On Cryptographic Properties of LFSR-based Pseudorandom Generators

Inaugural-Dissertation
zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften

der Universität Mannheim

vorgelegt von

Diplom-Wirtschaftsinformatiker Erik Zenner
aus Ludwigshafen (Rhein)

Mannheim, 2004

Dekan:	Professor Dr. Jürgen Potthoff
Referent:	Privatdozent Dr. Stefan Lucks
Korreferent:	Professor Dr. Wolfgang Effelsberg

Tag der mündlichen Prüfung: 24. November 2004

Abstract

Pseudorandom generators (PRGs) are used in modern cryptography to transform a small initial value into a long sequence of seemingly random bits. Many designs for PRGs are based on *linear feedback shift registers* (LFSRs), which can be constructed in such a way as to have optimal statistical and periodical properties.

This thesis discusses construction principles and cryptanalytic attacks against LFSR-based PRGs. After providing a full survey of existing cryptanalytical results, we introduce and analyse the *dynamic linear consistency test* (DLCT), a search-tree based method for reconstructing the inner state of a PRG. We conclude by discussing the role of the inner state size in PRG design, giving lower bounds as well as examples from practice that indicate the necessary size of a secure PRG.

Zusammenfassung

Pseudorandom Generators (PRGs) werden in der modernen Kryptographie verwendet, um einen kleinen Startwert in eine lange Folge scheinbar zufälliger Bits umzuwandeln. Viele Designs für PRGs basieren auf *linear feedback shift registers* (LFSRs), die so gewählt sind, dass sie optimale statistische und periodische Eigenschaften besitzen.

Diese Arbeit diskutiert Konstruktionsprinzipien und kryptanalytische Angriffe gegen LFSR-basierte PRGs. Nachdem wir einen vollständigen Überblick über existierende kryptanalytische Ergebnisse gegeben haben, führen wir den *dynamic linear consistency test* (DLCT) ein und analysieren ihn. Der DLCT ist eine suchbaum-basierte Methode, die den inneren Zustand eines PRGs rekonstruiert. Wir beschließen die Arbeit mit der Diskussion der erforderlichen Zustandsgröße für PRGs, geben untere Schranken an und Beispiele aus der Praxis, die veranschaulichen, welche Größe sichere PRGs haben müssen.

Acknowledgements

Many people have contributed to this thesis, some of them without even being aware of it. First and foremost, I wish to thank my two PhD supervisors, Dr. Stefan Lucks and Prof. Dr. Matthias Krause. Both of them had an open door and an open ear at all times and for all problems. While Stefan was my enthusiastic guide through the world of cryptography[1] and through the hardnesses of the PhD time, Matthias deserves all the credit for being my first teacher in cryptography and for handling all the adminstrative problems of becoming a PhD - including but not limited to making sure that there would always be sufficient money to keep me going[2].

It is impossible to credit all the helpful and friendly people at the university of Mannheim, but I would like to briefly mention two of them. Frederik Armknecht was not only the proof-reader for this thesis, he was also such a great colleague to have around that he actually became a friend. And Prof. Dr. Wolfgang Effelsberg has accompanied my way through the university for almost ten years, his reliability, friendliness, and undying concern for his students will always remain an inspiration to me.

The cryptographic community as a whole deserves my gratitude for being a very friendly and open environment. It still amazes me how even the most hardened and experienced crypto experts still listen to the opinions and ideas of seemingly total beginners, always giving them the credit of potentially being the next Einstein. I would like to express my particular thanks to David Naccache, Greg Rose, and Willi Meier for being supportive in the most unexpected ways.

I would like to thank my wonderful wife Melanie, who completes my life like nobody else could. She has gone with me through all the time of becoming a PhD, and I do not exaggerate by saying that it was her who kept me from becoming just another ivory tower lunatic. I finally want to thank my family, in particular my parents, who have brought me on the way and have given me the support that was necessary to complete the many years of studies that culmulated in this PhD thesis.

[1]Everyone needs a guide in a world where adversaries roam and provably secure algorithms get broken, where keys can be spurious and not even oracles can be trusted, where trapdoors are good and everyone named 'Eve' is evil, where mental poker is played and signatures may be blind, and where one billion years is too short a time to preserve a secret.

[2]This work was partially supported by the Landesgraduiertenförderung Baden-Württemberg between the years 2000 and 2002.

Contents

1 Introduction **1**
1.1 On pseudorandom generators and modern cryptography 1
1.2 On thesis structure and contributions 2
1.3 On notation 3

I Preliminaries **5**

2 Security Model **7**
2.1 Shannon's model 7
2.2 Notions of security 8
2.2.1 Perfect security 9
2.2.2 Asymptotic security 10
2.2.3 Empirical security 10
2.3 Attacker model 11

3 Pseudorandom Generators **15**
3.1 One-time pad and pseudorandom generators 15
3.2 Linear feedback shift registers 17
3.3 Introducing nonlinearity 19

II Survey of Existing Attacks **21**

4 Generic Attacks **23**
4.1 Introduction 23
4.2 Statistical testing 24
4.3 Period and linear complexity 26

5 Specific Attacks **27**
5.1 Two sample generators 27
5.2 Guessing attacks 28
5.3 Algebraic attacks 30
5.4 BDD attacks 33
5.5 Time-Memory-Data tradeoffs 34

6 Correlation Attacks **37**
6.1 Basic correlation attack . . . 37
6.2 Fast correlation attacks . . . 39
6.3 Correlation attacks and memory . . . 42
6.4 Correlation attacks and clock control . . . 43

III Backtracking Attacks **45**

7 The Dynamic Linear Consistency Test **47**
7.1 Generators under consideration . . . 47
7.2 The linear consistency test revisited . . . 48
7.3 A dynamic extension . . . 49
7.4 Computational resources . . . 50
7.5 Further improvements . . . 53

8 Self-Shrinking Generator **55**
8.1 The self-shrinking generator . . . 55
8.1.1 Description . . . 55
8.1.2 Previous work on cryptanalysis . . . 56
8.2 Applying the dynamic LCT attack . . . 58
8.3 Upper bounding the running time . . . 59
8.3.1 Well-formed vs. malformed trees . . . 60
8.3.2 Size of a well-formed tree . . . 62
8.3.3 Worst case considerations . . . 63
8.3.4 Total running time . . . 63
8.4 Experimental results . . . 64
8.5 Conclusions . . . 66

9 Clock-Controlled Generators **67**
9.1 Introduction . . . 67
9.2 On the efficiency of clock control guessing . . . 68
9.3 Application: attacking A5/1 . . . 70
9.4 Other generators . . . 73
9.5 Conclusions . . . 75

IV The Role of the Inner State **77**

10 Deployment of PRGs in Stream Ciphers **79**
10.1 Motivation . . . 79
10.2 Extending the basic model . . . 80
10.3 Outlook . . . 82

11 On the Role of the Inner State Size **85**
11.1 Defining the inner state size 85
11.1.1 Problem illustration 85
11.1.2 Autonomous finite state machines 86
11.1.3 Valid starting states 87
11.1.4 Final definition 88
11.2 Advantages 88
11.2.1 The necessity of large inner states 88
11.2.2 A generic construction 90
11.3 Disadvantages 91

12 Efficient Inner State Size **93**
12.1 Definition 93
12.2 Survey of fielded generators 94
12.3 Comparison of results 96

13 Conclusion **99**

Chapter 1

Introduction

1.1 On pseudorandom generators and modern cryptography

Cryptography: In recent years, cryptography had to deal with an ever increasing number of security issues. While *classical cryptography* was mainly concerned with the making or breaking of secret codes, the field has broadened since the dawn of what is denoted as *modern cryptography* in the early 1970s. In addition to the classical goal of achieving (or compromising) confidentiality of communication, many new tasks like data integrity, message authentication, or non-repudiation have been added.[1]

Nonetheless, providing confidentiality of communication is still a prime goal in modern cryptography. The increase in computational power of potential attackers endangers well-researched algorithms like the data encryption standard (DES [86]) even if they resist the constant tide of new cryptanalytic techniques. At the same time, computationally restricted platforms like smart cards, RFID tags, or sensor nodes require encryption algorithms that are more and more efficient. The gap in power between the potential attacker and the encryption device is widening, driving forward the search for algorithms that are both more efficient and more secure.

Pseudorandom generators: A very efficient building block for encryption algorithms is a *pseudorandom generator*. Such a device transforms a short initial value into a long stream of random-looking output bits. If the generator is to be used in a cryptographic setting, it should be impossible for anyone not knowing the initial value to tell for a given output sequence whether it was produced by the generator or not. Given such a cryptographically sound pseudorandom generator, a secure encryption algorithm can be constructed using standard techniques.

[1]For a more complete list of cryptographic goals, see p. 3 of [86].

