mathematical expression for this notion is hard to find because of the deletion process. To simplify the calculations we assume that this probability is given by the binomial distribution, $Bin[n_j, 0.5](n_j/2+k)$. This assumption does not take into account the fact that the bit $a_{n_1+n_j}$ might not be visible (printed) in the z-stream at all, since it might be deleted. However, we can disregard whether the bit is printed or not since we are only trying to estimate the surroundings of $a_{n_1+n_j}$. Thus, the assumed distribution is adequate, and we write

$$P(k \text{ is the best position for estimating } S_j) = \tag{6.21}$$

$$P(k \text{ is best}) =$$

$$Bin[n_j, 0.5](\frac{n_j}{2}+k) = \binom{n_j}{\frac{n_j}{2}+k}\left(\frac{1}{2}\right)^{n_j} \tag{6.22}$$

We note that the interval $\mathcal{I}_j$ is chosen such that $\sum_{k\in\mathcal{I}_j} P(k \text{ is best}) \approx 0.5$.

Probability that the sum of the majority bits equals zero

We will assume that the bits in the first $W-1$ vectors in the LFSR case can be approximated by the truly random case. Given this assumption, the probability V_j is given by the binomial distribution

$$V_j(\alpha) = Bin[L_a, 0.5](\alpha), \quad i = 1, \dots, W-1. \tag{6.23}$$

When xoring vectors together, we will denote the distribution of vectors (for the sum) by $Q_j(\alpha)$. We will add a superscript to this Q later, but for now we say that $Q_1(\alpha) = V_1(\alpha), \alpha = 0 \dots L_a$, since we have not added any vectors but the first. $Q_2(\alpha)$ will be the probability of a vector of weight α when we have xored two V distributions. We also introduce an operator to determine the majority bit. Let $Maj(\alpha)$ denote the majority bit of a vector of length L_a and weight α. We have

$$Maj(\alpha) = \begin{cases} 1 \text{ if } \alpha > (L_a - 1)/2, \\ 0 \text{ otherwise.} \end{cases}$$

The next lemma states the possible values of the weight and the number of different values obtained, when xoring two vectors together.

Lemma 6.2: *Let A be a fixed vector of length L_a and weight α. If we xor all possible vectors (one at a time) of weight β with A, then the possible weights of the xor sum are $\gamma = \alpha - \beta + 2\kappa$ where $max(0, \beta - \alpha) \leq \kappa \leq min(L_a - \alpha, \beta)$. The number of resulting vectors with weight γ is given by*

$$\binom{\alpha}{\beta - \kappa}\binom{L_a - \alpha}{\kappa}. \tag{6.24}$$

□

Proof. Denote by $\mathbf{B}$ the set of all vectors with weight equal to β. Assume κ of the ones in $\mathbf{B}$ coincide with the zeros of A. This implies that $\beta - \kappa$ of the ones in $\mathbf{B}$ coincide with the ones of A. The number of such vectors in $\mathbf{B}$ is

$$\binom{\alpha}{\beta - \kappa}\binom{L_a - \alpha}{\kappa}.$$

For the choice of κ we must have $0 \leq \beta - \kappa \leq \alpha$ or equivalently $\alpha - \beta \leq \kappa \leq \beta$, since at least zero and at most α of the ones can coincide. Similarly, we must have $0 \leq \kappa \leq L_a - \alpha$, since the number of zeros in A is $L_a - \alpha$. Combining these restrictions we get $max(0, \alpha - \beta) \leq \kappa \leq min(L_a - \alpha, \beta)$. The resulting weight of the xor sum is $\gamma = \alpha - (\beta - \kappa) + \kappa = \alpha - \beta + 2\kappa$, which proves the lemma. □

Thus, if we randomly pick *one* vector B of weight β, from a uniform distribution (over the set of vectors with weight β) and xor with the fixed vector A, we have the

probability

$$P_\gamma(\alpha, \beta, \kappa) = \frac{\binom{\alpha}{\beta - \kappa}\binom{L_a - \alpha}{\kappa}}{\binom{L_a}{\beta}}, \tag{6.25}$$

of obtaining a resulting vector of weight $\gamma = \alpha - \beta + 2\kappa$, where $max(0, \beta - \alpha) \leq \alpha \leq min(L_a - \alpha, \beta)$.

When xoring two vectors, we must keep track of whether the xor of the majority bits agrees with the majority bit of the sum. For example, we can get the vector 11110 by

Vector	Weight	Maj. bit		Vector	Weight	Maj. bit
11000	2	0	or by	11100	3	1
⊕00110	2	0		⊕00010	1	0
11110	4	1		11110	4	1

We can determine this condition by checking if $Maj(\alpha) \oplus Maj(\beta) = Maj(\gamma)$, and we will denote the corresponding probability mass as different variables, $Q_j^0(\alpha)$ for when the sum of the constituent majority bits agrees with the majority bit of the sum, and $Q_j^1(\alpha)$ for when it does not. One can think of the superscript as a parity bit for making the sum of the constituent majority bits and the resulting majority bit equal to zero.

Let $M(\alpha, \beta, \gamma) = Maj(\alpha) \oplus Maj(\beta) \oplus Maj(\gamma)$. Recalling (6.25) and the fact that we defined $Q_1^0(\alpha) = V_1(\alpha), \alpha = 0, \ldots, L_a$ we have for the sum of $j = 2, \ldots, W - 1$ vectors

$$Q_j^0(\gamma) = \sum_{Cond(\alpha,\beta,\kappa)} P_\gamma(\alpha, \beta, \kappa) Q_{j-1}^{M(\alpha,\beta,\gamma)}(\alpha) V_j(\beta), \tag{6.26}$$

$$Q_j^1(\gamma) = \sum_{Cond(\alpha,\beta,\kappa)} P_\gamma(\alpha, \beta, \kappa) Q_{j-1}^{1 \oplus M(\alpha,\beta,\gamma)}(\alpha) V_j(\beta), \tag{6.27}$$

where $Cond(\alpha, \beta, \kappa)$ determines the summation conditions according to

$$Cond(\alpha, \beta, \kappa) = \begin{cases} \forall \alpha, \beta, \kappa : \\ \alpha - \beta + 2\kappa = \gamma, \\ max(0, \beta - \alpha) \leq \kappa \leq min(L_a - \alpha, \beta). \end{cases} \tag{6.28}$$

In Q_{W-1}, we have the distribution of all but the last tap position vector. If we know that the sequence comes from an LFSR, we know that the last vector must force the total vector sum to the all-zero vector. Thus we find the probability that the xor of the majority bits (m_j) fulfils the recurrence equation to be

$$P_M = P(\sum_{j=1}^{W} m_j = 0) = \sum_{\alpha=0}^{L_a} Q_{W-1}^0(\alpha). \tag{6.29}$$

These are the probabilities for the vectors that *will* have 0 as "majority parity bit", thus fulfilling the recurrence equation. The derived probability P_M is used in (6.15).

Approximations of V_j^+ and V_j^-

Next, we derive the approximations of the probabilities $V_j^+(x)$ and $V_j^-(x)$, for $j = 1, \ldots, W$, starting with $V_j^+(x)$. Recall the definition

$$V_j^+(x) = P(w_H(S_j) = x | Imb(B_j) > T). \tag{6.30}$$

Since the analysis is independent of the actual reference position $z_{t_1} = a_{n_1}$, we will simplify the notation. The position $n_1 + n_j$, the jth tap position for the reference bit a_{n_1}, will simply be denoted n_j and the value of that bit is denoted a_{n_j}. We start by conditioning on the assumption that a_{n_j} (or at least its near surrounding) is visible inside the z-block B_j. If a_{n_j} falls outside B_j we do not get any information on the weight of S_j and must assume it to be binomially distributed. We have

$$\begin{aligned} P(w_H(S_j) = x | Imb(B_j) > T) = \\ & P(w_H(S_j) = x | Imb(B_j) > T, a_{n_j} \text{ in } B_j) P(a_{n_j} \text{ in } B_j) + \\ & (1 - P(a_{n_j} \text{ in } B_j)) Bin[L_a, 0.5](x). \end{aligned} \tag{6.31}$$

In the attack, we "slide" a block of length L_z in the z-stream, where the centre position $t_1 + n_j/2 + k$, is within an interval $\mathcal{I}_j$, cf. (6.5). Let K_k denote the block at centre position $t_1 + n_j/2 + k$. We have noted experimentally that the distribution of the weight of K_k is almost independent of k, except at the end points of the interval, where it does not change to a great degree. Thus, we drop the subscript of K_k and refer to its distribution as $P(w_H(K) = y)$. With this approximation we say that the first probability in the right hand side of (6.31), conditioned on a_{n_j} being within B_j, is only dependent on the probability of the weight of K. Hence we write

$$P(w_H(S_j) = x) = \sum_{y=0}^{L_z} P(w_H(S_j) = x | w_H(K) = y) P(w_H(K) = y), \tag{6.32}$$

where we have dropped the additional conditions $Imb(B_j) > t$ and a_{n_j} in B_j, for readability.

For $j = 2, \ldots, W$, the probability $P(a_{n_j} \text{ in } B_j)$ in (6.31) is approximated by the probability that we have $n_j/2 - (B_j - 1)/2$ or more, but $n_j/2 + (B_j - 1)/2$ or less, visible symbols in the z-stream at a-stream position n_j. We write

$$P(n_j \text{ in } B_j) = \sum_{v=\frac{n_j}{2}-\frac{B_j-1}{2}}^{\frac{n_j}{2}+\frac{B_j-1}{2}} \binom{n_j}{v} \left(\frac{1}{2}\right)^{n_j}, \quad j = 2, \ldots, W. \tag{6.33}$$

For $j = 1$ we know (by assumption) that $z_{t_1} = a_{n_1}$ and we have the probability $P(a_{n_1} \text{ in } B_1) = 1$.

Next, we consider the second probability on the right hand side of (6.32). Recalling the implicit condition $Imb(B_j) > T$, and the approximation that $w_H(K)$ is independent of the centre position, we state this probability as: *The probability that a random substring—of length L_z, chosen at random, in a random string B_j of length BL_j—has $w_H(K) = y$, given $Imb(B_j) > T$.*

Let w_H^T be the equivalent weight of the imbalance threshold T. It can be shown that

$$P(w_H(K) = y | w_H(B_j) > w_H^T) = \frac{\sum_{b=w_H^T+1}^{BL_j} F(y,b)}{\sum_{b=w_H^T+1}^{BL_j} \binom{BL_j}{b}}, \tag{6.34}$$

where

$$F(y,b) = \begin{cases} \binom{BL_j - L_z}{b-y}\binom{L_z}{y} & \text{if } max(0, b-(BL_j - L_z)) \leq y \leq min(L_z, b), \\ 0 & \text{otherwise .} \end{cases} \tag{6.35}$$

We proceed by evaluating $P(w_H(S_j) = x | w_H(K) = y)$, where we consider K to be the "best" sub-block in B_j, i.e. the sub-block with the most bits from the a-block S_j of length L_a, surrounding n_j. But, even if K is assumed to be the best sub-block, not all of the bits in K need to come from S_j. Thus, we also have to calculate the average weight of the bits not coming from S_j. Let d be the number of bits not coming from S_j (from either end of K), and let w_H^d denote their weight. We need to have at least one bit from S_j in K, since we assume K to be the best sub-block. Thus we can write

$$P(w_H(S_j) = x | w_H(K) = y) = \sum_{d=0}^{E} P(w_H(S_j) = x | d \text{ bits not from } S_j, w_H(K) = y) P(d \text{ bits not from } S_j), \tag{6.36}$$

where

$$P(w_H(S_j) = x | d \text{ bits not from } S_j, w_H(K) = y) = \\ \sum_{m=0}^{d} P(w_H(S_j) = x | w_H^d = m, d \text{ bits not from } S_j, w_H(K) = y) P(w_H^d = m). \tag{6.37}$$

Now, given $w_H^d = m$ we know that $y - m$ ones in K come from S_j and since $L_a = 2L_z - 1 = 2E + 1$ we have $E + d$ bits from S_j not printed in K. Hence, the probability $P(w_H(S_j) = x| \ldots)$ equals the probability of finding $x - (y - m)$ ones in $E + d$ samples from a random source

$$P(w_H(S_j) = x| \ldots) = \begin{cases} Bin[E + d, 0.5](x - (y - m)) & \text{if } x \geq y - m, \\ 0 & \text{otherwise.} \end{cases} \tag{6.38}$$

Since $w_H(K) = y$, the probability that $w_H^d = m$ is the probability that the weight of d random bits from K equal m and is given by

$$P(w_H^d = m) = Bin[d, \frac{y}{L_z}](m). \tag{6.39}$$

Finally, we need the approximation of $P(d \text{ bits not from } S_j)$. Since bits could be dropped at either end, but not more than $(L_z - 1)/2$ at one end, we can write

$$P(d \text{ bits not from } S_j) = \\ \sum_{\substack{d_1 + d_2 = d \\ 0 \leq d_1, d_2 \leq (L_z - 1)/2}} P(d_1 \text{ bits dropped left}) P(d_2 \text{ bits dropped right}), \tag{6.40}$$

where we use the approximation

$$P(d_1 \text{ bits dropped at an end}) = \begin{cases} \sum_{v=(L_z-1)/2}^{L_z-1} Bin[E, 0.5](v) & \text{if } d_1 = 0, \\ Bin[E, 0.5](\frac{L_z-1}{2} - d_1) & \text{if } d_1 > 0. \end{cases} \tag{6.41}$$

The derivation of the approximation for $V_j^-(x)$ is similar, but the condition for the imbalance is $Imb(B_j) < -T$, and (6.34) becomes

$$P(w_H(K) = y | w_H(B_j) < w_H^T) = \frac{\sum_{b=0}^{w_H^T - 1} F(y, b)}{\sum_{b=0}^{w_H^T - 1} \binom{BL_j}{b}}, \tag{6.42}$$

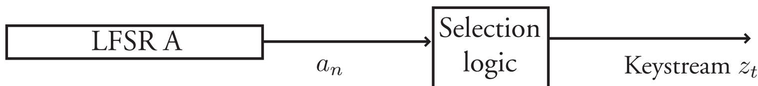
where w_H^T now is the equivalent Hamming weight of the imbalance $-T$.

The derived approximations for $V_j^+(x)$ and $V_j^-(x)$ are used in (6.13) to calculate the theoretical estimates of the attack complexity shown in Table 6.2 and 6.3.

7

Cryptanalysis of the self-shrinking generator

The shrinking generator, presented in Chapter 6, was refined in 1994 by Meier and Staffelbach [84] to an even more elegant construction; the self-shrinking generator (SSG). The two LFSRs from the shrinking generator are combined into a single LFSR, and the bits in the produced sequence are divided into pairs. The first bit of each pair is a *selection bit* that determines if the second bit, *the active bit*, is to be taken as an output keystream bit or discarded. The single LFSR in the SSG acts as both the selection source S and the generating source A in the shrinking generator. The SSG is pictured in Figure 7.1. From the selection logic we note that, on average, 3 out of



Selection logic:

$$\text{if} \quad (a_{2i}, a_{2i+1}) = \begin{cases} (1,1) & \text{output} \quad 1, \\ (1,0) & \text{output} \quad 0, \\ (0,1) & \text{discard} \quad a_{2i+1}, \\ (0,0) & \text{discard} \quad a_{2i+1}. \end{cases}$$

Figure 7.1: Model of the self-shrinking generator.

4 bits produced by the LFSR are discarded. Similarly to the shrinking generator, the secret key defines the initial state of the LFSR. It is also recommended in [84] that the feedback polynomial is determined from the secret key. An interesting observation made in [84] is that the SSG and the shrinking generator are equivalent in functionality. Each implementation of the SSG can be considered a special case of the shrinking generator and vice versa.

Very little of the previous cryptanalysis on the SSG has been concerned with the case where the feedback polynomial is unknown. The best known approach for this case is simply to guess the feedback polynomial and then apply one of the possible attack methods on the SSG with known feedback polynomial. In this chapter, it will be demonstrated that for a certain class of very weak feedback polynomials, the output sequences from the SSG can be distinguished from random sequences very efficiently (in polynomial time). It will be shown that for such a weak feedback polynomial, there also exists an efficient key recovery attack. Finally, a more general class of weak feedback polynomials is analysed and a distinguishing attack on the SSG, with a known feedback polynomial from this class, will be presented. The results in this chapter have previously been presented in [35].

This chapter is organised as follows. In Section 7.1 some related work is discussed. In Section 7.2 the first class of weak polynomials is defined and both the distinguishing attack and the initial state recovery attack are presented. Then, in Section 7.3 the more general class of weak polynomials is given and a distinguishing attack on the SSG for this class is presented. Some implementation aspects of the attack are discussed in Section 7.4 together with simulation results. Finally, in Section 7.5, a summary of the chapter is given.

7.1 Related work

Besides presenting the SSG, Meier and Staffelbach also presented a cryptanalysis of the new generator [84]. They observed that each bitpair has a biased probability given the keystream output. For example, the a-stream pair $(a_0, a_1) = (1, \tilde{z}_0)$ cannot occur given the keystream output $z_0, z_1, \ldots$. Here, $\tilde{z}_0$ denotes the bitwise complement of z_0. Furthermore, $(a_0, a_1) = (1, z_0)$ occurs with probability 0.5, whereas $(a_0, a_1) = (0, 0)$ and $(a_0, a_1) = (0, 1)$ both occur with probability 0.25. Continuing this line of argument, it is shown in [84] that the entropy of the initial state, of a length l SSG, is $0.75l$. Thus, the optimal guessing strategy for the initial state is to start with the most probable state and check if that guess was correct. If not, proceed with the second most probable state, et cetera. This approach gives an average computational complexity of $O(2^{0.75l})$.

In 1996, Mihaljevic presented a time-data trade-off attack [86]. The attack starts by assuming that a segment of $m < l/2$ keystream bits has been produced by a certain state of the LFSR. Consequently, m of the selection bits in the LFSR need to be equal

to 1. The attacker proceeds by guessing the positions of those ones and then searching over all m bit substrings in the keystream output, trying to find a position where the assumption is correct. The computational complexity of the attack varies from $O(2^{0.5l})$ up to $O(2^{0.75l})$, under less favourable circumstances. The required length N of the observed keystream varies from $2^{0.5l}$ to $2^{0.25l}$ respectively. The choice of the parameter m depends on the number of available keystream bits, and it is shown that the following relation must hold in order for the attack to be successful

$$N > m2^{l/2}\binom{l/2}{m}^{-1}.$$

In 2001, Zenner, Krause and Lucks [122] presented an attack on the SSG based on ideas from [18]. Their method is to guess bits, one at a time, in combination with the observed keystream output, and build a search-tree. Whenever they reach a condition which falsifies earlier guesses, they backtrack in the tree and make another guess. The computational complexity of this approach is $O(2^{0.69l})$ and it needs a very short length of observed keystream bits. This attack was later improved by Krause [74] using a technique called *Binary Decision Diagrams* (BDD). Krause further reduced the complexity to $O(2^{0.64l})$, also using a very short observed keystream sequence.

All previously known attacks assume a known feedback polynomial and the computational complexity is exponential in the length l of the LFSR. We will now present a new approach that is very efficient for a certain class of weak feedback polynomials.

7.2 A first class of weak polynomials

Before the attacks are discussed, we specify the attack model and introduce some notation for the attacks. Assume an SSG with a weight W feedback polynomial. The sequence produced by the LFSR is called the *a-stream* and is denoted $a_n,\ n \geq 0$. The keystream output is called the *z-stream* and is denoted $z_t,\ t \geq 0$. The attacks are known plaintext attacks so the keystream output $z_t,\ 0 \leq t \leq N$, is assumed to be known for some N. The attack model is pictured in Figure 7.2. Every second bit pro-

Linear recurrence: $a_n + a_{n+n_2} + \ldots + a_{n+n_W} = 0$

Figure 7.2: Model of the self-shrinking generator used in the attack.

duced by the LFSR is called a *selection bit* and has an even index. These bits determine

if the odd indexed bits, the *active bits*, are to be discarded or used as keystream bits, according to the previously described self-shrinking mechanism. The selection bits are assumed to be independent and to have a uniform probability distribution.

The first goal is to derive a distinguishing attack on the SSG with a weight 3 feedback polynomial of the form $1 + x^{l-1} + x^l$, with odd l. This means that we have a linear recurrence equation of the form

$$a_{n-1} + a_n = a_{n+l-1}, \quad n \geq 1. \tag{7.1}$$

If l is odd, then whenever a_n is an active bit, so is a_{n+l-1}. Assume that a_n, for some index n, is an active bit and is mapped to the output z_t for some t, denoted by $a_n \to z_t$. Then, $a_{n-1} = 1$ and we have

$$1 + z_t = a_{n+l-1}, \tag{7.2}$$

where z_t is a known observed bit from the z-stream. If a_{n+l-1} appears in the z-stream (i.e. $a_{n+l-2} = 1$), it must appear as one of the symbols $z_{t+1}, \ldots, z_{t+(l-1)/2}$ with a probability given by the binomial distribution of the $(l-3)/2$ selection bits in the interval (a_{n+l-2} is assumed to be 1 and thus excluded). The most probable position, $t + k$ (in the z-stream), for a_{n+l-1} is $k = \lfloor (l-3)/4 \rfloor + 1$. We write this probability as

$$\begin{aligned} P(a_{n+l-1} \to z_{t+k}) &= P(a_{n+l-2} = 1) \cdot P(a_{n+l-1} \to z_{t+k} | a_{n+l-2} = 1) \\ &= \frac{1}{2} \cdot \binom{(l-3)/2}{k-1} 2^{-(l-3)/2}. \end{aligned} \tag{7.3}$$

If we guess the position k correctly, we will have $z_t + z_{t+k} = 1$ with probability 1. If we guess incorrectly, the relation will hold with probability $1/2$. Thus, we have identified an expected correlation for this SSG as

$$\begin{aligned} P(z_t + z_{t+k} = 1) &= P(a_{n+l-1} \to z_{t+k}) \cdot 1 + (1 - P(a_{n+l-1} \to z_{t+k})) \cdot \frac{1}{2} \\ &= \frac{1}{2} + \frac{1}{2} P(a_{n+l-1} \to z_{t+k}) = \frac{1}{2} + \varepsilon. \end{aligned} \tag{7.4}$$

Recalling Example 3.2 in Section 3.2, we know that we need about $1/\varepsilon^2$ samples to distinguish the correlation in (7.4).

Let us give a brief example to determine the magnitudes of the expected correlations.

Example 7.1: Assume a feedback polynomial given by $1+x^{62}+x^{63}$. Therefore $l = 63$ and $k = (63-3)/4 + 1 = 16$. The probability in (7.3) is calculated to be

$$P(a_{n+62} \to z_{t+16}) = \frac{1}{2} \cdot \binom{30}{15} 2^{-30} = 0.0722,$$

and the correlation in (7.4) is calculated to be

$$P(z_t + z_{t+16} = 1) = \frac{1}{2} + \frac{1}{2}0.0722 = \frac{1}{2} + 0.536.$$

Thus, we need about $1/(0.036)^2 \approx 2^{10}$ bits of observed keystream to distinguish this particular SSG. Let us compare this result with known attacks. Even though the feedback polynomial is weak, the best key recovery attacks proposed in previous literature are still of order $2^{0.64l}$ which in this case is about 2^{40}. □

In Figure 7.3 the correlation probability (7.4) is plotted as a function of the length of the LFSR, together with $1/\varepsilon^2$ for the corresponding correlation. Note that the

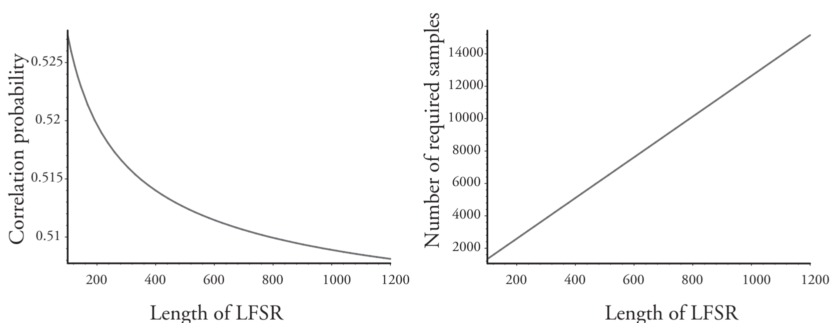


Figure 7.3: Left figure: Correlation probability, $1/2+\varepsilon$, given by (7.4) with $k = (l-3)/2+1$. Right figure: The required number of samples to distinguish the distribution, $1/\varepsilon^2$.

required length of the observed keystream sequence grows linearly with the length of the LFSR, making it much more efficient than previously known attacks, where the growth is exponential. However, the method only works for this class of weak polynomials of weight 3, with two taps sitting together.

An initial state recovery attack

To avoid an abundance of notation, we will choose a specific feedback polynomial as we now turn to focus on an initial state recovery attack. Therefore, assume a feedback polynomial $g(x) = 1 + x^{126} + x^{127}$, corresponding to the relation

$$a_n + a_{n+1} = a_{n+127}. \tag{7.5}$$

We start by squaring the feedback polynomial, in order to obtain some additional

relations, for example, $g(x)^2$ and $g(x)^4$ give

$$a_n + a_{n+2} = a_{n+254}, \quad \text{and} \tag{7.6}$$
$$a_n + a_{n+4} = a_{n+508}, \tag{7.7}$$

respectively. In the next step we construct the set of possible values for the first 20 values of the a-stream. To simplify the analysis, we accept an error probability in our attack and assume that the first 20 values of a_n produce at least 5 output bits (i.e. at least five of the ten even selection bits in a_n are ones). Introduce a set $\mathcal{L}$, containing all values of $(a_0, a_1, \ldots, a_{19})$ that are possible given the beginning of the z-stream $z_0, z_1, \ldots$.

Each element in $\mathcal{L}$ will give a corresponding sequence for the a-stream $(a_{127}, a_{128}, \ldots, a_{145})$ of length 19, through the recurrence relation (7.5). This will in turn give rise to a shrunken sequence that must appear somewhere in the z-stream. As the distance in (7.5) is 127, we can expect the position of this shrunken sequence to be about 31 steps ahead. If we do not find the shrunken sequence in the neighbourhood of position 31, we remove the corresponding value from $\mathcal{L}$, thereby reducing the size of $\mathcal{L}$. The same procedure is done for relations (7.6) and (7.7), reducing the size of the set $\mathcal{L}$ even further.

Next, we add all possible values (a_{20}, a_{21}) to the remaining elements in $\mathcal{L}$ and hence the size of $\mathcal{L}$ grows. Again, we test the z-stream for the resulting shrunken sequence according to (7.5), (7.6) and (7.7), and remove those values in $\mathcal{L}$ that do not support our observed keystream. This reduces the size of $\mathcal{L}$ again, et cetera.

The case of unknown feedback polynomial

The previously presented distinguisher and initial state recovery attack only work for the specified weak polynomial of weight 3, with two taps sitting together. This is of course a limitation, but the approach could also be interesting in a more general setting.

The SSG is proposed to be used with a secret feedback polynomial, i.e. the secret key determines which primitive polynomial is to be used, as well as the initial state of the LFSR. As mentioned before, the previously known attacks either assume a known feedback polynomial or determine it by guessing. It is a well-known fact that there exist roughly $2^l/l$ primitive polynomials of length l. So, the computational complexity of an attack assuming a known feedback polynomial thus increases with a multiplicative factor of approximately 2^l, resulting in an (asymptotic) complexity of $O(2^{1.64l})$ for the best general attack.

For our attack, we proceed as follows.

(*i*) Find a weak feedback polynomial of length l, or weak multiples of several of the primitive feedback polynomials.

(*ii*) Examine $2^l/l$ encryptions with different keys and apply the distinguishing attack.

(*iii*) Apply the initial state recovery attack when a weak polynomial (or multiple) has been found.

The proposed attack scheme uses $O(l \cdot 2^l)$ known plaintext bits encrypted under different keys. Its computational complexity is only $O(l \cdot 2^l)$ and hence, the complexity of the best known attack on this version of the SSG is decreased from $O(2^{1.64l})$ to $O(l \cdot 2^l)$.

7.3 A general class of weak polynomials

In this section, a more general class of weak feedback polynomials is examined and a distinguishing attack on the SSG using such a polynomial is presented. To make the notation more readable, we will describe the linear recurrence from the LFSR by its characteristic polynomial rather than the feedback polynomial. Recall from Section 2.2 that the characteristic polynomial is the reciprocal of the feedback polynomial.