Traditionally, pseudorandom generators are deployed in cryptographic hardware like mobile phones, pay-tv decoders, or radio equipment. In particular, many pseudorandom generators based on linear feedback shift registers (LFSRs) have been proposed over the years that are optimised for efficiency in hardware, making them particularly good choices for most of the computationally restricted platforms mentioned above. In a setting where the advanced encryption standard (AES, [28]) is too bulky and even the hardware-efficient tiny encryption algorithm (TEA, [120, 121]) requires too many gates, pseudorandom generators can solve the encryption problem in a very cost-effective way [106].

Surprisingly, this demand is not met by a matching supply. Most pseudorandom generators that have been used or published in the past have known weaknesses. Examples for designs with such problems include (but are not limited to) the A5/1 generator used in the GSM mobile phone standard [133], the E_0 generator from the Bluetooth standard [108], or the well-known RC4 algorithm deployed in a multitude of applications like the TLS/SSL standard for secure internet communication [75]. Even the European NESSIE competition had to turn down all prospective candidates for a pseudorandom generator standard due to a variety of security reasons [91]. Thus, design of efficient and secure pseudorandom generators remains an ongoing challenge and an important field in cryptographic research up to the present day.

1.2 On thesis structure and contributions

Contents: This thesis discusses the design and deployment of pseudorandom generators for cryptographic purposes. To this end, it is necessary to delve into cryptanalysis, which is the activity of searching for security weaknesses of cryptographic algorithms. The underlying goal of cryptanalysis is not destructive, but constructive: Only by improving the understanding of possible problems, it is possible to propose new design criteria for cryptographic systems (see, e.g., [74]). To this end, the thesis is organised as follows:

- Part I defines the general framework of the thesis and introduces important notions and concepts, including those informally mentioned in this introduction.

- Part II surveys the state of the art in cryptanalysis of pseudorandom generators. It describes both techniques against unknown designs (chapter 4) and against generators whose specifications are known (chapters 5-6), giving examples and resource estimates.

- Part III discusses *backtracking attacks*, a particular cryptanalytic technique applicable against LFSR-based pseudorandom generators. After introducing the basic method in chapter 7, its potential is demonstrated against the self-shrinking generator (chapter 8), and an upper bound on

the running time is proven and experimentally verified. A variant of the attack is successfully applied against a whole class of clock-controlled LFSR-based generators in chapter 9 and again, an upper bound on the security of such generators is proven mathematically and confirmed in a trial implementation.

- Part IV analyses the necessary inner state size for pseudorandom generators to be deployed in encryption algorithms. After introducing the necessary terminology in chapter 10, the inner state size is formally defined and its advantages and disadvantages are highlighted in chapter 11. In particular, lower bounds on the necessary size are obtained. While proving a formal upper bound is beyond the current state of cryptographic research, a survey of fielded pseudorandom generators is given in chapter 12, leading to the conclusion that in practice, inner state sizes very close to the theoretical lower bounds should be obtainable.

Publications: The contents of this thesis are based on a number of publications by the author, as follows:

- Parts I and II extend the survey on cryptanalytic techniques provided in [129].
- Chapter 8 of part III is based on the attack against the self-shrinking generator published in [132].
- Chapter 9 of part III on the efficiency of the clock control guessing attack was first discussed in [128].
- Part IV extends the considerations made in [130, 131] on the role of the inner state in stream cipher design.

1.3 On notation

The reader is expected to be familiar with the basic terms and notations both from theoretical and practical computer science. Beyond that, the following mathematical notations will be used throughout the thesis:

- By “iff”, we denote “if and only if”.
- $[n] := \{1, \ldots, n\}$ for $n \in \mathbb{N}^+$.
- If $f : S \to S$ is a function over a range S and $x \in S$, then f^i is defined recursively for $i \in \mathbb{N}^+$ by $f^1(x) := f(x)$ and $f^i(x) := f(f^{i-1}(x))$.
- $\log(x) := \log_2(x)$, i.e., logarithms are always to the base 2 unless indicated otherwise.

- Let D be a probability distribution over the set S. Then $x \in_D S$ means that x is drawn from S according to distribution D. A short notation for this is $x \leftarrow D$. With $D(x)$, we denote the probability for x to be drawn under distribution D.

Part I

Preliminaries

Chapter 2

Security Model

2.1 Shannon's model

Basic setting: The most basic task of cryptography is encryption. The setting was captured by Shannon in [110] as a modification of his well-known communication model, proposed in [109]. Consider two entities, named *sender* and *receiver*, who want to transmit an arbitrary message at an arbitrary point in time in complete privacy. There are two communication channels available:

- The *secret channel* is completely confidential. No information that is transmitted using this channel can be observed by a third party. However, the secret channel has the disadvantage of being available only at fixed points in time (e.g., when sender and receiver meet in person).
- The *public channel* can be observed by any interested third party. Thus, all information transmitted using this channel can be considered public. As opposed to the secret channel, the public channel is available at any time.

It is obvious that a confidential message cannot be sent across the secret channel, since it might not be available at the desired time. Nor can it be sent across the public channel, since it can be observed by third parties.

Encryption schemes: The use of an *encryption scheme* (or *cipher*) is the traditional solution to the above problem. Such a scheme consists of the following components:

1. A set $\mathcal{M}$ of *messages*, a set $\mathcal{C}$ of *ciphertexts* and a set $\mathcal{K}$ of *keys*.
2. A pair of functions $E : \mathcal{K}\times\mathcal{M} \to \mathcal{C}$ and $D : \mathcal{K}\times\mathcal{C} \to \mathcal{M}$, being computable by efficient algorithms and satisfying the following property:

$$D(k, E(k, m)) = m \qquad \forall m \in \mathcal{M},\ k \in \mathcal{K} \tag{2.1}$$

> E is denoted as *encryption function* and D as *decryption function*. Note that in order to meet condition (2.1), $E(k, \cdot)$ has to be a bijective function and $D(k, \cdot)$ its inverse for all $k \in \mathcal{K}$.

In a first step, sender and receiver agree on such an encryption scheme, using the public channel. They also exchange a key $k \in \mathcal{K}$, using the secret channel. Note that from now on, the knowledge about the key is all that distinguishes a legitimate sender and receiver from an arbitrary third party (Kerckhoffs' principle [66]).

Now sender and receiver are prepared to communicate privately as follows. Given a message $m \in \mathcal{M}$, the sender encrypts m under the key k by calculating $c = E(k, m)$. The ciphertext c is then transmitted using the public channel. On the receiver's side, *decryption* is performed by converting c back into $m = D(k, c)$, thus yielding the original message.

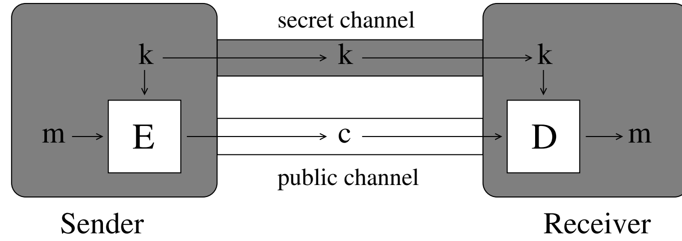


Figure 2.1: Shannon's model

The process of encryption, transmission and decryption is depicted in figure 2.1. Note that all information on white background is visible to all interested parties, while information on gray background is only available to the sender or receiver, respectively. In particular, a casual observer is aware of the functions E and D and of the ciphertext c. In some cases, he may be able to derive information about the message m or the key k from this data (e.g., if $E(k, \cdot)$ is the identical permutation). Informally, such an encryption will be called "insecure". However, in order to find "secure" encryption functions, an informal notion is not enough. Instead, a more precise concept of security is required.

2.2 Notions of security

In order to gain an understanding of security, it is necessary to introduce a malign third party that has access to all public information in the above model and tries to derive some of the secret information from it. Such a third party will be denoted as an *attacker*, the algorithm employed by him as an *attack*. Once the attacker is defined, the notion of security is derived in a straightforward way: A system is secure if the attacker is unable to achieve his goal.

By definition, the attacker knows the encryption and decryption functions E and D. He also has access to all information transmitted over the public channel.

He cannot, however, do anything but listen to the communication channel and do his own computations. In particular, he must not remove, change or add data on the public communication channel. Thus, he is called a *passive attacker*.

Several different definitions for attackers are possible. The most important ones will be briefly reviewed, roughly following a classification given by Rueppel in [102, 103]. In order to describe an attacker, the following questions must be answered:

1. *Type of attack:*
   - *Ciphertext-only attack*: the attacker knows only the ciphertext c.
   - *Known plaintext attack*: the attacker knows c and part of the corresponding message m.
   - *Chosen plaintext attack*: the attacker knows the ciphertext c. In addition, he can choose some messages m_i and obtains the corresponding ciphertexts c_i.
2. *Computational resources:* How many computations can he conduct, and how much memory space is available to him?
3. *Notion of success:* When is he successful? Is it sufficient to find out that a single message candidate m is more likely than others? Does he have to find a unique decryption m for the ciphertext c? Or is he required to find the key k that was used?

2.2.1 Perfect security (information-theoretic model)

Definition: This model was also proposed by Shannon in [110] and considers an all-powerful attacker.

1. *Type of attack:* Ciphertext-only. In addition, the attacker has complete access to the probability distributions of the messages and keys.
2. *Computational resources:* He has unlimited computational power.
3. *Notion of success:* He is already considered successful if, after reading a ciphertext c, the conditional probabilities $p(m|c)$ for the messages differ from the known probabilities $p(m)$.

Discussion: It is not possible to break a perfectly secure encryption scheme without extending Shannon's model by giving the attacker additional capabilities. Furthermore, it can be shown that perfectly secure encryption exists (see section 3.1 for an example). However, a cipher can only be perfectly secure if the key length is not smaller than the entropy of the message that is to be encrypted. Also, the key must never be re-used. Considering that modern applications include the encryption of multimedia web pages, phone calls or online videos with enormous data rates, the infeasibility of managing keys of appropriate length becomes obvious.

2.2.2 Asymptotic security (complexity-theoretic model)

Definition: This model was initiated by a work of Goldwasser and Micali [42] and uses concepts from complexity theory. For an introduction to this field of research, refer to [41].

1. *Type of attack:* Known plaintext or chosen plaintext.
2. *Computational resources:* The attacker is limited to "feasible operations" in an asymptotic sense. Given a *security parameter* λ (often the key length), the computational resources available to the attacker are upper bounded by a function in $O(p(\lambda))$, with p being a polynomial.
3. *Notion of success:* He is successful if he can distinguish the encryption function $E(k, \cdot)$ from a truly random function with significant probability. Again, "significant" is defined in an asymptotic sense, i.e., the success probability must be lower bounded by a function in $\Omega(1/q(\lambda))$, for some polynomial q.

Discussion: In cryptography, asymptotic analysis can be misleading. Recall that an asymptotical lower bound only guarantees that functions in $\Omega(g(\lambda))$ are lower bounded by $c \cdot g(\lambda)$ for some positive c as λ approaches infinity. This implicit assumption, however, does not always hold for realistic values of λ, which tend to be rather small[1]. Thus, an encryption scheme that is secure in an asymptotic sense need not be secure for practical values of λ.

Another frequent misconception about the asymptotic approach is its claim of providing "provable security". In most cases, the security of a scheme is proven *under the assumption* that its building blocks are secure. In this way, the problem of proving the security is not solved, but only transferred to smaller entities. What would be needed in the end is an exponential lower bound on the complexity of a simple computational problem. The task of finding such a lower bound, however, is related (but not identical) to the well-known question of whether or not $P \neq NP$ holds, and remains equally unsolved down to the present day.

2.2.3 Empirical security (system-theoretic model)

Definition: For the given reasons, neither of the first two approaches is very influential in practical cipher design. Instead, a rather vague definition is used as follows.

1. *Type of attack:* Known plaintext or chosen plaintext.
2. *Computational resources:* The attacker's computational resources are limited to the best system that money can buy, plus some security margin.
3. *Notion of success:* He is successful if he can obtain any information about the message m that he did not have previously.

[1] Often, λ denotes the key length, ranging from 40 to 256 for practical systems.

Discussion: The vague definition of the attacker's capabilities in the empirical model implies two problems: It is not known what capabilities the best realistic attacker might have. And, worse yet, algorithm theory does not provide us with the tools to give precise (i.e., non-asymptotic) resource estimates. Thus, the security of an encryption scheme can never be proven in the empirical model - it can only (for particularly bad schemes) be disproven by implementing and demonstrating a working attack.

Knowing that black-or-white answers are usually not possible, research has to fill the shades of gray in between. To this end, all known attacks that are expected to be more efficient than full search over the key space have to be considered, and running time estimates have to be given (e.g., using asymptotics or doing trial runs on a weakened scheme). In this way, potential threats to the security of the system can be identified, even though different experts may end up with differing opinions on whether the cipher is actually secure or not.

Concluding, users can never be sure that an empirically secure scheme is actually unbreakable. There is always the possibiliy that an attack exists but has not been discovered yet, or worse: has been kept secret. Nonetheless, trust in the security of an encryption scheme will increase over time. Again, a comparison with complexity theory seems appropriate. In NP-hardness theory, confidence in the hardness of NP-complete problems is gained from years of trying to find polynomial time algorithms. Similarly, in the empirical model, confidence in a cipher is gained from years of trying to break it.

Given such confidence in known algorithms, however, the security of *new* cryptographic schemes can be specified more readily. If the design and proof techniques proposed by Bellare and Rogaway [6, 5] are used, the minimum resources required to break the new scheme can be specified exactly as long as the minimum resources to break the underlying scheme are known from experience.

2.3 Attacker model

As could be seen from the discussions in the preceding section, all security models combine advantages with drawbacks. Their usefulness differs, depending on the application context. Since our purpose is to present and discuss a number of attack techniques against *practical* encryption schemes, the suitability of the security models can be assessed as follows:

- *Information-theoretic model:* All practical schemes have a limited key size. On the other hand, they must be able to encrypt long messages. Thus, they can never be perfectly secure.

- *Complexity-theoretic model:* For most existing ciphers, asymptotical security can neither be proven nor disproven. The model is more suited for the design of new ciphers[2]. On the other hand, most encryption schemes de-

[2] The same holds for Bellare's and Rogaway's extension to the system-theoretical model or other, more recent notions of security like the *bounded storage model* by Maurer [78].

signed under this paradigm have rather inefficient encryption algorithms, making them unsuitable for practical use.

- *System-theoretic model:* Even though this model is completely unsatisfactory from a theoretical point of view, it seems to be the only suitable way of analysing encryption schemes that are already in existence. In fact, all widely used ciphers (like DES [92], IDEA [72] or AES [93]) have been evaluated in this way, and even systems designed under the complexity-theoretical paradigm tend to use building blocks whose security has been analysed solely under this model. Thus, it seems reasonable to choose the empirical approach on security for our purposes.

Thus, our considerations in this thesis will assume a system-theoretical notion of security. Since this notion is rather vague in its general form, we will use a special instance of an attacker, as follows.

Type of attack: The attacker can mount known-plaintext attacks. This means that the attacker knows the ciphertext c and part of the corresponding message m. Note that such an attacker is stronger than an attacker who is limited to ciphertext-only attacks, increasing the probability of finding security problems.

Computational resources: We assume the attacker to operate on a uniform computational model, like a Turing machine or a Random-access machine, whose computational behaviour is similiar to that of a programmable microprocessor. He is able to conduct all computational operations that run faster than a full search over the key space. Similarly, he is limited to a memory space that is smaller than what would be necessary in order to save all keys.

As a first indication, the computational requirements of an attack are given in asymptotical form. However, in order to avoid the pitfalls of asymptotics (like hidden large factors), all attacks are also implemented. Running time or memory space estimates will be given based on experimental data for small key lengths.

Notion of success: Given the ciphertext c and a piece of the message, there are two possible goals for the attacker:

1. Finding the set $\mathcal{M}' \subseteq \mathcal{M}$ of all *message candidates*. A message $m' \in \mathcal{M}$ is a message candidate if it matches the known piece and if $\exists k \in \mathcal{K}$ such that $E(k, m') = c$.

2. Finding the set $\mathcal{K}' \subseteq \mathcal{K}$ of all *key candidates*. A key candidate k' is defined via
$$k' \in \mathcal{K}' \quad \Leftrightarrow \quad \exists m' \in \mathcal{M}' : E(k', m') = c$$

Note that the second goal is more ambitious. Once $\mathcal{K}'$ is known, it is possible to reconstruct $\mathcal{M}'$ by calculating $m' = D(k', c)$ for all $k' \in \mathcal{K}'$. On the other

hand, deriving the set of key candidates from the set of message candidates is usually not feasible.

Chapter 3

Pseudorandom Generators

3.1 One-time pad and pseudorandom generators

One-time pad (OTP): In [118], G. Vernam introduced a simple encryption algorithm. Let $m, c, k \in \{0,1\}^n$, then the encryption function is $E(k,m) = k \oplus m$ and the corresponding decryption function is $D(k,c) = k \oplus c$. Here, $\oplus$ denotes the bitwise xor of its operands.

It can be proven [110] that this encryption scheme is indeed perfectly secure if a random key is available that is never re-used for a second encryption (thus the name *one-time pad*). This implies, however, that the key must be as long as the message to be encrypted. As mentioned in section 2.2, managing keys of appropriate size is usually not feasible.