We start by assuming an SSG with a characteristic polynomial (or a multiple of the characteristic polynomial) of the form

$$\begin{aligned} g(x) &= g_1(x) + x^M g_2(x), \quad \text{where} \\ g_1(x) &= c_0 + c_1x^1 + c_2x^2 + \ldots + c_kx^k, \\ g_2(x) &= d_0 + d_1x^1 + d_2x^2 + \ldots + d_kx^k, \end{aligned}$$

for binary constants $c_i, d_i,\ i = 0, \ldots, k$. The polynomials $g_1(x)$ and $g_2(x)$ have degree $\leq k$, for some small fixed integer k. This corresponds to a linear recurrence relation of the form

$$\sum_{i=0}^{k} c_i a_{n+i} = \sum_{i=0}^{k} d_i a_{n+M+i}. \tag{7.8}$$

We now introduce blocks (or vectors) of a certain size $B_a > k$, containing consecutive symbols from the a-stream. Let

$$(a_n, a_{n+1}, \ldots, a_{n+(B_a-1)}), \tag{7.9}$$

denote such a block in the a-stream. By calculating the left hand side of (7.8), starting at different positions in (7.9), we construct the following block

$$\mathbf{h}_1 = (\sum_{i=0}^{k} c_i a_{n+i}, \sum_{i=0}^{k} c_i a_{n+1+i}, \ldots, \sum_{i=0}^{k} c_i a_{n+(B_h-1)+i}), \tag{7.10}$$

of length $B_h = B_a - k$. Similarly, for the right hand side of (7.8) but now starting at the a-stream position a_{n+M}, we construct the following block

$$\mathbf{h}_2 = (\sum_{i=0}^{k} c_i a_{n+M+i}, \sum_{i=0}^{k} c_i a_{n+M+1+i}, \ldots, \sum_{i=0}^{k} c_i a_{n+M+(B_h-1)+i}), \tag{7.11}$$

of the same length B_h. Clearly, due to (7.8), we have

$$\mathbf{h}_1 + \mathbf{h}_2 = \mathbf{0},$$

where $\mathbf{0}$ is the zero block of length B_h.

Now, $\mathbf{h}_1$ and $\mathbf{h}_2$ are unknown quantities, but through the keystream output sequence, an estimate can be calculated. For a fixed t, we know that $a_{n+1} \rightarrow z_t$ for some even n, implying that the selection bit $a_n = 1$. We conclude that the sequence $z_t, z_{t+1}, \ldots$ provides some information about $(a_n, a_{n+1}, \ldots, a_{n+(B_h-1)})$ and thus about $\mathbf{h}_1$. This information is collected by calculating the probability

$$P(\mathbf{h}_1 | z_t, z_{t+1}, \ldots). \tag{7.12}$$

Similarly, the keystream output symbols around the $(t + M/4)$th position will provide some information about $(a_{n+M}, a_{n+M+1}, \ldots, a_{n+M+(B_h-1)})$ and thus about $\mathbf{h}_2$. But here we do not know exactly where a_{n+M} appears in the z-stream or if it appears at all. Therefore, we need to consider an interval near $z_{t+M/4}$, and collect the information about $\mathbf{h}_2$ as

$$P(\mathbf{h}_2 | \ldots, z_{t+M/4}, z_{t+M/4+1}, \ldots). \tag{7.13}$$

By combining these two probabilities we can build a classical distinguisher by calculating the probability

$$\gamma_t = P(\mathbf{h}_1 + \mathbf{h}_2 = \mathbf{0} | z_t, z_{t+1}, \ldots; \ldots, z_{t+M/4}, z_{t+M/4+1}, \ldots). \tag{7.14}$$

For $z_t,\ t \geq 0$, being a truly random sequence, the expected value of γ_t is 2^{-B_h}, whereas for a keystream output sequence from the SSG, the expected value will be larger. Again we use a log-likelihood test, and for suitable N_0 calculate

$$\Lambda = \sum_{t=0}^{N_0-1} \log_2 \left(\frac{\gamma_t}{2^{-B_h}} \right), \begin{cases} \text{if } \Lambda \geq \Lambda_0, \text{ decide for self-shrinking,} \\ \text{if } \Lambda < \Lambda_0, \text{ decide for random sequence,} \end{cases} \tag{7.15}$$

where Λ_0 is a threshold value. Given an observed keystream output of N bits, the value N_0 depends on M and on how many z-stream bits we choose to condition on in (7.13).

Restrictions on the class

Not all values of M and degrees of $g_1(x)$ and $g_2(x)$ are suitable for the proposed distinguishing attack. As mentioned previously, k is the maximum value for the degrees of $g_1(x)$ and $g_2(x)$. Assume for simplicity that $g_1(x)$ has degree $k_1 = k$ and $g_2(x)$ has degree $k_2 \leq k$. When calculating the first position in the vector $\mathbf{h}_1$, we use the a-stream block

$$(a_n, a_{n+1}, \ldots, a_{n+k_1}), \tag{7.16}$$

where a_n is a selection bit and $a_{n+1} \to z_t$ for some observed z_t (compare with (7.10)). For the next position in $\mathbf{h}_1$, we use the a-stream block

$$(a_{n+1}, a_{n+2}, \ldots, a_{n+k_1+1}),$$

et cetera.

Now, assume that k_1 is even. Since a_n is a selection bit, so is a_{n+k_1}. The distribution of each active bit in (7.16) is dependent on the bits previously assumed, conditioned on the output. However, if a_{n+k_1} is a selection bit, the distribution of that bit will not be conditioned on anything, since we do not estimate it together with its active bit, and the distribution of a_{n+k_1} will be uniform.

Since a_{n+k_1} is used in the summation for the first position, its uniform distribution will effectively mask the biased distribution of the other bits used in the summation. As a result the distribution of the first bit position of $\mathbf{h}_1$ will be uniform.

For the second position in $\mathbf{h}_1$, the situation is reversed. Now, both a_{n+1} and a_{n+k_1+1} are active bits and the summation of the bits in that a-stream block will yield a biased distribution for the sum. Continuing this line of reasoning, we see that every second bit position in $\mathbf{h}_1$ will have a uniform distribution.

Next, we note that if M is even, the first bit in the a-stream block used for calculating $\mathbf{h}_2$ will also be a selection bit. This means that if k_2 is odd, the last bit a_{n+M+k_2} will be an active bit, and the resulting distribution for the first position of $\mathbf{h}_2$ will be biased. The second position will be uniform, et cetera. However, when adding the vectors as done in (7.14), the biased first position of $\mathbf{h}_2$ will be cancelled out by the uniformly distributed first position of $\mathbf{h}_1$. For the second position, the biased bit from $\mathbf{h}_1$ will be cancelled out by the uniformly distributed second position of $\mathbf{h}_2$, and so on. To prevent this, we state the following rule as to which values of M and which degrees of $g_1(x)$ and $g_2(x)$ are allowable:

> If M is even/odd, then the difference in degree of $g_1(x)$ and $g_2(x)$ must be even/odd respectively.

Hence, the uniformly distributed bits from $\mathbf{h}_1$ and $\mathbf{h}_2$ will overlap and the resulting vector will be biased.

7.4 Implementation aspects and simulation results

Firstly, we consider the calculations of the distributions in (7.12) and (7.13).

For the distribution of $\mathbf{h}_1$, we assume a block in the z-stream of length B_z, starting at z_t for some $t \geq 0$. Recall the additional assumptions that $a_n = 1$ and $a_{n+1} \rightarrow z_t$. The first aim is to calculate the distribution

$$P\left((a_n, a_{n+1}, \ldots, a_{n+(B_h-1)})|z_t, z_{t+1}, \ldots, z_{t+(B_z-1)}\right). \tag{7.17}$$

The distribution $P(\mathbf{h}_1|z_t, \ldots, z_{t+(B_z-1)})$ is then straightforward to derive and can be pre-computed and stored in a look-up table. For the calculation of the probability

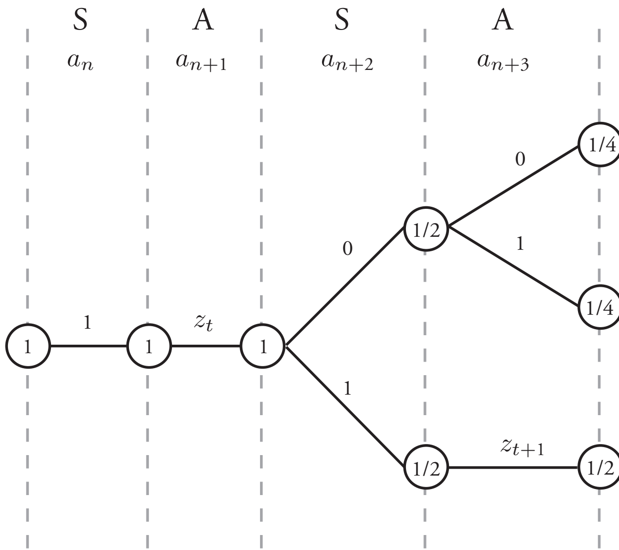


Figure 7.4: The beginning of the trellis used to calculate the conditioned probability (7.17). "S" denotes a selection bit and "A" denotes an active bit. The edges are labelled with the possible bit values of the a-bit and the nodes are labelled with the probability of that pattern.

in (7.17) we can use a trellis, which is a branching tree with probabilities in the nodes, see Figure 7.4. Each possible pattern for the a-block can be read off the tree from left to right. The end nodes give the probability of that particular pattern.

The distribution of $\mathbf{h}_2$ is somewhat more complicated as we do not know exactly where those a-bits are printed in the z-stream. The aim is again to derive the proba-

bility

$$P\left((a_{n+M}, a_{n+M+1}, \ldots, a_{n+M+(B_h-1)})|(\ldots, z_{t+M/4}, z_{t+M/4+1}, \ldots)\right). \quad (7.18)$$

Without going too deep into technicalities, let z_τ be the position in the z-stream where the first active bit from the considered a-block is visible. If M is odd, a_{n+M} is an active bit and if M is even, a_{n+M} is a selection bit. Similarly to Figure 7.4, we can build a trellis conditioned on the assumption that z_τ is the first visible bit, and then weight the derived node probability with the probability that z_τ is the first bit. The probability that z_τ is the first visible bit from the a-block is given by the binomial distribution. We have in our implementation chosen an interval for z_τ proportional to $\sqrt{M}$. From the distribution in (7.18), the distribution of $\mathbf{h}_2$ can easily be derived.

With pre-computed tables of the conditioned probabilities in (7.12) and (7.13), the time complexity of the attack is rather modest. We need to scan the observed keystream once, making it linear in the parameter N_0. For each position z_t, we scan the keystream near $z_{t+M/4}$, over an interval proportional to $\sqrt{M}$, in order to derive the weighted distribution of the second a-block. Finally, we combine the two distributions $\mathbf{h}_1$ and $\mathbf{h}_2$, which takes a time proportional to B_h.

Simulation results

Some simulation results for different configurations are shown in Figures 7.5, 7.6, 7.7 and 7.8. For these figures, the x-axis shows the $\log_2$ of N_0, the number of trials for the log-likelihood test (7.15) and the y-axis shows the log-likelihood ratio. Typically, a log-likelihood ratio larger than 10 would be sufficient for a correct decision with high probability.

The number of trials, N_0, we can perform in the log-likelihood test (7.15), with an observed keystream sequence of length N, is about $N_0 = N - M - \sqrt{M}$. Thus, N needs to be slightly larger than indicated in the simulation figures.

The simulations were done by first choosing suitable $g_1(x)$, $g_2(x)$ and M, then factorising $g(x) = g_1(x) + x^M g_2(x)$ to find a primitive characteristic polynomial for the SSG. Of course, for an attack, we would have to start with a characteristic polynomial and then find a multiple of the desired form. A simple procedure for finding such a multiple could, for example, be as follows.

(*i*) Let $f(x)$ denote the given feedback polynomial. Pick a suitable polynomial $g_1(x)$ and starting value for M.

(*ii*) Calculate the remainder $r(x)$ in

$$g_1(x)x^M = f(x)h(x) + r(x), \quad (7.19)$$

using the Euclidean algorithm.

(*iii*) If $deg(r(x)) \leq k$ we have found a candidate $g_2(x) = r(x)$. Otherwise increase M and calculate (7.19) again, et cetera.

The degree and weight of the polynomials are shown in Table 7.1, where the entries in "Original LFSR" are for the actual characteristic polynomial of the SSG used in the simulations.

Plotted in	Original LFSR		$g_1(x)$		$g_2(x)$		
Figure	Degree	Weight	Degree	Weight	Degree	Weight	M
7.5	45	21	4	3	6	3	1000
7.6	53	29	6	5	6	5	1004
7.7	50	29	6	5	6	5	5016
7.8	42	15	8	7	8	7	1016

Table 7.1: Configuration data for the simulations.

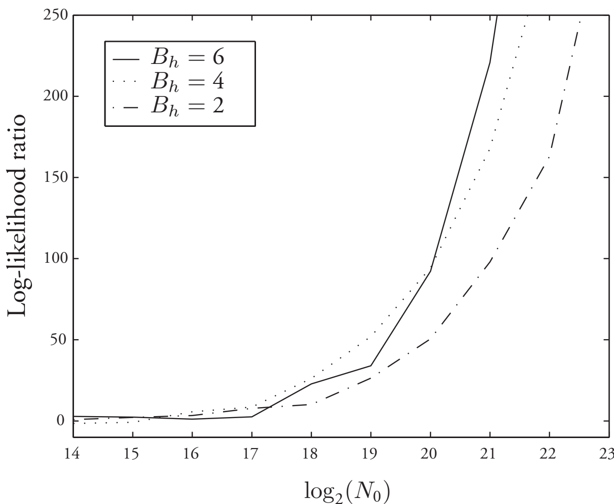


Figure 7.5: Log-likelihood ratio as a function of the logarithm of the number of trials. The specific configuration used in this simulation of the attack is given in Table 7.1.

In Figure 7.5 the polynomial $g_1(x)$ is of degree 4, weight 3 and $g_2(x)$ is of degree 6, weight 3. The distance between them is $M = 1000$. Three plots are drawn for different block length B_h. Larger block length means the log-likelihood ratio increases at a greater rate, but the attack needs more memory and is somewhat slower.

Using $B_h = 6$, the attack is successful with an observed keystream sequence of length approximately $N \approx 2^{18}$.

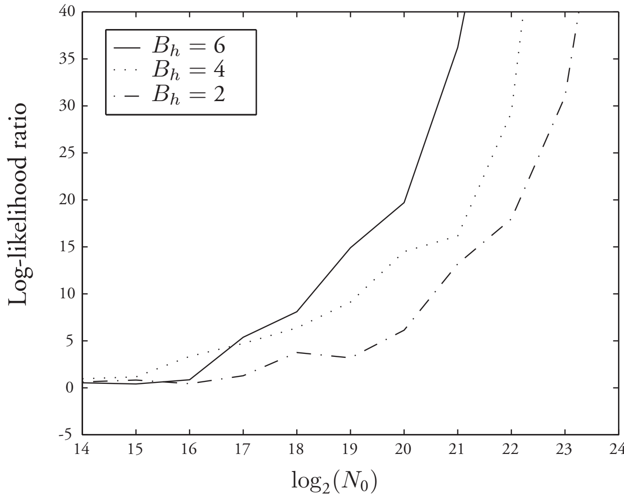


Figure 7.6: Log-likelihood ratio as a function of the logarithm of the number of trials. The specific configuration used in this simulation of the attack is given in Table 7.1.