Pseudorandom generator (PRG): A *pseudorandom generator* is a function $G : \{0,1\}^l \to \{0,1\}^*$ that expands a short *seed* into a bit sequence of arbitrary length. In order to be of cryptographic interest, G has to be computable by an efficient algorithm. In practice, it is implemented by a finite state machine with output, as displayed in figure 3.1. The components of such a generator are (see, e.g., [100]):

1. An inner state $S_i \in \{0,1\}^l$,
2. an update function $f : \{0,1\}^l \to \{0,1\}^l$ that modifies the inner state between two outputs, and
3. an output function $g : \{0,1\}^v \to \{0,1\}$, $v \leq l$, that computes the next output bit from (part of) the current inner state.

Note that the seed value S_0 and the relation $S_i = f(S_{i-1})$ form a recurrence, defining the sequence of all inner states that the generator assumes over time. Also note that the generator can assume at most 2^l different inner states, yielding an upper bound on the least period of 2^l.

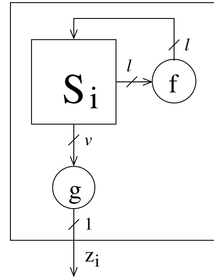


Figure 3.1: Pseudorandom Generator

Deployment of PRG: Given a PRG G, a seed value S_0 can be expanded into an output stream $z = G(k)$ of arbitrary length. This allows us to construct a *pseudo-OTP*, using an encryption function $E(k, m) = z \oplus m$ and a corresponding decryption function $D(k, m) = z \oplus c$, as described in figure 3.2.

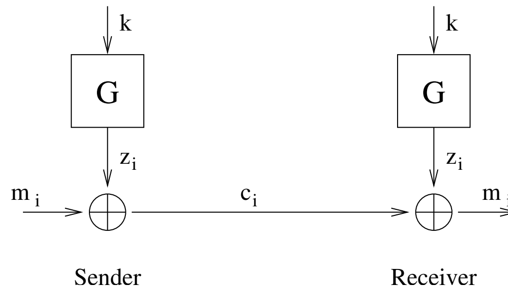


Figure 3.2: Deployment of Pseudorandom Generator

Note that in general, the seed S_0 must not be confused with the key k. In practice, S_0 is generated from k (and possibly some additional information) by some initialisation function. Thus, overall security depends both on this initialisation function and on G. More details on the transformation of k into S_0 will be given in chapters 10 to 12.

For the moment, however, hold in mind that for cryptographic systems, every component should be as strong as possible (cf., e.g., [33]), independently of the other building blocks. Thus, when considering the security of the PRG, the existence of an initialisation function can be ignored, assuming instead that the seed is equal to the key, i.e., $S_0 = k \in \{0, 1\}^l$. For this reason, in the next sections, the terms "seed" and "key" will be used synonymously when considering the security of a PRG.

Security of a pseudorandom generator: In cryptography, a pseudorandom generator G is secure iff a pseudo-OTP using G is secure.

Remember that the attacker can mount known-plaintext attacks, meaning that he knows the ciphertext $c = c_1, \ldots, c_n$ and some message bits $m_{i_1}, \ldots, m_{i_s}$, where $\{i_1, \ldots, i_s\} \subset [n]$. Note that this is equivalent to giving the attacker the corresponding output bits $z_{i_1}, \ldots, z_{i_s}$ right away.

Success is defined as the ability to find either the set of message candidates or the set of key candidates.

1. Finding the set $\mathcal{M}'$ of consistent messages is equivalent to finding the set $\mathcal{Z}'$ of consistent output streams, which is defined as follows:

   $$z' \in \mathcal{Z}' \quad \Leftrightarrow \quad z'_i = z_i \;\; \forall i \in \{i_1, \ldots, i_s\} \quad \text{and} \quad \exists k' \in \mathcal{K} : G(k') = z'$$

   Such an attack is sometimes denoted as *prediction attack*, since its goal is to predict the unknown output bits.

2. If finding the set $\mathcal{K}'$ of consistent keys is possible, it can be reconstructed from $z_{i_1}, \ldots, z_{i_s}$ directly. Thus, the set $\mathcal{K}'$ of consistent keys can be redefined by

   $$k' \in \mathcal{K}' \quad \Leftrightarrow \quad G_i(k') = z_i \;\; \forall i \in \{i_1, \ldots, i_s\} \;,$$

   where $G_i(k)$ denotes the i-th output bit of generator G under key k. This attack is also denoted as *key reconstruction attack*.

3.2 Linear feedback shift registers

Sequences from linear recurrences: Remember that the sequence of inner states $(S_0, S_1, \ldots)$ is defined recursively via $S_0 = k, S_i = f(S_{i-1})$. It would be desireable to choose f such that the least period of the sequence $(S_0, S_1, \ldots)$ is 2^l, i.e., $S_{2^l} = S_0$ and $S_j \neq S_0$ for $0 < j < 2^l$.

A class of recursions that is particularly well understood are *linear recursions*. A linear recursion is defined by a matrix M via

$$S_i = M \cdot S_{i-1}.$$

Note that no linear recursion can iterate through all 2^l possible states, since for all M, it holds that $M \cdot \vec{0} = \vec{0}$, where $\vec{0}$ is the all-zero vector. On the other hand, if the seed $\vec{0}$ is disallowed, it is possible to construct linear recurrences that iterate through all of the remaining $2^l - 1$ inner states.

Linear feedback shift registers (LFSRs): Let $S_i = (s_0^i, \ldots, s_{l-1}^i)$ for arbitrary $i \geq 1$. Consider a special kind of linear recursion, as defined by the following matrix operation:

$$\begin{pmatrix} s_0^i \\ s_1^i \\ \vdots \\ s_{l-2}^i \\ s_{l-1}^i \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \\ a_0 & a_1 & a_2 & \ldots & a_{l-1} \end{pmatrix} \cdot \begin{pmatrix} s_0^{i-1} \\ s_1^{i-1} \\ \vdots \\ s_{l-2}^{i-1} \\ s_{l-1}^{i-1} \end{pmatrix}$$

A more intuitive description of the recursion is as follows:

$$s_j^i = \begin{cases} s_{j+1}^{i-1} & \text{if } 0 \leq j < l-1 \\ \sum_{k=0}^{l-1} a_k s_k^{i-1} & \text{if } j = l-1 \end{cases}$$

This means that the bits of the inner state are shifted to the left, as displayed in figure 3.3, with the leftmost bit being discarded and the rightmost bit being replaced by a linear combination of the previous inner state bits. Computation of n output bits takes $O(l \cdot n)$ computational steps and is easily parallelised in hardware. The overall construction is denoted as *linear feedback shift register* (LFSR).

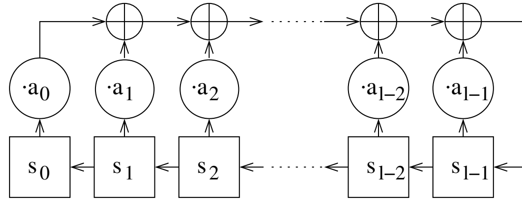


Figure 3.3: Linear Feedback Shift Register

LFSRs and m-sequences: Linear feedback shift registers are mathematically well understood. In particular, the *feedback vector* $(a_0, a_1, \ldots, a_{l-1})$ can be chosen in such a way that the sequence $(S_0, S_1, \ldots)$ iterates through all $2^l - 1$ possible inner states[1]. This makes maximum period LFSRs good building blocks for pseudorandom generators.

Consider the immediate use of such an LFSR as pseudorandom generator, creating the output sequence via $z_i = g(S_i) := s_k^i$ for a fixed $k, 0 \leq k \leq l-1$. It can be shown that the resulting sequence satisfies Golomb's criteria [56], which are defined as follows:

- The output sequence has the same period as the inner states, i.e., $2^l - 1$.
- Fix an arbitray integer r, $1 \leq r \leq l$, and consider a full period of output bits. Then every bit pattern of length r occurs exactly 2^{l-r} times, with the exception of 0^r, which occurs $2^{l-r} - 1$ times. The sequence is said to have *ideal statistics*.
- Consider one full period of the output sequence and shift it cyclically by r positions, $1 \leq r < 2^l - 1$. Then the Hamming distance between the original sequence and its shifted versions is $2^{l-1} - 1$ for all shifts r.

A sequence meeting the above requirements is sometimes denoted as maximal sequence or *m-sequence*, and the generating LFSR is called *m-LFSR*.

[1] For a proof and further details on LFSRs, cf. [56].

Cryptographic limitations: Notwithstanding the good statistical properties, m-sequences do not make good output streams. Note that the dependency between the output bits and the inner state S_0 can be modelled by a system of linear equations, implying the following attacks:

- If the attacker knows the feedback vector $(a_0, \ldots, a_{l-1})$, he can reconstruct the seed S_0 from l arbitrary output bits by solving a system of linear equations. This can be done in $O(l^3)$ computational steps (compared to $O(l^2)$ steps for the generation of l bits) and is feasible for any realistic parameter l.
- If the feedback vector is unknown, the seed S_0 can be reconstructed given $2l$ consecutive output bits, solving a system of $2l$ linear equations. Thus, this attack requires $O(l^3)$ computational steps too, being slower than the attack with known feedback only by a small constant factor.