In Figure 7.6, both the degree and the weight of $g_1(x)$ and $g_2(x)$ are greater. Note that the scale on the y-axis has changed. With this configuration, we need $N \approx 2^{19}$ observed keystream bits.

In Figure 7.7, the distance between $g_1(x)$ and $g_2(x)$ is increased to $M = 5016$. This increases the interval in which a_{n+M} has a high probability to be found and naturally the attack becomes more difficult. For a log-likelihood ratio of 10, we see that $N \approx 2^{21}$ observed keystream bits are required.

The last simulation, shown in Figure 7.8, uses once again a smaller value of M, but the degree is 8 and the weight is 7 for both $g_1(x)$ and $g_2(x)$. Increased weight results in less bias for $\mathbf{h}_1$ and $\mathbf{h}_2$ and thus increases the attack complexity. Here we need $N \approx 2^{24}$ to be able to reliably distinguish the output from the SSG from a random sequence.

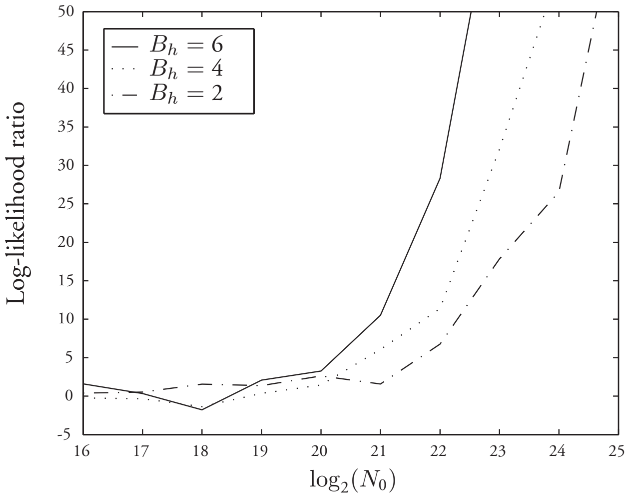


Figure 7.7: Log-likelihood ratio as a function of the logarithm of the number of trials. The specific configuration used in this simulation of the attack is given in Table 7.1.

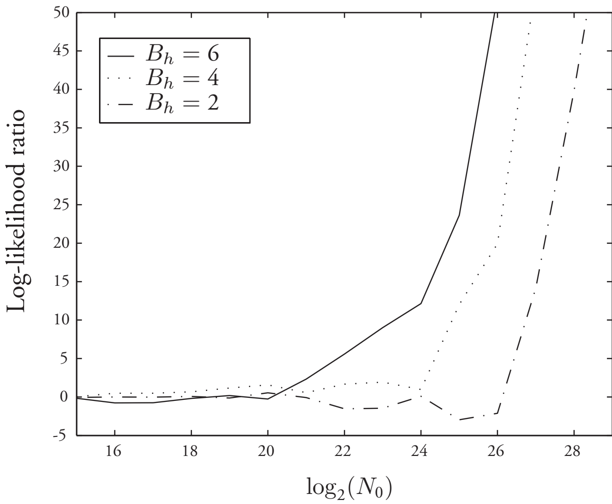


Figure 7.8: Log-likelihood ratio as a function of the logarithm of the number of trials. The specific configuration used in this simulation of the attack is given in Table 7.1.

7.5 Summary

In this section, an efficient distinguishing attack on the self-shrinking generator with a known feedback polynomial was presented. The attack is applicable to feedback polynomials (or multiples thereof) of weight 3, with two taps sitting together. This class is called the first class of weak polynomials. The computational complexity of the attack is linear in the degree of the feedback polynomial, making it a huge improvement as compared to existing attacks.

We also presented an initial state recovery algorithm for the SSG with a known feedback polynomial from this first class. The distinguishing and the initial state recovery attacks can also be used in conjunction with each other in the case of an unknown feedback polynomial. Thereby, the attack can first identify the use of a key which results in a weak polynomial from this class and subsequently recover the key. This approach has lower computational complexity and better performance than previously known attacks, in the case of unknown feedback polynomial.

Next, a more general class of weak polynomials was considered, and a powerful distinguishing attack for that class was presented. The computational complexity of this attack is still an open problem, but simulations show that the approach is efficient.

A natural extension of this attack would be to try to find an initial state recovery algorithm for the general class of polynomials. Another interesting task would be to investigate the performance of this attack if we not only use polynomials $g(x)$ with two clusters of taps at a distance M, but also three or more clusters of taps. The polynomial $g(x)$ could then, for example, be of the form $g(x) = g_1(x) + x^{M_1}g_2(x) + x^{M_2}g_3(x)$.

Finally, we note that the ideas behind the described attack, for the general class of polynomials, can also be used to derive a similar kind of attack on the shrinking generator.

8

SNOW—a new family of stream ciphers

We will now leave the subject of cryptanalysis and turn to the perhaps more complicated area of cryptography, the construction of cryptographic primitives. The construction of good ciphers can be considered more difficult than attacking them since there are many properties that need to be balanced against each other. Some of the main considerations are typically security level, encryption speed, speed of initialisation, hardware print (in terms of gates), software print (in terms of memory) and other features, for example, the use of an IV in a frame based communication.

In Chapter 2 we discussed the basics of stream ciphers constructed from LFSRs and in this chapter, we will refine some ideas and present a new family of stream ciphers called SNOW. Currently, there exists two versions of this cipher, SNOW 1.0 and SNOW 2.0. Both versions of the cipher are keystream generators based on an LFSR defined over the field $\mathbb{F}_{2^{32}}$, where the nonlinearity is provided by a Finite State Machine (FSM).

From a cryptographical point of view, stream ciphers have not had as strong attention within the scientific research community as block ciphers. Nevertheless, many public communication products and military ditto have employed stream ciphers as the means for privacy. In this thesis we have, for example, analysed two ciphers, $A5/1$ and E_0, used in well-known civilian systems. Note however, that both these ciphers were developed without the scrutiny of the research community. This is also the case for the military applications, although this is much more understandable.

The interest in stream cipher development was recently boosted by the European NESSIE project [92], that began in 1999. The goal for the NESSIE project was "*to*

put forward a portfolio of strong cryptographic primitives" for governmental and industrial use. The approach taken was to publish a "Call for Cryptographic Primitives", inviting the scientific research community to participate in both the construction and the evaluation of the new primitives. Primitives in several areas were considered, including block ciphers, stream ciphers, public-key primitives, MACs, et cetera. There were five new designs[1] submitted to NESSIE in the stream cipher category, including the first version of SNOW (SNOW 1.0) [29]. The NESSIE project has now ended and none of the submitted stream ciphers were chosen to be the NESSIE recommendation. The SOBER-t16 and SOBER-t32 ciphers were removed primarily due to the attacks presented in Chapter 3. SNOW 1.0 was removed due to a guess-and-determine attack and a distinguishing attack to be discussed in Section 8.2. These attacks revealed some weaknesses in the design and a new improved version of the cipher, SNOW 2.0, was developed. SNOW 2.0 was first presented in [33].

Remark. As we will keep the original notations [29, 33] for the description of the two ciphers, parts of this chapter may not conform to the rest of this thesis. Furthermore, some designations will be used in both the description of SNOW 1.0 and SNOW 2.0 and thus the meaning of those notations will change within the chapter as well. The meaning will be made explicit whenever there is a risk of confusion.

This chapter is organised as follows. In Section 8.1, the first version SNOW 1.0 is presented. The known attacks and weaknesses of this first design are then discussed in Section 8.2. Thereafter, in Section 8.3, SNOW 2.0 is presented in detail, and some design differences between the two versions are highlighted in Section 8.4. In Section 8.5 implementation aspects of SNOW 2.0 are considered and in Section 8.6 an attack on SNOW 2.0 is discussed. Finally, a summary of the chapter is given in Section 8.7.

8.1 A description of SNOW 1.0

SNOW 1.0 is a word oriented stream cipher with a word size of 32 bits. The cipher is described with two possible key sizes, 128 and 256 bits. As usual, the encryption starts with a key initialisation, giving the components of the cipher their initial values, but firstly, we will concentrate on the operational description.

The generator is depicted in Figure 8.1. It consists of a length 16 LFSR defined over $\mathbb{F}_{2^{32}}$, feeding a finite state machine. The FSM consists of two 32 bit registers, called R1 and R2, as well as some operations to calculate the output and the next state (the next value of R1 and R2).

The keystream is generated by bitwise adding (xoring) the output of the FSM with the last entry of the LFSR. After that the whole cipher is clocked once, and the next

[1] Counting SOBER-t16 and SOBER-t32 as one family

Figure 8.1: A schematic diagram of SNOW 1.0

32 bits of the keystream are calculated by again xoring the output of the finite state machine and the last entry of the LFSR. We clock again and continue in this fashion.

Returning to Figure 8.1, the LFSR has a primitive feedback polynomial over $\mathbb{F}_{2^{32}}$ given by

$$p(x) = x^{16} + x^{13} + x^7 + \alpha^{-1},$$

where $\mathbb{F}_{2^{32}}$ is generated by the irreducible polynomial

$$\pi(x) = x^{32} + x^{29} + x^{20} + x^{15} + x^{10} + x + 1,$$

over $\mathbb{F}_2$, and $\pi(\alpha) = 0$.

Furthermore, let $s(1), s(2), \dots s(16) \in \mathbb{F}_{2^{32}}$ be the state of the LFSR. Here $s(1)$ is associated with the leftmost memory element in Figure 8.1, and $s(16)$ with the rightmost. We consider a representation of the elements in $\mathbb{F}_{2^{32}}$, using the base $\{\alpha^{31}, \dots, \alpha^2, \alpha, 1\}$, i.e. if $y \in \mathbb{F}_{2^{32}}$ then y is represented by $(y_{31}, y_{30}, \dots, y_1, y_0)$, where

$$y = y_{31}\alpha^{31} + y_{30}\alpha^{30} + \dots + y_1\alpha + y_0.$$

We consider $y_{31}, y_{30}, \dots$ to be the most significant bits (MSB) and $\dots, y_1, y_0$ to be the least significant bits (LSB).

The output of the FSM, called FSM_{out}, is calculated as

$$FSM_{\text{out}} = (s(1) \boxplus R1) \oplus R2.$$

The output of the FSM is xored with $s(16)$ to form the keystream, i.e.

$$\text{running key} = FSM_{\text{out}} \oplus s(16).$$

The keystream is finally xored with the plaintext, producing the ciphertext.

Inside the FSM, the new values of R1 and R2 are given according to

$$\begin{aligned} \text{tempR1} &= ((FSM_{\text{out}} \boxplus R2) \lll) \oplus R1, \\ R2 &= S(R1), \\ R1 &= \text{tempR1}. \end{aligned}$$

We recall the notation $x \boxplus y$, which here denotes the integer addition of x and y mod 2^{32} and the addition $x \oplus y$ denotes the field addition (xor). The notation $x \lll$ is a cyclic shift of x 7 steps to the left.

Finally, the S-Box, denoted $S(x)$, consists of four identical 8-to-8 bit S-Boxes and a permutation of the resulting bits. It works as follows. The input x is split into 4 bytes, from most significant to least significant byte. Each of the bytes enters a nonlinear mapping from 8 bits to 8 bits.

Let the input to the nonlinear mapping be $w = (w_7, w_6, \ldots, w_0)$ and let the output be $r = (r_7, r_6, \ldots, r_0)$. Both vectors are considered as representing elements in $\mathbb{F}_{2^8}$ using the polynomial base $\{\beta^7, \ldots, \beta, 1\}$ generated by the irreducible polynomial $\pi(x) = x^8 + x^5 + x^3 + x + 1$ and $\pi(\beta) = 0$. The nonlinear mapping is defined to be

$$r = w^7 + \beta^2 + \beta + 1,$$

where the arithmetic is in $\mathbb{F}_{2^8}$.

After the mapping above has been applied to each byte, the bits in the resulting word are permuted. The permutation is described by

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
3	10	20	24	0	14	17	29	7	13	18	25	5	12	23	27

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	8	21	26	4	9	19	31	2	11	16	28	6	15	22	30

which should be interpreted as the 31st bit position is mapped to the 3rd, the 30th

bit is mapped to the 10th, et cetera. Using a vector notation, this can be written as

$$y = (y_{31}, y_{30}, y_{29}, \dots, y_1, y_0) \rightarrow (y_8, y_0, y_{24}, \dots, y_{15}, y_{27}).$$

The S-box is shown in Figure 8.2, where $y = S(x)$ and $\gamma = \beta^2 + \beta + 1$.

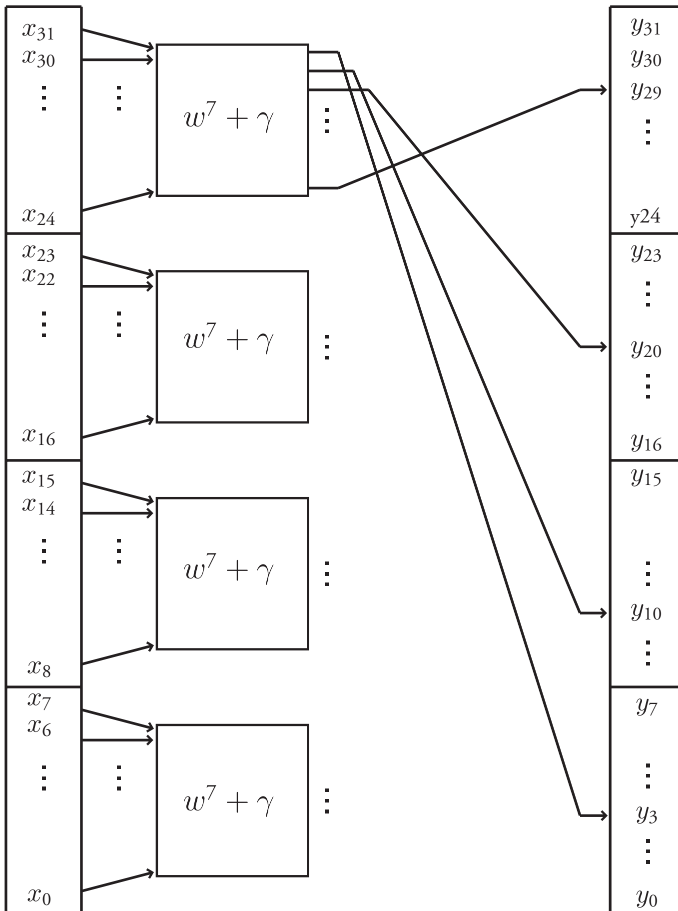


Figure 8.2: The S-Box in SNOW 1.0, $y = S(x)$.

Modes of operation

Two different modes of operation are specified for SNOW 1.0. These are referred to as standard mode and IV mode, respectively.

Standard mode: In standard mode SNOW 1.0 implements a fast cryptographic pseudo-random number generator. This means that for each seed, which in this case is a secret key denoted by k, SNOW 1.0 outputs a pseudo-random number sequence.

IV mode: In IV mode the generator is initialized using two variables, the secret key k and a known initialisation value (IV). This means that for a given secret key k, the generator now produces a set of pseudo-random number sequences, one for each IV value. Since the produced sequences are supposed to be indistinguishable from truly random sequences in all aspects, the IV mode of SNOW 1.0 can be said to implement a *length-increasing pseudo-random function* (from the set of IV values to the set of possible sequences). The length of the output sequences is usually larger than the IV length.

In SNOW 1.0, the IV value is a 64 bit value, represented by the two 32 bit words (IV_2, IV_1). The IV value thus ranges from 0 to $2^{64} - 1$, where IV_2 is the most significant word and IV_1 is the least significant word.

The use of an IV value is optional and applications requiring an IV value typically reinitialise the cipher frequently with a fixed key but the IV value is changed. This could be the case if two parties agreed on a common secret key but wish to communicate multiple messages, for example, in a frame based setting. Frequent reinitialisation could also be desirable from a resynchronisation perspective in, for example, a radio based environment.

Since the IV mode will use frequent reinitialisation, the performance of the key initialisation will be an important performance parameter. Hence, the key initialisation in the IV mode uses less SNOW 1.0 clockings than in the standard mode (32 versus 64).

Key initialisation

Let the secret key k be denoted by $k = (k(1), k(2), k(3), k(4))$ in the 128 bit case and $k = (k(1), k(2), k(3), k(4), k(5), k(6), k(7), k(8))$ in the 256 bit case.

The key initialisation is done as follows. The LFSR is first initialised with the key. In the 128 bit case, the LFSR initialisation is

$$\begin{aligned} &s(1) = k(1) \oplus IV_1, \quad s(2) = k(2), \quad s(3) = k(3), \quad s(4) = k(4) \oplus IV_2, \\ &s(5) = k(1) \oplus \mathbf{1}, \quad s(6) = k(2) \oplus \mathbf{1}, \quad s(7) = k(3) \oplus \mathbf{1}, \quad s(8) = k(4) \oplus \mathbf{1}, \end{aligned}$$

and for the second half of the register,

$$\begin{aligned} &s(9) = k(1), \quad s(10) = k(2), \quad s(11) = k(3), \quad s(12) = k(4), \\ &s(13) = k(1) \oplus \mathbf{1}, \quad s(14) = k(2) \oplus \mathbf{1}, \quad s(15) = k(3) \oplus \mathbf{1}, \quad s(16) = k(4) \oplus \mathbf{1}, \end{aligned}$$

where **1** denotes the all one vector (32 bits).

In the 256 bit case, the LFSR initialisation is correspondingly,

$$\begin{array}{llll} s(1) = k(1) \oplus IV_1, & s(2) = k(2), & s(3) = k(3), & s(4) = k(4) \oplus IV_2, \\ s(5) = k(5), & s(6) = k(6), & s(7) = k(7), & s(8) = k(8), \\ s(9) = k(1) \oplus \mathbf{1}, & \ldots & & s(16) = k(8) \oplus \mathbf{1}. \end{array}$$

In standard mode, we assume $IV_1 = IV_2 = 0$. After the LFSR has been initialised, R1 and R2 are both set to zero.

Then the cipher is clocked exactly v times without producing any running key. Instead, the output of the finite state machine is fed back into the feedback loop of the LFSR, as shown in Figure 8.3. In standard mode $v = 64$, and in IV mode $v = 32$. In

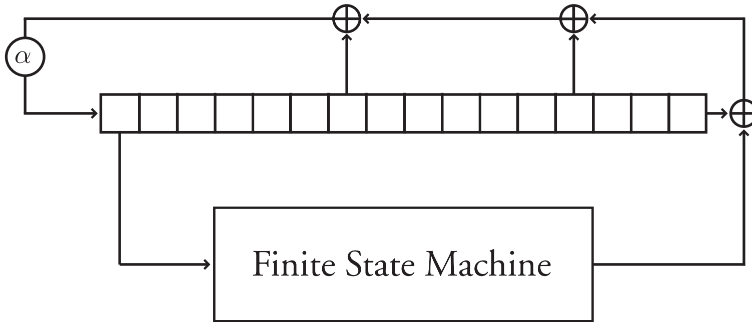


Figure 8.3: The key initialisation of SNOW 1.0.

one clock cycle, the next value of $s(1)$, here called newS(1), is given by

$$\text{newS}(1) = \alpha(s(7) \oplus s(13) \oplus s(16) \oplus FSM_{\text{out}}).$$

After v clockings, the LFSR and the two registers R1, R2 have received their values from the initialisation phase. The first 32 bits of the running key are now available as $FSM_{\text{out}} \oplus s(16)$.

The maximum allowed length of the running key (output sequence) is set to 2^{50} words after which SNOW 1.0 must be rekeyed.

This concludes the description of SNOW 1.0. The reader is referred to [29] for a discussion on implementation aspects and design choices of SNOW 1.0. Test vectors for SNOW 1.0 can be found in the NESSIE project documentation [92].

8.2 Attacks on SNOW 1.0

As previously mentioned, two attacks have been published on SNOW 1.0. In this section we will outline the attacks and identify the weaknesses enabling them.

The first is a guess-and-determine attack by Hawkes and Rose [58]. Their attack has a data complexity of 2^{95} words and a time complexity of 2^{224} operations. If we use SNOW 1.0 with a key size of 256 bits, this attack is faster than the generic exhaustive key search attack. With the exception of some clever initial choices made by Hawkes and Rose, basically two properties in SNOW 1.0 are used to reduce the complexity of the attack below that of the exhaustive key search. Firstly, the fact that the FSM has only *one* input $s(1)$. This enables an attacker to invert the operations in the FSM and derive more unknowns from only a few guesses. The second property is an unfortunate choice of feedback polynomial in SNOW 1.0. Recall that the linear recurrence equation is given by

$$s_{t+16} = \alpha(s_{t+9} + s_{t+3} + s_t). \tag{8.1}$$

There is a distance of 3 words between s_t and s_{t+3} and a distance of $6 = 2 \cdot 3$ between s_{t+3} and s_{t+9}. Thus, by squaring (8.1)

$$s_{t+32} = \alpha^2(s_{t+18} + s_{t+6} + s_t) \tag{8.2}$$

we see that $(s_{t+i} \oplus s_{t+i+6})$ can be considered as a single input to either equation. Hence, the attacker does not need to determine both s_{t+i} and s_{t+i+6} explicitly, but only the xor sum to use in (8.1) and (8.2).

A second weakness in the choice of the feedback polynomial emerges when considering *bitwise* linear approximations. Using the same technique as in Chapter 3, we can compute the 2^{32}th power of the feedback polynomial $p(x) = x^{16} + x^{13} + x^7 + \alpha^{-1} \in \mathbb{F}_{2^{32}}[x]$, resulting in the polynomial

$$p^{2^{32}}(x) = x^{16 \cdot 2^{32}} + x^{13 \cdot 2^{32}} + x^{7 \cdot 2^{32}} + \alpha^{-1 \cdot 2^{32}} \in \mathbb{F}_{2^{32}}[x]. \tag{8.3}$$

Since $\alpha \in \mathbb{F}_{2^{32}}$ we have $\alpha^{-1 \cdot 2^{32}} = \alpha^{-1}$, and a summation of $p(x) + p^{2^{32}}(x)$ yields

$$x^{16 \cdot 2^{32}} + x^{13 \cdot 2^{32}} + x^{7 \cdot 2^{32}} + x^{16} + x^{13} + x^7. \tag{8.4}$$

Dividing (8.4) with x^7 gives us a linear recurrence equation satisfying

$$s_{t+16 \cdot 2^{32}-7} + s_{t+13 \cdot 2^{32}-7} + s_{t+7 \cdot 2^{32}-7} + s_{t+9} + s_{t+6} + s_t = 0 \tag{8.5}$$

In (8.5) we have derived a linear recurrence equation that holds for each single bit position. Hence, any bitwise correlation found in the FSM can be turned into a distinguishing attack.

In 2002, Coppersmith, Halevi and Jutla [17] found such a correlation. Denote the input word to the FSM at time t by f_t and the output word from the FSM at time t by F_t. Let $x[i]$ denote bit number i in word x, where $i = 0, \ldots, 31$ and $i = 0$ is the LSB. The correlation found in [17] can now be stated as

$$P(\sigma_t = 0) \approx \frac{1}{2} + 2^{-9.3}, \tag{8.6}$$

where

$$\sigma_t = f_t[15] \oplus f_t[16] \oplus f_{t+1}[22] \oplus f_{t+1}[23] \oplus F_t[15] \oplus F_{t+1}[23]. \tag{8.7}$$

By combining (8.6) and (8.7) with (8.5), a distinguishing attack can be mounted on SNOW 1.0, using similar techniques as in Chapter 3.

The resulting distinguishing attack in [17] needs approximately 2^{95} words of output to distinguish a keystream generated by SNOW 1.0 from a truly random sequence. The computational complexity of the attack is approximately 2^{100}.

By computer search, we have also found other smaller correlations, often involving similar bit positions as the one found in [17]. The strong correlations seem to be caused by an interaction between the permutation in the S-Box and the cyclic shift by 7 in the FSM, but the exact reason for the large correlations is unclear.

8.3 A description of SNOW 2.0

As we now turn to a description of SNOW 2.0, it must be emphasised that the notations from the previous sections are no longer valid and will be redefined in the following discussion. The new version is schematically a small modification of the original construction, see Figure 8.4. The word size is unchanged (32 bits) and the LFSR length is again 16, however, the feedback polynomial has been changed. The FSM has two input words, taken from the LFSR, and the running key is formed as the xor between the FSM output and the last element of the LFSR, as in SNOW 1.0. The operation of the cipher is as follows. Firstly, a key initialisation is performed. This operation provides the LFSR with a starting state as well as giving the internal FSM registers $R1$ and $R2$ their initial values. Next the cipher is clocked once and the first keystream symbol is read out[2]. Then the cipher is clocked again and the second keystream symbol is read, et cetera.

Let us give a detailed description of the cipher, starting with the LFSR. The main reason for choosing the specific feedback polynomial in SNOW 1.0, was to have a fast realisation in software. By choosing a multiplication with the same primitive element as the base is constructed from, we can realise the multiplication with just one left shift and a possible xor with a known bit pattern. However, this choice opens up possible

[2] Observe the change from SNOW 1.0, where the first symbol was read out *before* the cipher was clocked.

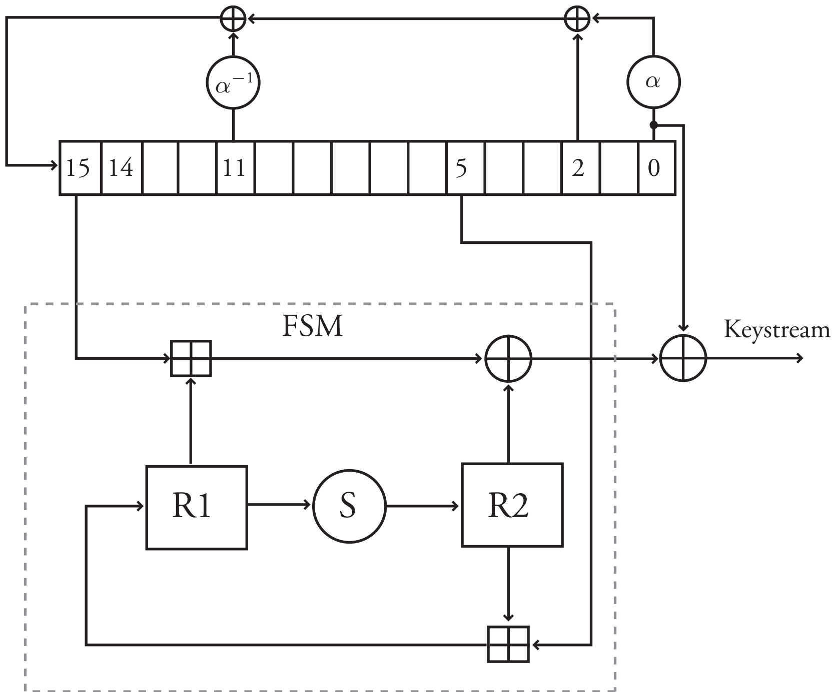


Figure 8.4: A schematic diagram of SNOW 2.0. To simplify the layout, the LFSR elements are labelled only with the relative numbers. For example, the element labelled 14 should thus be interpreted as s_{t+14}.

weaknesses, as discussed in Section 8.2. In SNOW 2.0, we have two different elements involved in the feedback loop, α and α^{-1}, where α is now a root of a primitive polynomial of degree 4 over $\mathbb{F}_{2^8}$. To be more precise, the feedback polynomial of SNOW 2.0 is given by

$$\pi(x) = \alpha x^{16} + x^{14} + \alpha^{-1} x^5 + 1 \in \mathbb{F}_{2^{32}}[x], \tag{8.8}$$

where α is a root of

$$x^4 + \beta^{23} x^3 + \beta^{245} x^2 + \beta^{48} x + \beta^{239} \in \mathbb{F}_{2^8}[x], \tag{8.9}$$

and β is a root of

$$x^8 + x^7 + x^5 + x^3 + 1 \in \mathbb{F}_2[x]. \tag{8.10}$$

Let the state of the LFSR at time $t \geq 0$ be denoted $(s_{t+15}, s_{t+14}, \ldots, s_t)$, $s_{t+i} \in \mathbb{F}_{2^{32}}$, $i \geq 0$. The element s_t is the rightmost element (or first element to exit) as indicated in Figure 8.4, and the sequence produced by the LFSR is $(s_0, s_1, s_2, \ldots)$.

By time $t = 0$, we mean the time instance directly after the key initialisation. Then the cipher is clocked once before producing the first keystream symbol, i.e. the first keystream symbol, denoted z_1, is produced at time $t = 1$. The produced keystream sequence is denoted $(z_1, z_2, z_3, \ldots)$.