Concluding, m-LFSRs can be a useful building block for PRG, but some further work is required to prevent attacks that make use of the inherent linearity.

3.3 Introducing nonlinearity

If the update function of a PRG is modelled by an m-LFSR, nonlinearity has to be introduced into the output stream. In cryptographic literature and practice, there are a number of standard techniques that can be used to transform an m-sequence into a highly nonlinear output sequence. Note that all techniques presented below can be combined in the construction of a PRG.

Nonlinear filtering: The most obvious construction uses an m-LFSR to model the update function f, i.e., $f(S) = M \cdot S$ for an LFSR-type matrix M. In this case, the only possibility to introduce nonlinearity into the output stream is the use of a nonlinear output function g. Such a generator is denoted as *filtering generator* [103].

Nonlinear combination: A similar approach is the use of two or more m-LFSRs with pairwise differing lengths and feedback vectors. In this design, the output function g uses part of the inner states of all LFSRs in order to generate the output. Such a generator is called *combination generator* [103].

Nonlinear update: Filtering and combination generators have strictly linear inner states; nonlinearity is introduced using the output function g. It is, however, also possible to add nonlinearity to the inner states without sacrificing the advantages of m-LFSRs. In this case, memory is partitioned in l_1 linear and l_2 nonlinear bits, with $l_1 + l_2 = l$. There are two update functions, where $f_1 : \{0,1\}^{l_1} \to \{0,1\}^{l_1}$ is an LFSR-type matrix, while $f_2 : \{0,1\}^{l} \to \{0,1\}^{l_2}$ is a suitably chosen nonlinear function. Note that in order for the output stream to be nonlinear, the output function must use at least some of the nonlinear

bits. For historical reasons, PRG of this type are denoted as *generators with memory.*

Irregular clocking: Another method of introducing nonlinearity directly into the inner state is irregular clocking. For such generators, the inner state S_i is segmented into several substates $S_{i,1}, \dots, S_{i,q}$, e.g., by considering each LFSR as a separate substate. Each substate has its own update function $f_1, \dots, f_q$. A clock control function $c : \{0,1\}^l \to \mathbb{Z}^q$ determines how often each update function is applied before the next valid inner state is reached, i.e., for $c(S_{i-1}) = (c_1, \dots, c_q)$, the next inner state is determined by

$$S_i = ({f_1}^{c_1}(S_{i-1,1}), \dots, {f_q}^{c_q}(S_{i-1,q}))$$

Note that in some cases, c_j may be negative. Surprisingly, even very simple designs with irregular clocking (e.g., with $q = 2$) lead to strongly nonlinear inner state recurrences, making this technique a powerful tool in PRG design. The general class of PRG using irregular clocking is denoted as *clock control generators.*

Part II

Survey of Existing Attacks

Chapter 4

Generic Attacks

4.1 Introduction

Two-step security analysis: In chapter 2, the attacker was defined as operating in the empirical security model. In order to provide security against such an attacker, the designer of a pseudorandom generator has to provide two kinds of analysis:

1. In a first step, the security of the generator against previously known attacks has to be tested. In order to do so, the designer has to be aware of known attack techniques against pseudorandom generators. Only if none of these techniques can be applied successfully to the new generator, the second phase is entered.

2. In the second phase, the designer has to search for new attacks against his specific generator. Since this task is much more difficult than the application of existing attacks, the designer is well advised to get the help of as many experts as he can find. This is true even if the designer himself is an expert, simply because four eyes see more than two.

Note that the diligent and successful completion of both analysis phases does not provide a security guarantee. Resistance against attacks both old and new is a necessary, but not a sufficient criterion for security.

Generic vs. specific attacks: The first part of this thesis will provide a survey of existing attack techniques against pseudorandom generators. In this context, we distinguish two broad classes of attacks:

- *Generic attacks* are applicable even if the attacker does not know the design of the generator. Most generic attacks date back to the beginnings of public cryptographic research, and for many years, the security of pseudorandom generators was measured against them. Generic attacks will be discussed in the current chapter.

- In contrast, in order to apply *specific attacks*, the attacker has to know the internal structure of the generator. Specific attacks are more recent than the generic ones, and some of them can only be directed against certain classes of generators. They will be discussed in chapters 5 to 6. Furthermore, the novel attack techniques presented in part III of this thesis fall into this category as well.

4.2 Statistical testing

First and foremost, the attacker must not be able to observe any regularities in the output stream. If this was the case, he could predict additional bits of the output sequence, yielding a prediction attack. For this reason, it must not be possible to tell the output stream apart from a truly random sequence. This concept is formalised by the notion of statistical hypothesis testing.

Hypothesis testing: Let $z \in \{0,1\}^s$ be a bitstring that is either random (hypothesis H_0) or pseudo-random (hypothesis H_1). Further, let $X : \{0,1\}^s \to \mathbb{R}$ be a random variable that can be efficiently computed from z. Then denote the probability distribution of $X(z)$ by D_0 if z was drawn according to H_0, and by D_1 otherwise.

Given an observation x for the random variable $X(z)$, the attacker's goal is to decide whether x was drawn according to distribution D_0 or D_1. Note that this is only possible if the distributions D_0 and D_1 differ. This difference is measured by the *statistical distance* between D_0 and D_1, which can be defined as

$$|D_0 - D_1| = \sum_x |D_0(x) - D_1(x)| \ .$$

The larger the statistical distance is, the weaker is the pseudorandom generator. For a distinguishing attack, a *decision rule* $R : \mathbb{R} \to \{0,1\}$ is used to decide whether a given x was drawn according to D_0 or D_1. The quality of such a decision rule is measured by the advantage

$$\mathrm{adv}(R) = \frac{1}{2} \left| \Pr_{x \leftarrow D_0}(R(x) = 0) - \Pr_{x \leftarrow D_1}(R(x) = 0) \right| \ .$$

The decision rule that achieves the greatest advantage is the maximum likelihood rule, which outputs 0 iff $D_0(x) > D_1(x)$. The advantage achieved by this rule is $\frac{1}{4} \cdot |D_1 - D_2|$. Note, however, that the distributions $D_0(x)$ and $D_1(x)$ must be known in order to implement this rule, which is not always the case.

Sample randomness tests: A *statistical test* is a combination of a suitable random variable and decision rule, where both must be efficiently computable. Several statistical tests have been proposed that should be passed by every pseudorandom generator. As an example, the following tests due to Beker and Piper [4] formed the basis for the FIPS 140-2 statistical tests of randomness

[94], until the specification of concrete test procedures was removed from the standard in december 2002.

- *Poker test:* For values $m \ll s$, a random bitstring is expected to contain all possible bit patterns i of length m equally often. Let $k := \lfloor \frac{s}{m} \rfloor$ and divide the bitstring z into k non-overlapping parts of length m. If the number of occurrences of i is denoted by s_i, the random variable

$$X = \frac{2^m}{k} \left(\sum_{i=(0..0)}^{(1..1)} s_i^2 \right) - k \ ,$$

  follows a χ^2 distribution[1] with $2^m - 1$ degrees of freedom for random sequences.

  A special case is $m = 1$, which tests whether the number of zeroes roughly equals the number of ones in the sequence. With $m = 1$ and $k = s$, the random variable simplifies to

$$X = \frac{(s_0 - s_1)^2}{s}$$

  and follows a χ^2 distribution with 1 degree of freedom. This case is also known as *frequency test* or *monobit test*.

- *Runs test:* A *run* is a maximum-length substring that consists only of zeroes (*gap*) or ones (*block*). Denote by G_i and B_i the number of gaps and blocks of length i, respectively. Then for $k \ll s$ and $1 \leq i \leq k$, a random sequence should have the property that G_i and B_i should be close to the expected value, which is $e_i = (s - i + 3)/2^{i+2}$. Thus, the random variable

$$X = \sum_{i=1}^{k} \frac{(B_i - e_i)^2}{e_i} + \sum_{i=1}^{k} \frac{(G_i - e_i)^2}{e_i}$$

  should follow a χ^2 distribution with $2k - 2$ degrees of freedom for random bitstrings.

- *Autocorrelation test:* A random sequence should bear no similarity to its shifted versions. For a given shift d, $1 \leq d \leq \lfloor \frac{n}{2} \rfloor$, this is measured by the hamming distance

$$D_d = \sum_{i=0}^{s-d-1} z_i \oplus z_{i+d} \ .$$

  For random bitstrings, D_d is $(\frac{s-d}{2}, \frac{s-d}{4})$ normal distributed, implying that

$$X = 2 \left(D_d - \frac{s-d}{2} \right) / \sqrt{s-d}$$

  follows a standard normal distribution.

[1] Details on the χ^2 distribution and its use for testing purposes can be found in any textbook on statistics.