The FSM has two registers, denoted $R1$ and $R2$, each holding 32 bits. The value of the registers at time $t \geq 0$ is denoted $R1_t$ and $R2_t$ respectively. The input to the FSM is (s_{t+15}, s_{t+5}) and the output of the FSM, denoted F_t, is calculated as

$$F_t = (s_{t+15} \boxplus R1_t) \oplus R2_t, \quad t \geq 0, \tag{8.11}$$

and the keystream is given by

$$z_t = F_t \oplus s_t, \quad t \geq 1. \tag{8.12}$$

Here we use the notation $\boxplus$ for integer addition modulo 2^{32} and $\oplus$ for bitwise addition (xor). The registers $R1$ and $R2$ are updated with new values according to

$$R1_{t+1} = s_{t+5} \boxplus R2_t \quad \text{and} \tag{8.13}$$
$$R2_{t+1} = S(R1_t) \quad t \geq 0. \tag{8.14}$$

The S-Box

The S-Box, denoted by $S(w)$, is a permutation on $\mathbb{Z}_{2^{32}}$ based on the round function of Rijndael [25]. Let $w = (w_3, w_2, w_1, w_0)$ be the input to the S-Box, where $w_i, i = 0 \ldots 3$ are the four bytes of w. Assume w_3 to be the most significant byte. Let

$$w = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{pmatrix}, \tag{8.15}$$

be a vector representation of the input to the S-Box. Firstly, we apply the Rijndael S-Box, denoted S_R, to each byte, giving us the vector

$$\begin{pmatrix} S_R[w_0] \\ S_R[w_1] \\ S_R[w_2] \\ S_R[w_3] \end{pmatrix}. \tag{8.16}$$

In the *MixColumn transformation* of Rijndael's round function, each 4 byte word is considered a polynomial in y over $\mathbb{F}_{2^8}$, defined by the irreducible polynomial $x^8 + x^4 + x^3 + x + 1 \in \mathbb{F}_2[x]$. Each word can be represented by a polynomial of at most degree 3. Next we consider the vector in (8.16) as representing a polynomial over $\mathbb{F}_{2^8}$ and multiply with a fixed polynomial $c(y) = (x + 1)y^3 + y^2 + y + x \in \mathbb{F}_{2^8}[y]$

modulo $y^4 + 1 \in \mathbb{F}_{2^8}[y]$. This polynomial multiplication can (as done in Rijndael) be computed as a matrix multiplication,

$$\begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{pmatrix} = \begin{pmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{pmatrix} \begin{pmatrix} S_R[w_0] \\ S_R[w_1] \\ S_R[w_2] \\ S_R[w_3] \end{pmatrix}, \quad (8.17)$$

where (r_3, r_2, r_1, r_0) are the output bytes from the S-Box. These bytes are concatenated to form the word output from the S-Box, $r = S(w)$.

Key initialisation

SNOW 2.0 takes two parameters as input values; a secret key of either 128 or 256 bits and a publicly known 128 bit initialisation value, IV. The IV value is considered as a four word input $IV = (IV_3, IV_2, IV_1, IV_0)$, where IV_0 is the least significant word. The possible range for IV is thus $0 \ldots 2^{128} - 1$. The use of an IV value is optional and applications not requiring an IV value typically use $IV = (0, 0, 0, 0)$.

The key initialisation is done as follows. Denote the registers in the LFSR by $(s_{15}, s_{14}, \ldots, s_0)$ from left to right in Figure 8.4. Thus, s_{15} corresponds to the element holding s_{t+15} during normal operation of the cipher. Let the secret key be denoted by $K = (k_3, k_2, k_1, k_0)$ in the 128 bit case and by $K = (k_7, k_6, k_5, k_4, k_3, k_2, k_1, k_0)$ in the 256 bit case, where each k_i is a word and k_0 is the least significant word. Firstly, the shift register is initialised with K and IV according to

$$\begin{array}{llll} s_{15} = k_3 \oplus IV_0, & s_{14} = k_2, & s_{13} = k_1, & s_{12} = k_0 \oplus IV_1, \\ s_{11} = k_3 \oplus \mathbf{1}, & s_{10} = k_2 \oplus \mathbf{1} \oplus IV_2, & s_9 = k_1 \oplus \mathbf{1} \oplus IV_3, & s_8 = k_0 \oplus \mathbf{1}, \end{array}$$

and for the second half of the register,

$$\begin{array}{llll} s_7 = k_3, & s_6 = k_2, & s_5 = k_1, & s_4 = k_0, \\ s_3 = k_3 \oplus \mathbf{1}, & s_2 = k_2 \oplus \mathbf{1}, & s_1 = k_1 \oplus \mathbf{1}, & s_0 = k_0 \oplus \mathbf{1}, \end{array}$$

where $\mathbf{1}$ denotes the all one vector (32 bits).

In the 256 bit case, the LFSR initialisation is correspondingly

$$\begin{array}{llll} s_{15} = k_7 \oplus IV_0, & s_{14} = k_6, & s_{13} = k_5, & s_{12} = k_4 \oplus IV_1, \\ s_{11} = k_3, & s_{10} = k_2 \oplus IV_2, & s_9 = k_1 \oplus IV_3, & s_8 = k_0, \\ s_7 = k_7 \oplus \mathbf{1}, & s_6 = k_6 \oplus \mathbf{1}, & \ldots & s_0 = k_0 \oplus \mathbf{1}. \end{array}$$

After the LFSR has been initialised, R1 and R2 are both set to zero. Then, the cipher is clocked 32 times without producing any output symbols. Instead, the output

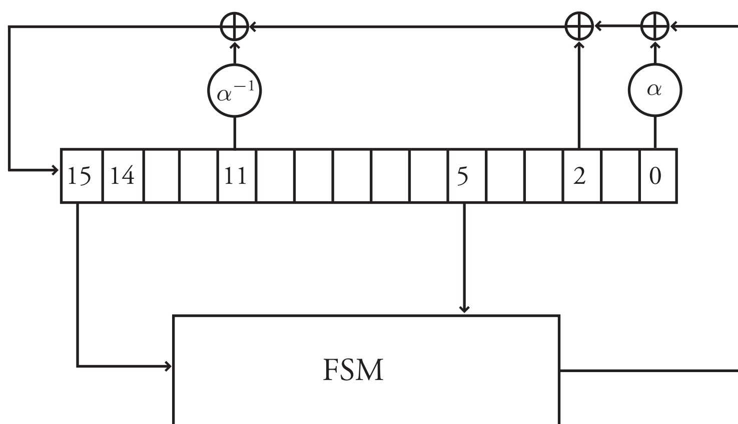


Figure 8.5: Cipher operation during key initialisation.

of the FSM is incorporated in the feedback loop, see Figure 8.5. Thus, during the 32 clocks in the key initialisation, the next element to be inserted into the LFSR is given by

$$s_{t+16} = \alpha^{-1} s_{t+11} \oplus s_{t+2} \oplus \alpha s_t \oplus F_t. \tag{8.18}$$

After the 32 clockings the cipher is shifted back to normal operation (Figure 8.4) and clocked once before the first keystream symbol is produced. The maximum number of keystream words allowed is set to 2^{50}, then the cipher must be rekeyed. This limit provides a bound for cryptanalysis and implies no practical limits to the operation of the cipher. A practical application needing more than 2^{50} words using the same key is rather unlikely.

8.4 Design differences—SNOW 1.0 vs. SNOW 2.0

In this section we highlight the differences between SNOW 2.0 and SNOW 1.0 and their expected security improvements. We start with the choice of feedback polynomial. In SNOW 1.0 the multiplication can be implemented by a single left shift of the word, followed by a possible xor with a known pattern of weight 6. This means that the resulting word is, in many positions, only a shift of the original word. In SNOW 2.0, we define $\mathbb{F}_{2^{32}}$ as an extension field over $\mathbb{F}_{2^8}$ and each of the two multiplications can be implemented as a *byte* shift together with an unconditional xor with one of 256 possible patterns. This results in a better spreading of the bits in the feedback loop, and improves the resistance against linearisation attacks, as discussed in Chapter 3. The use of *two* constants in the feedback loop also improves the resistance against bitwise linear approximation attacks, as discussed in Section 8.2. There is no

known method to manipulate the feedback polynomial such that the resulting linear recurrence holds for each bit position and has reasonably low weight. The unconditional xor also seems to improve speed, by removing the possible branch prediction error in a pipelined processor.

The FSM in SNOW 2.0 now takes *two* inputs. This makes a guess-and-determine type of attack more difficult. Given the output of the FSM, together with $R1$ and $R2$, it is no longer possible to deduce the next FSM state directly. The update of $R1$ does not depend on the output of the FSM, but on a word taken from the LFSR. This suggests that similar correlations to those found in [17] would be much weaker. A recent attack on SNOW 2.0, presented in Section 8.6, also verifies this conclusion.

The S-Box in SNOW 2.0 is byte oriented, similarly to in SNOW 1.0, but the final bit permutation in SNOW 1.0 does not diffuse as much as the new design. In SNOW 1.0, each input byte to the S-Box affects only 8 bits of the output word. The choice of the new S-Box, based on the round function of Rijndael, provides a much stronger diffusion. Each output bit now depends on each input bit.

8.5 Implementation aspects of SNOW 2.0

The design of SNOW 2.0 was done with a fast software implementation in mind. We have chosen a minimum number of different operations; xor, integer addition, byte shift of a word, and table lookups, all available on modern processors. Even though there are many possible tradeoffs in a software implementation, we will discuss some of the design aspects which have high impact in software.

We start with the LFSR. The field $\mathbb{F}_{2^{32}}$ is defined as an extension field over $\mathbb{F}_{2^8}$, with $\alpha \in \mathbb{F}_{2^{32}}$ being the root of the degree 4 polynomial

$$x^4 + \beta^{23}x^3 + \beta^{245}x^2 + \beta^{48}x + \beta^{239} \in \mathbb{F}_{2^8}[x]. \tag{8.19}$$

Hence, we have the degree reduction of α given by

$$\alpha^4 = \beta^{23}\alpha^3 + \beta^{245}\alpha^2 + \beta^{48}\alpha + \beta^{239}. \tag{8.20}$$

In the feedback loop, multiplication with α and α^{-1} can be implemented as a simple byte shift plus an additional xor with one of 256 possible patterns. This can be seen from the representation of a word as a polynomial in $\mathbb{F}_{2^8}[x]$ using $(\alpha^3, \alpha^2, \alpha, 1)$ as the base. Thus, any element w in $\mathbb{F}_{2^{32}}$ can be written as

$$w = c_3\alpha^3 + c_2\alpha^2 + c_1\alpha + c_0, \tag{8.21}$$

where (c_3, c_2, c_1, c_0) are the bytes of w, c_0 being the least significant byte. Multiplying w with α will yield a reduction according to (8.20) as follows

$$\begin{aligned} \alpha w &= c_3\alpha^4 + c_2\alpha^3 + c_1\alpha^2 + c_0\alpha \qquad (8.22) \\ &= (c_3\beta^{23} + c_2)\alpha^3 + (c_3\beta^{245} + c_1)\alpha^2 + (c_3\beta^{48} + c_0)\alpha + c_3\beta^{239}. \end{aligned}$$

Similar calculations can be done for the multiplication with α^{-1}. Thus, to get a fast implementation of the LFSR feedback, one can use pre-computed tables

$$MUL_\alpha[c] = (c\beta^{23}, c\beta^{245}, c\beta^{48}, c\beta^{239}), \quad (8.23)$$
$$MUL_{\alpha^{-1}}[c] = (c\beta^{16}, c\beta^{39}, c\beta^{6}, c\beta^{64}), \quad (8.24)$$

where c runs through all elements in $\mathbb{F}_{2^8}$. The pseudo-code for the multiplication would be

```
// Multiplication  w*alpha
// ("<<" is left shift, ">>" is right shift)
result=(w<<8) xor MUL_a[w>>24];
// Multiplication  w*alpha^-1
result=(w>>8) xor MUL_ainverse[w and 0xff];
```

The S-Box is implemented using the same techniques as done in Rijndael [25] and SCREAM [54]. Recall the expression for the S-Box, $r = S(w)$

$$\begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{pmatrix} = \begin{pmatrix} x & x+1 & 1 & 1 \\ 1 & x & x+1 & 1 \\ 1 & 1 & x & x+1 \\ x+1 & 1 & 1 & x \end{pmatrix} \begin{pmatrix} S_R[w_0] \\ S_R[w_1] \\ S_R[w_2] \\ S_R[w_3] \end{pmatrix}. \quad (8.25)$$

The matrix multiplication can be split up into a linear combinations of the columns

$$\begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{pmatrix} = S_R[w_0]\begin{pmatrix} x \\ 1 \\ 1 \\ x+1 \end{pmatrix} + S_R[w_1]\begin{pmatrix} x+1 \\ x \\ 1 \\ 1 \end{pmatrix} + S_R[w_2]\begin{pmatrix} 1 \\ x+1 \\ x \\ 1 \end{pmatrix} + S_R[w_3]\begin{pmatrix} 1 \\ 1 \\ x+1 \\ x \end{pmatrix}.$$

By using four tables of words, each of size 256, defined by

$$T_0[a] = \begin{pmatrix} xS_R[a] \\ S_R[a] \\ S_R[a] \\ (x+1)S_R[a] \end{pmatrix}, \quad T_1[a] = \begin{pmatrix} (x+1)S_R[a] \\ xS_R[a] \\ S_R[a] \\ S_R[a] \end{pmatrix},$$

$$T_2[a] = \begin{pmatrix} S_R[a] \\ (x+1)S_R[a] \\ xS_R[a] \\ S_R[a] \end{pmatrix}, \quad T_3[a] = \begin{pmatrix} S_R[a] \\ S_R[a] \\ (x+1)S_R[a] \\ xS_R[a] \end{pmatrix},$$

we can easily implement the S-Box by addressing the tables with the four bytes (w_3, w_2, w_1, w_0) of the input word w. In pseudo-code we can write

```
// Calculate r=S-Box(w)
r=T0[byte0(w)] xor T1[byte1(w)] xor T2[byte2(w)]
    xor T3[byte3(w)];
```

where `byte0(w)` means the least significant byte of w, et cetera.

We have two different C implementations, both using tables for feedback multiplication and S-Box operations. The first version (*version 1*) implements the LFSR with an array using the *sliding window* technique, see for example [57]. This version is considered an "easy to read" standard reference version. The second version (*version 2*) implements the cipher with "hard coded" variables for the LFSR. This version produces $16 \cdot 32 = 512$ bits of keystream in each procedure call, corresponding to 16 consecutive clockings. Table 8.1 indicates the speed of the two versions. For the key setup in SNOW 1.0, the IV mode is used as reference, since it also uses 32 clockings in the initialisation phase. This accounts for a more reasonable comparison. The tests

	SNOW 1.0		SNOW 2.0	
Operation	version 1	version 2	version 1	version 2
Key setup	925	-	937	-
Keystream generation	47	34	38	18

Table 8.1: Number of cycles needed for key setup and cycles per word for keystream generation on a Pentium 4 @1.8GHz.

were run on a PC with an Intel 4 processor running at 1.8 GHz, and 512 Mb of memory. Each program was compiled using gcc with optimisation parameter "-O3" and *inline* directives in the code.

Test vectors for SNOW 2.0 can be found in [33].

8.6 Attacks on SNOW 2.0

In the design of SNOW 2.0, much care was taken to enhance the resistance to linear masking attacks. The choice of α and α^{-1} as multiplication elements in the feedback polynomial renders the method of taking the 2^{32}th power of the polynomial and deriving a bitwise relation obsolete, but remains very efficient in a software implementation. Other elements could have been used at the expense of a slower realisation.