Other tests have been proposed by Golomb [56], Beker and Piper [4], Knuth [68], Maurer [77], and many others. Again, remember from section 4.1 that such tests are necessary, but by no means sufficient criteria for good pseudorandom sequences.

4.3 Period and linear complexity

Remember that a pseudorandom generator is a finite state machine with at most 2^l inner states. Thus, an output sequence generated by such a generator must become cyclic after at most 2^l output bits. As a consequence, the more significant bits of the sequence can be modelled as a function of the less significant bits by a suitable recurrence relation.

Let R be a recurrence relation with $x_i = R(x_{i-1}, \ldots, x_{i-k})$. Then k is denoted as the *length* of R. If an output stream can be described by such a relation and if the attacker has at least k consecutive output bits at his disposal, he can easily predict the full output sequence. In the following, two particularly simple types of recurrences will be discussed.

Period: Consider an infinite bitstream $z = (z_0, z_1, \ldots)$. If there exist values $\rho, \theta \in \mathbb{N}$ such that $z_i = z_{i+\rho}$ for all $i \geq \theta$, the sequence is said to be ρ-periodic with pre-period θ. The smallest value ρ for which z is ρ-periodic is called the *least period* or simply *period* of z.

Since pseudorandom generators have at most 2^l inner states, it holds that $\theta + \rho \leq 2^l$. Since for all $i \geq \theta + \rho$, the attacker can use the recurrence $z_i = z_{i-\rho}$ to predict additional bits of the output stream, it is paramount that at most $\theta + \rho$ output bits are generated with the same key. Thus, all sequences generated by a pseudorandom generator should have large periods.

Linear complexity In some cases, the periodic part of the sequence z can be described by a linear recurrence relation R such that $k < \rho$. The length k of the shortest such linear recurrence is denoted as *linear complexity*[2] $\mathrm{LC}(z)$. Put another way, the linear complexity is the length of the smallest LFSR that generates the sequence z. Note that the period recurrence itself is a linear recurrence, such that $\mathrm{LC}(z) \leq \rho$.

There exists an efficient algorithm by Berlekamp and Massey [76] that constructs the shortest linear recurrence describing z. Since this algorithm needs only $2 \cdot \mathrm{LC}(z)$ output bits and takes only $O(\mathrm{LC}(z)^2)$ computational steps, an attacker can easily simulate an output sequence with a low linear complexity by a linear relation. Thus, high linear complexity is a necessary requirement for all sequences generated by a pseudorandom generator.

[2] Sometimes also the term *linear equivalence* is used.

Chapter 5

Specific Attacks

5.1 Two sample generators

As discussed in section 4, in order to apply a specific attack to a generator, its inner workings have to be known. This, however, does not mean that the wheel has to be re-invented for every generator. A number of attack techniques are known that can be used against classes of PRGs, implying that resistance against such attacks is a minimal requirement for any such generator.

In order to illustrate the workings of these attacks, two simple pseudorandom generators will be considered in the next two chapters: The Geffe generator and the $\{1, 2\}$-clocked generator. For both generators, the underlying LFSRs can be chosen such that resistance against the generic attacks presented in chapter 4 is provided.

Throughout the descriptions, LFSRs are denoted by capital letters. The length of LFSR X is denoted as l_X, and the output sequence generated by X is $x = (x_0, x_1, \ldots)$.

Geffe generator: The Geffe generator was introduced in [40]. It is a nonlinear combination generator, as discussed in subsection 3.3. It consists of three m-LFSRs A, B and C, producing m-sequences a, b and c. Output bit z_i is generated using the Boolean function

$$\begin{aligned} z_i &= (c_i \wedge a_i) \vee (\overline{c_i} \wedge b_i) \\ &= c_i \cdot a_i \oplus c_i \cdot b_i \oplus b_i \ . \end{aligned}$$

This means that $z_i = a_i$ if $c_i = 1$, and $z_i = b_i$ otherwise. Thus, c_i is denoted as control bit and register C as control register. For an illustration of the generator, see figure 5.1.

$\{1, 2\}$-clocked generator: This generator is a special case of the basic clock-controlled shift register arrangement proposed by Gollmann and Chambers in [55]. It consists of two m-LFSRs A and C, where C is called the control register

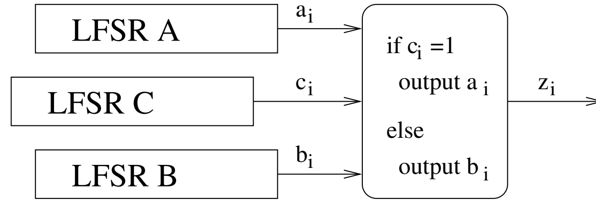


Figure 5.1: Geffe Generator

of the arrangement. If $c_i = 0$, register A is clocked by one step; otherwise, A is clocked by two steps. Thus, the generator is sometimes named *step1-step2 generator*. The output is taken directly from register A, meaning that $z_i = a_{s(i)}$, where $s(i) = \sum_{k=0}^{i}(c_k + 1) = (\sum_{k=0}^{i} c_k) + i + 1$. An alternative description is as follows: Given the inner bitstream a and a pointer to the last used bit, the next output bit z_i is obtained by deleting c_i bits from a and using the next available bit as output. An illustration of the generator is given in figure 5.2.

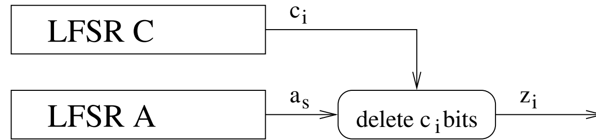


Figure 5.2: $\{1, 2\}$-Clocked Generator

5.2 Guessing attacks

Brute force search: In the context of pseudorandom generators, a brute force search is conducted by guessing all l bit of the inner state. For each of his guesses, the attacker runs the generator to generate l output bits and compares the result with the known output stream. If they differ in at least one bit, the guess is discarded as being wrong. Otherwise, the guess is added to the set of *key candidates*. For a generator that passes the standard statistical tests (see section 4.2), the number of false guesses in this set should be close to zero if the number of output bits available to the attacker is large enough. If there exists more than one key candidate, the correct one is determined by running the generator to decrypt all the message.

Remember from section 2.3 that the attacker is not able to conduct a complete brute force attack. Thus, valid attacks in our security model must use strictly less computational steps than are necessary to run the generator 2^l times. One option for such an attack is as follows: The attacker conducts a

brute force attack over *part* of the inner state, tries to derive as much information as possible, and carries on. The following are examples for such attacks.

Guess-and-verify: First, the attacker guesses $l' < l$ bit of the inner state. If it is possible to efficiently discard a fraction q of these guesses as being wrong ($0 < q < 1$), the attacker is left with only $(1-q) \cdot 2^{l'}$ candidate partial guesses. If for each such candidate, he does a complete search over the remaining $l - l'$ bit, he ends up with a total running time of $2^{l'} + (1-q) \cdot 2^{l'} \cdot 2^{l-l'} = 2^{l'} + (1-q) \cdot 2^{l}$ computational steps. If $q > 2^{l'-l}$, the computational effort for this attack is strictly less than 2^l. To illustrate, consider the following attack on the Geffe generator:

- *Geffe generator:* The attacker guesses the inner states of registers A and B. Thus, he can construct the complete sequences a and b. Whenever $a_i = b_i \neq z_i$ for a given index i, he has identified a wrong guess. Given enough output stream (slightly more than $4 \cdot (l_A + l_B)$ bit), he can uniquely identify the correct seed for A and B amongst the guesses. It only remains to do a brute force search over C, yielding a total running time of $2^{l_A + l_B} + 2^{l_C}$. If $l_A = l_B = l_C = l/3$, the attack takes roughly $2^{2l/3}$ computational steps.

Guess-and-determine: Here, too, the attacker guesses $l' < l$ bit of the seed. Instead of verifying directly, however, he tries to construct additional parts of the inner state from the output stream. Only after doing so, the brute force search is completed by guessing the missing part of the seed. If on the average, the attacker derives m bit from the output stream, this attack takes roughly $2^{l'} \cdot 2^{l-l'-m} = 2^{l-m}$ computational steps. As an example, consider the Geffe and $\{1, 2\}$-clocked generators:

- *Geffe generator:* The attacker guesses the complete inner state of register C. Now, however, he does know which output bit z_i stems from which register A or B. As long as these bits are part of the seed, the attacker does not have to guess this part anymore. Assuming that $l_A = l_B = l_C = l/3$, the attacker obtains an average of $l/3$ output bits that can be written directly into registers A and B. Thus, the overall work effort of the attack is only $2^{l-l/3} = 2^{2l/3}$.

- $\{1, 2\}$*-clocked generator:* Here, too, the attacker guesses the complete inner state of register C. In this way, for each bit z_i, he can determine the corresponding position k in the inner bitstream $a = (a_0, a_1, \ldots)$. Note that on the average, 2 out of 3 seed bits of register A can be read from the output stream. Thus, the attacker has a total effort of $2^{l-2l_A/3}$ guesses. If $l_A = l_C = l/2$, this yields a computational effort of $2^{2l/3}$ generator runs.