Watanabe et al. have recently shown that it is possible to use a mask directly on the linear recurrence [119]. If the LFSR is considered built over the vector field $\mathbb{F}_2^{32}$ instead of over $\mathbb{F}_{2^{32}}$, the multiplications with α and α^{-1} are linear transformations and

hence can be written as matrix multiplications. Let A and A_{inv} denote the matrix for the α and α^{-1} multiplication respectively. In the vector field, the inner product $\odot$ of two vectors is defined (in the usual way) and for any row vector mask Γ, the following equation holds

$$\Gamma \odot (s_{t+16} \oplus A_{inv} \odot s_{t+11} \oplus s_{t+2} \oplus A \odot s_t) = 0, \qquad (8.26)$$

where the state variables $s_{t+i},\ i = 0, 2, 11, 16$ are considered column vectors. Due to the linearity, (8.26) can be rewritten as

$$\Gamma \odot s_{t+16} \oplus (\Gamma \odot A_{inv}) \odot s_{t+11} \oplus \Gamma \odot s_{t+2} \oplus (\Gamma \odot A) \odot s_t = 0. \qquad (8.27)$$

By the same property that allows for a fast implementation of the multiplication, the masks $(\Gamma \odot A)$ and $(\Gamma \odot A_{inv})$ are easily calculated and will essentially be a shifted version of Γ, with some distorted values in the highest and lowest byte respectively.

Now, if a correlation is found within the FSM for a certain mask, Γ, and the correlation in the FSM for the other masks $\Gamma \oplus A$ and $\Gamma \oplus A_{inv}$ are not too small, the exact same technique as in [17] can be used.

In the pre-proceedings version of [119], a mask $\Gamma = 0x03018001$ is given and the corresponding correlation is stated to be $2^{-102.5}$. This result is however incorrect since the mask value for $(\Gamma \odot A)$ is erroneously calculated. In a private email correspondence with the authors, this mistake was confirmed. The best (correct) mask value found is instead given by $\Gamma = 0x0303600c$ and the corresponding correlation is 2^{-113}, resulting in a distinguishing attack of complexity 2^{226}. At the time of writing, it is unclear whether the correct values will appear in the proceedings or not since the printing of the proceedings may already have begun.

8.7 Summary

In this chapter two new stream ciphers, SNOW 1.0 and SNOW 2.0, were presented. Both versions are very fast ciphers with a symbol size of 32, making them suitable for a software implementation on a 32 bit processor. Both version utilises (optionally) an IV mode for fast reinitialisation.

Firstly, SNOW 1.0 was presented, together with a detailed operational description. The key initialisation procedure for both IV mode and standard (non-IV mode) was given. Thereafter, some of the discovered weaknesses of SNOW 1.0 were discussed and the two known attacks were presented.

Next, the second version of the cipher, SNOW 2.0, was presented and some of the design differences from the first version were emphasised together with the expected security improvement. Some implementation aspects, in order to maximise the encryption speed of SNOW 2.0, were also considered.

Finally, an evaluation of the resistance to linear masking attacks (using results from other authors) was given. The conclusion is that SNOW 2.0 is much more resistant to this kind of attacks compared to SNOW 1.0. At present, the best attack on SNOW 2.0 is a distinguishing attack requiring 2^{226} output words and roughly the same time complexity.

9 Concluding remarks

In this thesis, various stream ciphers built using LFSRs are considered. Several new attacks on well-known ciphers are presented and theoretical and simulation results concerning their performance are given. Several interesting problems remain however, especially in the area of distinguishing attacks. Firstly, the framework for these attacks must be clarified. In a recent paper by Rose and Hawkes [101], distinguishing attacks on stream ciphers are discussed from the perspective that, in most cases, these attacks do not have any security implications for the use of the cipher. In [101] it is argued that distinguishing attacks are properly related to the amount of data available, not to the key length. These kind of questions are very important to answer in order to gain comparable public confidence in stream ciphers, as compared to block ciphers.

One often proposed replacement for a dedicated stream cipher is a block cipher in, for example, output feedback mode (OFB) or counter mode (CTR). This circumvents the problem of error propagation for block ciphers used in many other modes (e.g. CBC mode). However, there is a trivial distinguishing attack for a block cipher in counter mode, using $2^{n/2}$ output blocks, where n is the block size of the block cipher. As an example, AES (128 bit block) with a 256 bit key in counter mode, has a distinguishing attack of order 2^{64}. For a stream cipher with a similar key size, this attack would be considered devastating. To fairly compare the security of a stream cipher to that of a block cipher in CTR mode, for example, a common framework is needed.

Also in the *Strategic Roadmap for Cryptology* (STORK) document "Open Problems in Cryptology" [116], distinguishing attacks are discussed. The importance of achieving a firm theoretical understanding of how to build stream ciphers resistant to the kind of linear masking attacks discussed in Chapter 3 and Chapter 8 is pointed out.

The conclusion of the NESSIE project was that none of the submitted stream ciphers achieved the required security level and hence no stream cipher was selected

as a recommendation. Both the SOBER-t family and SNOW 1.0 was removed primarily due to the distinguishing attacks discussed in this thesis. I believe that future applications need the encryption speed provided by dedicated stream ciphers and the cryptographical research community should work towards settling the question of the theoretical and practical implications of a distinguishing attack. I have the impression that there is too large a gap between the theoretical discussions in the research community and the practical needs of different applications. The former should not exclude the latter, even within the research community.

Apart from the distinguishing attacks, the recent developments in algebraic attacks are very interesting and need more attention both from an analysis and a design perspective. The questions regarding the complexity of these attacks, as well as the conditions for successful analysis, are something to add to the agenda for future work.

For the construction of stream ciphers, the toolbox of good, well documented strategies is not as rich as for block ciphers. Most theoretical results are concerned with Boolean combining functions or nonlinear filters. These results are important, but modern stream ciphers must be built such that the encryption speed is much faster than provided by a bitwise output of the keystream. In this area, there is much interesting work to be done. In particular, the new construction presented in this thesis, the SNOW family, would be a suitable cipher to investigate as a more general construction. For example, what properties of the FSM can we quantify and what implications do they have in cryptanalysis? This would be an interesting extension of the design part of this thesis.

Bibliography

[1] R.J. Anderson and M. Roe. Notes on A5. Available at http://jya.com/crack-a5.htm, Accessed August 18, 2003, 1994. 67, 69

[2] F. Armknecht. A linearization attack on the Bluetooth key stream generator. Available at http://eprint.iacr.org/2002/191/, Accessed September 18, 2003, 2002. 93

[3] S. Babbage. A space/time tradeoff in exhaustive search attacks on stream ciphers. In *European Convention on Security and Detection*, number 408 in IEE Conference Publication, 1995. 34, 69

[4] S. Babbage, C. De Cannière, J. Lano, B. Preneel, and J. Vanderwalle. Cryptanalysis of Sober-t32. In T. Johansson, editor, *Fast Software Encryption 2003*, To be published in Lecture Notes in Computer Science. Springer-Verlag, 2003. 51, 58, 59

[5] E. Barkan, E. Biham, and N. Keller. Instant ciphertext only cryptanalysis of GSM encrypted communication. In D. Boneh, editor, *Advances in Cryptology—CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 600–616. Springer-Verlag, 2003. 70

[6] E. Biham and Orr Dunkelman. Cryptanalysis of the A5/1 GSM stream cipher. In B.E. Roy and E. Okamoto, editors, *Progress in Cryptology—INDOCRYPT 2000*, volume 1977 of *Lecture Notes in Computer Science*, pages 43–51. Springer-Verlag, 2000. 34, 70

[7] A. Biryukov and A. Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In T. Okamoto, editor, *Advances in Cryptology—ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 2000. 34

[8] A. Biryukov, A. Shamir, and D. Wagner. Real time cryptanalysis of A5/1 on a PC. In B. Schneier, editor, *Fast Software Encryption 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, 2000. 34, 70, 94

[9] D. Bleichenbacher and S. Patel. SOBER cryptanalysis. In L.R. Knudsen, editor, *Fast Software Encryption'99*, volume 1636 of *Lecture Notes in Computer Science*, pages 305–316. Springer-Verlag, 1999.

[10] SIG Bluetooth. Bluetooth specification. Available at http://www.bluetooth.com, Accessed August 18, 2003, 2003. 17, 31, 83, 86, 87

[11] M. Briceno, I. Goldberg, and D. Wagner. GSM cloning. Available at http://www.isaac.cs.berkeley.edu/isaac/gsm.html, Accessed August 18, 2003, 1998. 67

[12] M. Briceno, I. Goldberg, and D. Wagner. A pedagogical implementation of A5/1. Available at http://jya.com/a51-pi.htm, Accessed August 18, 2003, 1999. 12, 67

[13] A. Canteaut and M. Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In B. Preneel, editor, *Advances in Cryptology—EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 573–588. Springer-Verlag, 2000. 36, 96

[14] V. Chepyzhov, T. Johansson, and B. Smeets. A simple algorithm for fast correlation attacks on stream ciphers. In B. Schneier, editor, *Fast Software Encryption 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 181–195. Springer-Verlag, 2000. 36

[15] V. Chepyzhov and B. Smeets. On a fast correlation attack on certain stream ciphers. In D. W. Davies, editor, *Advances in Cryptology—EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 176–185. Springer-Verlag, 1991. 36

[16] A. Clark, J.D. Golić, and E. Dawson. A comparison of fast correlation attacks. In D. Gollmann, editor, *Fast Software Encryption'96*, volume 1039 of *Lecture Notes in Computer Science*, pages 145–157. Springer-Verlag, 1996.

[17] D. Coppersmith, S. Halevi, and C.S. Jutla. Cryptanalysis of stream ciphers with linear masking. In M. Yung, editor, *Advances in Cryptology—CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 515–532. Springer-Verlag, 2002. 41, 104, 141, 146, 149

[18] D. Coppersmith, H. Krawczyk, and Y. Mansour. The shrinking generator. In D.R. Stinson, editor, *Advances in Cryptology—CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 22–39. Springer-Verlag, 1993. 95, 107, 119

[19] N. Courtois. Higher order correlation attacks, XL algorithm and cryptanalysis of Toyocrypt. In P.J. Lee and C.H. Lim, editors, *Information Security and Cryptology - ICISC 2002*, volume 2587 of *Lecture Notes in Computer Science*, pages 182–199. Springer-Verlag, 2003. 37

[20] N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In B. Preneel, editor, *Advances in Cryptology—EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Springer-Verlag, 2000. 37

[21] N. Courtois and W. Meier. Algebraic attacks on stream ciphers with linear feedback. In E. Biham, editor, *Advances in Cryptology—EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 345–359. Springer-Verlag, 2003. 37

[22] N. Courtois and J. Patarin. About the XL algorithm over GF(2). In M. Joye, editor, *Topics in Cryptology—CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 141–157. Springer-Verlag, 2003. 37

[23] N. Courtois and J. Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In Y. Zheng, editor, *Advances in Cryptology—ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer-Verlag, 2002. 37

[24] T. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley series in Telecommunication. Wiley, 1991. 36, 46, 47, 48, 91, 100

[25] J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer-Verlag, 2002. 9, 143, 147

[26] J. Dhem, F. Koeune, P. Leroux, P. Mestré, J-J. Quisquater, and J. Williams. A practical implementation of the timing attack. Technical Report CG-1998/1, UCL Crypto Group Technical Report Series, 1998. 38

[27] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22:644–654, 1976. 6

[28] C. (Cunsheng) Ding, G. Xiao, and W. Shan. *The Stability Theory of Stream Ciphers*, volume 561. Springer-Verlag, New York, NY, USA, 1991. 41

[29] P. Ekdahl and T. Johansson. SNOW - a new stream cipher. In *Proceedings of First Open NESSIE Workshop*, 2000. 134, 139

[30] P. Ekdahl and T. Johansson. Some results on correlations in the Bluetooth stream cipher. In *Proceedings of 10th Joint Conference on Communications and Coding, Obertauern, Austria*, page 16, 2000. 83, 93

[31] P. Ekdahl and T. Johansson. Another attack on A5/1. In *Proceedings of International Symposium on Information Theory*, page 160. IEEE, 2001. 62

[32] P. Ekdahl and T. Johansson. Distinguishing attacks on SOBER-t16 and SOBER-t32. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 210–224. Springer-Verlag, 2002. 41

[33] P. Ekdahl and T. Johansson. A new version of the stream cipher SNOW. In K. Nyberg and H. Heys, editors, *Selected Areas in Cryptography—SAC 2002*, volume 2595 of *Lecture Notes in Computer Science*, pages 47–61. Springer-Verlag, 2002. 134, 148

[34] P. Ekdahl and T. Johansson. Another attack on A5/1. *IEEE Transactions on Information Theory*, 49(1):284–289, January 2003. 62

[35] P. Ekdahl, T. Johansson, and W. Meier. A note on the self-shrinking generator. In *Proceedings of International Symposium on Information Theory*, page 166. IEEE, 2003. 118

[36] P. Ekdahl, W. Meier, and T. Johansson. Predicting the shrinking generator with fixed connections. In E. Biham, editor, *Advances in Cryptology—EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 330–344. Springer-Verlag, 2003. 96

[37] ETSI. European Telecommunications Standards Institute. Available at http://www.etsi.org/, Accessed October 7, 2003, 2003. 12

[38] N. Ferguson and B. Schneier. *Practical Cryptography*. Wiley Publishing, Inc., 2003. 39

[39] S.R. Fluhrer. E_0. Private email correspondence, 2000. 88

[40] S.R. Fluhrer and S. Lucks. Analysis of the E_0 encryption system. In S. Vaudenay and A.M. Youssef, editors, *Selected Areas in Cryptography—SAC 2001*, volume 2259 of *Lecture Notes in Computer Science*, pages 38–48. Springer-Verlag, 2001. 93

[41] C.G. Günther. Alternating step generators controlled by de Bruijn sequences. In D. Chaum and W.L. Price, editors, *Advances in Cryptology—EUROCRYPT'87*, volume 304 of *Lecture Notes in Computer Science*, pages 91–103. Springer-Verlag, 1988. 31, 34

[42] J.D. Golić. Correlation via linear sequential circuit approximation of combiners with memory. In R.A. Rueppel, editor, *Advances in Cryptology—EUROCRYPT'92*, volume 658 of *Lecture Notes in Computer Science*, pages 113–123. Springer-Verlag, 1993. 31

[43] J.D. Golić. Towards fast correlation attacks on irregularly clocked shift registers. In L.C. Guillou and J-J. Quisquater, editors, *Advances in Cryptology—EUROCRYPT'95*, volume 921 of *Lecture Notes in Computer Science*, pages 248–262. Springer-Verlag, 1995. 108

[44] J.D. Golić. Computation of low-weight parity-check polynomials. *Electronic Letters*, 32(21):1981–1982, October 1996. 90, 96

[45] J.D. Golić. Correlation properties of a general binary combiner with memory. *Journal of Cryptology*, 9(2):111–126, 1996.

[46] J.D. Golić. Linear models for keystream generators. *IEEE Transactions on Computers*, 45(1):41–49, January 1996. 108

[47] J.D. Golić. On the security of nonlinear filter generators. In D. Gollman, editor, *Fast Software Encryption'96*, volume 1039 of *Lecture Notes in Computer Science*, pages 173–188. Springer-Verlag, 1996. 30

[48] J.D. Golić. Cryptanalysis of alleged A5 stream cipher. In W. Fumy, editor, *Advances in Cryptology—EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 239–255. Springer-Verlag, 1997. 69

[49] J.D. Golić. Linear statistical weakness of alleged RC4 keystream generator. In W. Fumy, editor, *Advances in Cryptology—EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 226–238. Springer-Verlag, 1997.

[50] J.D. Golić. Correlation analysis of the shrinking generator. In J. Kilian, editor, *Advances in Cryptology—CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 440–457. Springer-Verlag, 2001. 108

[51] J.D. Golić, V. Bagini, and G. Morgari. Linear cryptanalysis of Bluetooth stream cipher. In L.R. Knudsen, editor, *Advances in Cryptology—EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 238–255. Springer-Verlag, 2002. 92, 93

[52] J.D. Golić and R. Menicocci. A new statistical distinguisher for the shrinking generator. Available at http://eprint.iacr.org/2003/041, Accessed September 29, 2003, 2003. 108

[53] J.D. Golić and L. O'Connor. Embedding and probabilistic correlation attacks on clock-controlled shift registers. In A. De Santis, editor, *Advances in Cryptology—EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 230–243. Springer-Verlag, 1995. 107

[54] S. Halevi, D. Coppersmith, and C.S. Jutla. Scream: A software-efficient stream cipher. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 195–209. Springer-Verlag, 2002. 32, 147

[55] R. Harris. *Enigma*. Random House Inc., 1996. 2

[56] P. Hawkes and G.G. Rose. Primitive specification and supporting documentation for SOBER-t16 submission to NESSIE. In *Proceedings of First Open NESSIE Workshop*, 2000. Available at http://www.cryptonessie.org, Accessed October 5, 2003. 41, 44, 46

[57] P. Hawkes and G.G. Rose. Primitive specification and supporting documentation for SOBER-t32 submission to NESSIE. In *Proceedings of First Open NESSIE Workshop*, 2000. Available at http://www.cryptonessie.org, Accessed October 5, 2003. 41, 44, 46, 148

[58] P. Hawkes and G.G. Rose. Guess-and-determine attacks on SNOW. In K. Nyberg and H. Heys, editors, *Selected Areas in Cryptography—SAC 2002*, volume 2595 of *Lecture Notes in Computer Science*, pages 37–46. Springer-Verlag, 2002. 34, 140

[59] M. Hermelin and K. Nyberg. Correlation properties of the Bluetooth combiner. In J. Song, editor, *Information Security and Cryptology—ICISC'99*, volume 1787 of *Lecture Notes in Computer Science*, pages 17–29. Springer-Verlag, 2000. 88, 92

[60] F.H. Hinsley and A. Stripp. *The Inside Story of Bletchley Park*. Oxford University Press, Reissue, 2001. 2

[61] R. Johannesson and K.Sh. Zigangirov. *Fundamentals of Convolutional Coding*. IEEE Series on Digital and Mobile Communication. IEEE Press, 1999. 100

[62] A.J. Johansson. Theory and use of chaotic oscillators in electronic communication. Licenciate in Engineering Thesis, Department of Applied Electronics, Lund University, 2000. 4

[63] T. Johansson. Reduced complexity correlation attacks on two clock-controlled generators. In K. Ohta and D. Pei, editors, *Advances in Cryptology—ASIACRYPT'98*, volume 1541 of *Lecture Notes in Computer Science*, pages 342–357. Springer-Verlag, 1998. 108

[64] T. Johansson and F. Jönsson. Fast correlation attacks based on turbo code techniques. In M.J. Wiener, editor, *Advances in Cryptology—CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 181–197. Springer-Verlag, 1999. 36

[65] T. Johansson and F. Jönsson. Improved fast correlation attacks on stream ciphers via convolutional codes. In J. Stern, editor, *Advances in Cryptology—EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 347–362. Springer-Verlag, 1999. 36

[66] T. Johansson and F. Jönsson. Fast correlation attacks through reconstruction of linear polynomials. In M. Bellare, editor, *Advances in Cryptology—CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 300–315. Springer-Verlag, 2000. 36, 96

[67] F. Jönsson. *Some Results on Fast Correlation Attacks*. PhD thesis, Lund University, Department of Information Technology, P.O. Box 118, SE–221 00, Lund, Sweden, 2002. 36

[68] D.A. Kahn. *The CodeBreakers: The Story of Secret Writing*. Macmillan Publishing, New York, 1967, 1967. 2

[69] E. Key. An analysis of the structure and complexity of nonlinear binary sequence generators. *IEEE Transactions on Information Theory*, 22:732–736, 1976. 30

[70] A. Kipnis and A. Shamir. Cryptanalysis of the HFE public key cryptosystem. In M.J. Wiener, editor, *Advances in Cryptology—CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 19–30. Springer-Verlag, 1999. 37

[71] L.R. Knudsen, W. Meier, B. Preneel, V. Rijmen, and S. Verdoolaege. Analysis methods for (alleged) RC4. In K. Ohta and D. Pei, editors, *Advances in Cryptology—ASIACRYPT'98*, volume 1998 of *Lecture Notes in Computer Science*, pages 327–341. Springer-Verlag, 1998.

[72] P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Advances in Cryptology—CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer-Verlag, 1996. 38

[73] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology—CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer-Verlag, 1999. 37

[74] M. Krause. BDD-based cryptanalysis of keystream generators. In L.R. Knudsen, editor, *Advances in Cryptology—EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 222–237. Springer-Verlag, 2002. 119

[75] R. Lidl and H. Niederreiter. *Finite Fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2nd edition, 1997. 24

[76] D. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. 13, 39

[77] I. Mantin and A. Shamir. Practical attack on broadcast RC4. In M. Matsui, editor, *Fast Software Encryption 2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 152–164. Springer-Verlag, 2001. 32

[78] J.L. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15:122–127, 1969. 24

[79] M. Matsui. Linear cryptanalysis method for DES cipher. In T. Helleseth, editor, *Advances in Cryptology—EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer-Verlag, 1994. 41, 104

[80] M. Matsui. New block encryption algorithm MISTY. In E. Biham, editor, *Fast Software Encryption'97*, volume 1267 of *Lecture Notes in Computer Science*, pages 54–68. Springer-Verlag, 1997.

[81] R.N. McDonough and A.D. Whalen. *Detection of Signals in Noise*. Academic Press, Inc., 2nd edition, 1995. 46

[82] W. Meier and O. Staffelbach. Fast correlation attacks on stream ciphers. In C.G. Günter, editor, *Advances in Cryptology—EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 301–316. Springer-Verlag, 1988. 36

[83] W. Meier and O. Staffelbach. Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1(3):159–176, 1989. 36

[84] W. Meier and O. Staffelbach. The self-shrinking generator. In A. De Santis, editor, *Advances in Cryptology—EUROCRYPT'94*, volume 905 of *Lecture Notes in Computer Science*, pages 205–214. Springer-Verlag, 1994. 117, 118

[85] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997. 2, 9, 26

[86] M. Mihaljevic. A faster cryptanalysis of the self-shrinking generator. In J. Pieprzyk and J. Seberry, editors, *First Australasian Conference on Information Security and Privacy ACISP'96*, volume 1172 of *Lecture Notes in Computer Science*, pages 182–189. Springer-Verlag, 1996. 34, 118

[87] M. Mihaljevic, M. Fossorier, and H. Imai. A low-complexity and high-performance algorithm for the fast correlation attack. In B. Schneier, editor, *Fast Software Encryption 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 196–212. Springer-Verlag, 2000. 36

[88] M. Mihaljevic and J.D. Golić. A fast iterative algorithm for a shift register initial state reconstruction given the noisy output sequence. In J. Seberry and J. Pieprzyk, editors, *Advances in Cryptology—AUSCRYPT'90*, volume 453 of *Lecture Notes in Computer Science*, pages 165–175. Springer-Verlag, 1990. 36

[89] M. Mihaljevic and J.D. Golić. A comparison of cryptoanalytic principles based on iterative error-correction. In D.W. Davis, editor, *Advances in Cryptology—EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 527–531. Springer-Verlag, 1991. 36

[90] T.T. Moh. On the method of XL and its inefficiency against TTM. Available at http://eprint.iacr.org/2001/047/, Accessed August 18, 2003, 2001. 37

[91] B. Moret. *The Theory of Computation*. Addison-Wesley, 1998. 33

[92] NESSIE. New European Schemes for Signatures, Integrity, and Encryption. Available at http://www.cryptonessie.org, Accessed August 18, 2003, 1999. 41, 133, 139

[93] K. Nyberg. On the construction of highly nonlinear permutations. In R.A. Rueppel, editor, *Advances in Cryptology—EUROCRYPT'92*, volume 658 of *Lecture Notes in Computer Science*, pages 92–98. Springer-Verlag, 1992. 29

[94] E. Pasalic. *On Boolean Functions in Symmetric-Key Ciphers*. PhD thesis, Lund University, Department of Information Technology, P.O. Box 118, SE–221 00, Lund, Sweden, 2003. 28, 29

[95] W.T. Penzhorn. Correlation attacks on stream ciphers: Computing low weight parity checks based on error correction codes. In D. Gollman, editor, *Fast Software Encryption'96*, volume 1039 of *Lecture Notes in Computer Science*, pages 159–172. Springer-Verlag, 1996. 36

[96] W.T. Penzhorn and G.J. Kühn. Computation of low-weight parity checks for correlation attacks on stream ciphers. In C. Boyd, editor, *Cryptography and Coding - 5th IMA Conference*, volume 1025 of *Lecture Notes in Computer Science*, pages 74–83. Springer-Verlag, 1995. 36

[97] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978. 6

[98] P. Rogaway and D. Coppersmith. A software-optimised encryption algorithm. In R.J. Anderson, editor, *Fast Software Encryption'93*, volume 809 of *Lecture Notes in Computer Science*, pages 56–63. Springer-Verlag, 1994. 32

[99] P. Rogaway and D. Coppersmith. A software-optimized encryption algorithm. *Journal of Cryptology*, 11(4):273–287, 1998. 32

[100] S. Roman. *Coding and Information Theory*. Graduate Texts in Mathematics. Springer-Verlag, 1992. 88

[101] G.G. Rose and P. Hawkes. On the applicability of distinguishing attacks against stream ciphers. Available at http://eprint.iacr.org/2002/142, Accessed October 5, 2003, 2002. 151

[102] G.G. Rose and P. Hawkes. Turing: A fast stream cipher. In T. Johansson, editor, *Fast Software Encryption 2003*, To be published in Lecture Notes in Computer Science. Springer-Verlag, 2003. 31

[103] R.A. Rueppel. Correlation immunity and the summation generator. In H.C. Williams, editor, *Advances in Cryptology—CRYPTO'85*, volume 218 of *Lecture Notes in Computer Science*, pages 260–272. Springer-Verlag, 1086. 86

[104] R.A. Rueppel. *Analysis and Design of Stream Ciphers*. Communication and Control Engineering Series. Springer-Verlag, 1986. 107

[105] M-J.O. Saarinen. A time-memory tradeoff attack against LILI-128. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 231–236. Springer-Verlag, 2002. 34

[106] M. Schafheutle. A first report on the stream ciphers SOBER-t16 and SOBER-t32. NESSIE document NES/DOC/SAG/WP3/025/2, NESSIE, Available at http://www.cryptonessie.org/, Accessed August 18, 2003, 2001. 41

[107] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley and Sons, New York, 2nd edition, 1996. 13

[108] H. Sebag-Montefiore. *Enigma: The Battle for the Code*. John Wiley and Sons, New York, 2001. 2

[109] J. Seberry, X.M. Zhang, and Y. Zheng. On constructions and nonlinearity of correlation immune functions. In T. Helleseth, editor, *Advances in Cryptology—EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 181–199. Springer-Verlag, 1993.

[110] J. Seberry, X.M. Zhang, and Y. Zheng. Relationships among nonlinearity criteria. In A. De Santis, editor, *Advances in Cryptology—EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 376–388. Springer-Verlag, 1994. 29

[111] C.E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 27:656–715, 1949. 10

[112] T. Siegenthaler. Correlation-immunity of non-linear combining functions for cryptographic applications. *IEEE Transactions on Information Theory*, 30:776–780, 1984. 27, 34

[113] T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Transactions on Computers*, 34:81–85, 1985. 34

[114] L. Simpson, J.D. Golić, and E. Dawson. A probabilistic correlation attack on the shrinking generator. In C. Boyd and E. Dawson, editors, *Information Security and Privacy '98*, volume 1438 of *Lecture Notes in Computer Science*, pages 147–158. Springer-Verlag, 1998. 107

[115] S. Singh. *The Code Book*. Fourth Estate London, 1999. 2

[116] STORK. Strategic Roadmap for Crypto. Available at http://www.stork.eu.org, Accessed October 5, 2003, 2002. 151

[117] H.L. Van Trees. *Detection Estimation, and Modulation Theory, Part I*. John Wiley and Sons Inc., 1968. 46, 47

[118] D. Wagner. A generalized birthday problem. In M. Yung, editor, *Advances in Cryptology—CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303. Springer-Verlag, 2002. 96

[119] D. Watanabe, A. Biryukov, and C. De Canniere. A distinguishing attack of SNOW 2.0 with linear masking method. In *Selected Areas in Cryptography—SAC 2003*, To be published in Lecture Notes in Computer Science. Springer-Verlag, 2003. 148, 149

[120] D. Watanabe, S. Furuya, H. Yoshida, K. Takaragi, and B. Preneel. A new keystream generator MUGI. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption 2002*, volume 2365 of *Lecture Notes in Computer Science*, pages 179–194. Springer-Verlag, 2002. 32

[121] M.J. Wiener. Efficient DES key search—an update. *CryptoBytes*, 3(2):6–8, Autumn 1997. 9

[122] E. Zenner, M. Krause, and S. Lucks. Improved cryptanalysis of the self-shrinking generator. In V. Varadharajan and Y. Mu, editors, *Australasian Conference on Information Security and Privacy ACISP'01*, volume 2119 of *Lecture Notes in Computer Science*, pages 21–35. Springer-Verlag, 2001. 119