Linear consistency test The linear consistency test (LCT) was proposed by Zeng, Yang and Rao [125]. It can be considered as a combination of the above guessing attacks, making use of the linearity of the inner bitstreams. To carry

out the test, the attacker guesses $l' < l$ bit of the inner state in such a way that the relationship between the remaining bits and the output bits can be modelled by a system of linear equations. If this system is contradictory, the initial guess is discarded. If it has maximum rank, it can be solved, yielding the unknown bits of the key candidate. On the other hand, if some linear equations are linearly dependend on the others, it is usually possible to construct additional equations until the system has full rank and can be solved. Remembering that solving a system of linear equations in n unknowns requires $O(n^3)$ computational steps using Gauss' elimination algorithm, it follows that the running time of this attack is in $O(2^{l'} \cdot (l - l')^3)$. For illustration, consider the following examples:

- *Geffe generator:* As in the case of a guess-and-determine attack, the attacker guesses the full inner state of register C. Thus, for each output bit z_i, he knows whether it stems from the inner bitstream a or b. In a first step, consider l_A output bits that stem from register A, i.e., $a_i = z_i$. Now note that each bit a_i can be written as linear combination of the initial state $(a_0, \ldots, a_{l_A-1})$, yielding one linear equation. Given slightly more than l_A such output bits, the resulting system of equations is either contradictory or has full rank with high probability. Thus, the attacker can either discard his guess for C, or he can construct the complete inner state of register A. In a second step, he proceeds analogously for register B. Thus, the overall running time of the attack is in $O(({l_A}^3 + {l_B}^3) \cdot 2^{l_C})$.
- $\{1, 2\}$*-clocked generator:* Analogously, the attacker guesses the full inner state of register C. For each output bit z_i, this yields an index j such that $z_i = a_j$. Again, if all a_j are seen as linear combinations of the initial state $(a_0, \ldots, a_{l_A-1})$, slightly more than l_A output bits should suffice to construct a system of linear equations that is either contradictory or can be solved uniquely. The running time of this attack is in $O({l_A}^3 \cdot 2^{l_C})$.

A more detailed description and important extensions of the linear consistency test will be given in chapter 7.

5.3 Algebraic attacks

Preliminaries: While the linear consistency test uses linear equations to reconstruct the seed of a pseudorandom generator, algebraic attacks use *nonlinear equations*. A nonlinear equation in $x_1, \ldots, x_l$ is of the form

$$\bigoplus_{i=1}^{n} M_i = c \ ,$$

where $c \in \{0, 1\}$ is a constant, n is a positive integer and the M_i are monomials of the form

$$M_i = \prod_{j=1}^{l} {x_j}^{c(i,j)} \ , \qquad c(i, j) \in \{0, 1\} \ .$$

The degree d_i of a monomial M_i is defined as $d_i = \sum_{j=1}^{l} c(i,j)$. The degree of an equation is the maximum over all monomial degrees d_i in the equation. Analogously, the degree of a system of equations is the maximum over all monomial degrees in the equation system.

In principle, nonlinear equations can be used to describe each output bit as a nonlinear combination of the generator's seed. There are, however, two problems associated with this approach:

- While a linear equation in variables $x_1, \ldots, x_l$ can have at most l monomials, a nonlinear equation of degree d has up to

$$N_d := \sum_{k=1}^{d} \binom{l}{k} \in O(l^d)$$

  monomials. In the worst case, for $d = l$, up to 2^l monomials can occur in one single equation. Thus, working with nonlinear equations can only be efficient if the number of monomials in each equations is not too large.

- Even worse, solving systems of nonlinear equations is known to be NP-hard [39]. Thus, no algorithm that efficiently solves all systems of nonlinear equations exists under the current state of research in computer science. On the other hand, finding such a universal algorithm is not the attacker's goal anyway. In our model, he is successful if he can solve a significant part of the systems of equations that occur in this special context.

The linearisation technique: Assume that the attacker has a system of nonlinear equations at his disposal, where each equation describes the dependency between one output bit and the corresponding seed bits. He could try to solve this system by *linearisation*, replacing each nonlinear monomial by a single dummy variable. In this way, he ends up with a system of linear equations which can be solved using standard techniques like the Gaussian eliminiation algorithm. Finally, the dummy variables have to be replaced by the original monomials again, in the hope that a unique solution (or at least a small set of solution candidates) can be identified.

Note that during linearisation, the number of variables in the system of equations increases dramatically. Thus, the attacker needs up to N_d linearly independent linearised equations, i.e., the number of required output bits can be very large. At the same time, the attacker is throwing away valuable information contained in the monomials of high degree. For an example, consider the information loss when replacing the nonlinear equation $x_1 x_2 x_3 x_4 = 1$ by the linearised equation $M_1 = 1$.

The extension technique: An important improvement over mere linearisation is the extension technique proposed by Kipnis and Shamir [67]. Given a system of nonlinear equations, the attacker constructs additional equations by multiplying the existing ones with monomials of small degree. If the degree of

the resulting equation is not greater than a specified threshold, the equation is added to the system of equations. In this way, a nonlinear system with a lot of redundant information is generated.

In a second step, the extended system of equations is linearised as described above. This time, however, the attacker has a lot more equations at his disposal, reducing the number of output bits required. Still, for the attack to work, the original system of equations has to be over-specified. In [25], some evidence (though no formal proof) is given that the attack requires more than l output bits, but that the number of additional bits is small.

Sample attack: Remember that for the *Geffe generator*, an output bit z_i can be described by the nonlinear equation

$$z_i = c_i \cdot a_i \oplus c_i \cdot b_i \oplus b_i \ . \tag{5.1}$$

Thus, each output bit contributes one equation of degree 2 to the system of equations. If the conjecture by Kipnis and Shamir is correct, it would suffice to collect slightly more than $N_2 = \frac{l^2+l}{2}$ output bits in order to build a solvable system of equations of this basic type. Using the Gaussian elimination algorithm, the computational effort for such an attack would be in the order of $O({N_2}^3) = O(l^6)$, yielding a polynomial time attack on the Geffe generator.

However, using the extension technique, the number of output bits required can be reduced even further. As an example, multiplying equation (5.1) with c_i, $a_i \cdot b_i$, and $c_i \cdot a_i$ (resp.) yields the new equations

$$\begin{aligned} 0 &= c_i \cdot a_i \oplus z_i \cdot c_i \ , \\ 0 &= a_i \cdot b_i \oplus z_i \cdot a_i \cdot b_i \ , \\ 0 &= c_i \cdot a_i \oplus z_i \cdot c_i \cdot a_i \ , \end{aligned}$$

all of which also have degree 2 once z_i is replaced by a bit value from the output stream. Now, each output bit contributes 4 equations to the system of equations, reducing the overall number of known output bits needed to perform the attack.

Concluding remarks: The attack presented above is the most simple form of algebraic attack. It is sometimes denoted as *XL attack*, from its components eXtension and Linearisation. Note, however, that more efficient variations exist, although it seems hard to find precise estimates for the required running time or output bits.

All aspects of algebraic attacks are currently a very active field of research. Research topics include how to find nonlinear equations of low degree, how to solve these equations as efficiently as possible, how to estimate the resources required by the algorithms, how to apply algebraic attacks against specific ciphers, and others.

5.4 BDD attacks

General idea: In [71], an attack technique based on binary decision diagrams (BDDs) was presented. A BDD is a graph-based representation of a Boolean function, as described, e.g., in [119]. In particular, the attack uses so-called free BDDs (or FBDDs), which have the special property that representations of two functions F_1 and F_2 are efficiently merged to $F_1 \wedge F_2$.[1] Another important property is that given an FBDD representation of a function, all satisfying assignments to its input variables can be efficiently constructed.

In a BDD attack, the goal is to reconstruct the output of the LFSRs, denoted as the *internal bitstream* y. This bitstream has to meet two kinds of conditions:

1. It has to be a correct output of the LFSRs, i.e., all linear dependencies between the internal bits must be met.

2. It must lead to the correct output stream, given the nonlinearity mechanism (clock control, nonlinear filtering etc.) for the generator considered.

Note that each linear dependency and each output bit defines one Boolean function F such that $F(y) = 1$ iff y meets the condition. In a BDD attack, each condition is represented by an FBDD. These FBDDs are subsequently merged to represent one single function whose variables are the bits of the internal bitstream and which outputs 1 for all candidate bitstreams that are consistent with the linear recurrences and the output bits. If the number of output bits is large enough, the number of satisfying assignments to this FBDD gets small, and so does its size. Now the satisfying candidates can be efficiently constructed from the FBDD, and given these bits of the internal bitstream, the attacker also obtains the initial states of the LFSRs, as discussed in section 3.2.

Performance issues: The performance of the BDD attack depends on the nonlinearity mechanism used. Given the internal bitstream, the *information rate* α is the information (in bit) that one bit of a randomly chosen output stream gives about the internal bitstream. Then the running time and memory requirements of a BDD attack can be estimated to be in $O(2^{\frac{1-\alpha}{1+\alpha}l})$. For the sample generators, this implies the following:

- *Geffe generator:* The information rate is $\alpha = 1/3$. Thus, running time and memory required are in $O\left(2^{\frac{2/3}{4/3} \cdot l}\right) = O(2^{0.5 \cdot l})$.

- $\{1, 2\}$*-clocked generator:* The information rate is $\alpha = 2/5$. Consequently, the resources required can be estimated to be in $O\left(2^{\frac{3/5}{7/5} \cdot l}\right) = O(2^{0.43 \cdot l})$.

[1] Note, however, that as with all representation of a Boolean function in n variables, an FBDD can have a size of up to 2^n. Merging is only efficient in the size of the representations for F_1 and F_2.

Note that the resource estimate is asymptotical and that only an upper bound is provided. However, experimental results by Schleer [107] and Stegemann [115] indicate that both running time and memory required for an implementation of the attack are indeed very close to the theoretical estimate.

5.5 Time-Memory-Data tradeoffs

By definition in section 2.3, the attacker is not able to conduct a brute force search over the key space. He may, however, attempt a search over a small part of the key space, hoping that the produced output string is observed in the known output stream. As long as the available output stream is small (say, little more than l bit), his probability of success is negligibly small. The picture changes, though, if a sufficiently long string of output bits is available. In this case, the attacker can make use of a time-memory-data tradeoff attack, with the probability of finding a pre-computed value amongst the observed output stream being close to one.

The birthday problem: Such collision-based attack techniques are surprisingly successful not only in PRG cryptanalysis, but also in attacking other cryptographic primitives like block ciphers or hash functions.[2] The mathematical reason for this success lies in a number of results from probability theory that have been termed *birthday problem*. The cryptographically most relevant instances of the birthday problem are defined as follows:

1. *Collision within one set:* Let an urn contain M balls numbered 1 to M. One ball is drawn at a time, with replacement, the number is written down. What is the expected number N of draws until the first collision occurs, i.e., the same ball is drawn for the second time?

2. *Collisions between two sets:* Let an urn contain M balls numbered 1 to M. First, a set of N_1 balls is drawn without replacement, the numbers are written down. Then the balls are placed back into the urn. Now balls are drawn, with replacement, and the number is compared to the numbers of the list. What is the expected number N_2 of draws before a collision with the list occurs?

Since exact collision probabilities for the birthday problem are difficult to handle in practice, asymptotic estimates are being used for cryptographic purposes. In the case of a collision within one set, the expected number of necessary draws can be approximated as $N \approx \sqrt{M}$ for large values of M. For collisions between two sets, the estimate $N_2 \approx M/N_1$ is used.

Basic Time-Memory tradeoff attack: In the *precomputation phase*, the attacker selects N_1 different keys at random, and for each key computes the first

[2]For definitions, see, e.g., [86].

l output bits produced by the generator. The resulting tupel of output string and key is saved in a hash table, indexed by the output string. During *realtime phase*, the attacker is given $N_2 + l - 1$ bits of generator output. From this, he generates N_2 overlapping output strings of length l. Each string is looked up in the hash table, and if a match is found, the corresponding key can be read directly from the table. Note that if $N_1 \cdot N_2 > 2^l$, the attacker is likely to succeed using this method.

The computational effort during precomputation is determined by running the generator N_1 times and storing $2l \cdot N_1$ bits in a hash table. In the realtime phase, an expected N_2 table lookups generate the main bulk of work. Thus, the overall effort is roughly $N_1 + N_2$ computational steps. In the best case, $N_1 \approx N_2 \approx 2^{l/2}$, allowing the attacker to break the system in roughly $2^{l/2+1}$ computational steps, using $2l \cdot 2^{l/2}$ bits of memory.

Improvements: In the basic time-memory tradeoff attack, both time and memory requirements for the pre-computation phase are in the order of N_2. In practice, however, computation time is considerably cheaper than memory. This problem is solved by the *time-memory-data* tradeoff [11], which allows for a more sophisticated choice of the attack parameters. Using this technique, the parameters T (realtime computation time), P (pre-processing computation time), D (number of known output bits), and M (number of memory bits available) can be chosen in any way, as long as the conditions $P = 2^l/D$, $D^2 \leq T \leq 2^l$, and $TM^2D^2 = 2^{2l}$ are satisfied.

As another problem arising in practice, realtime computation time is determined by the number of table lookups. Since the table of samples is very large, it must be stored on hard disk, and disk access is slower than RAM access by a factor of about 4 million (or 2^{22}). While in theory, this constant factor is often neglected, it slows down a practical attack considerably. A solution to this problem is *sampling*, where only states that generate certain output patterns are stored on disk [11]. As a consequence, only those output strings that display this pattern have to be looked up, keeping the overall computational time constant, but reducing the number of disk accesses.

Chapter 6

Correlation Attacks

6.1 Basic correlation attack

Consider the Geffe generator or any other PRG based on a number of LFSRs and a nonlinear combining function g. While g must be balanced for all generators that pass the statistical tests given in section 4.2, there is a more subtle danger. For some choices of g, a correlation between an input bit a and the corresponding output bit z can be observed. As an example, consider the combining function

$$g(a, b, c) = (c \wedge a) \vee (\overline{c} \wedge b)$$

of the Geffe generator. While the output is balanced, the probability that $z = g(a, b, c) = a$ is $3/4$.

An analogy to coding theory: For any combination generator and any input bit a to the combining function g, the relation between a and output bit z can be modelled in a coding theoretic setting as a noisy channel,[1] as shown in figure 6.1. Each output bit z can be seen as a noisy version of the input bit a,

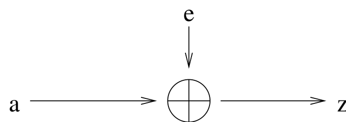


Figure 6.1: A noisy channel

i.e., z can be modelled as $z = a \oplus e$ for some noise bit e with $\Pr(e = 1) =: p_e$. Let $z_0, z_1, \ldots, z_{n-1}$ $(n > l)$ be the known output bits. It is known that the corresponding vector $(a_0, a_1, \ldots, a_{n-1})$ was generated by an LFSR of length l_A.

[1] For an introduction to the theory of linear codes, see any textbook on coding theory, e.g., [96],[117].

Thus, $(a_0, a_1, \ldots, a_{n-1})$ can be considered as a codeword in a linear code, with l_A information bits and $n - l_A$ checking bits. The problem of reconstructing the contents of register A is equivalent to finding the codeword $(a_0, a_1, \ldots, a_{n-1})$ with the least Hamming distance to the known output word $(z_0, z_1, \ldots, z_{n-1})$.

However, the general problem of reconstructing the nearest codeword in an arbitrary linear code is NP-hard [7]. In coding theory, this problem was solved by deliberately choosing linear codes in such a way that decoding is easy. A surprising consequence was that when the duality of coding theory and cryptanalysis was discovered by Siegenthaler in 1984 [111, 112], no generic algorithms for the decoding of arbitrary linear codes were known. Ever since, cryptographers have developed algorithms for the decoding problem, enabling increasingly powerful correlation attacks.

Siegenthaler's attack: The first algorithm for correlation attacks was proposed by Siegenthaler [112]. Consider a combining function g and input variable a such that $p_e := \Pr(a \neq z) < 1/2$. Given output bits $z_0, \ldots, z_{n-1}$, the attacker proceeds as follows. He guesses the complete contents of register A and generates n bits of internal bitstream $a_0, \ldots, a_{n-1}$. Then, he computes the Hamming distance

$$D_a = \sum_{i=0}^{n-1} (a_i \oplus z_i) \ ,$$

where $\oplus$ denotes bitwise addition over GF(2), while the overall sum is computed over the integers.

Note that the distribution of D_a differs, depending on whether the guess for A is correct or not. If the guess was right, D_a is binomially distributed with expected value $\mu = n \cdot p_e$ and variance $\sigma^2 = n \cdot p_e \cdot (1 - p_e)$. If it was wrong, however, the vector $(a_0, \ldots, a_{n-1})$ behaves like a random n-bit string, leading to $\mu = n/2$ and $\sigma^2 = n/4$.

These differing distributions yield a statistical test on our guess for A. Depending on the values for n and p_e, the attacker will set a threshold D' such that a guess for A is accepted as a partial key candidate if $D_a > D'$. Note that the distinguishing power of this test increases with growing n and $|p_e - 1/2|$. In the best case, exactly one candidate guess for A will be derived, immediately yielding the correct contents of register A. Otherwise, several candidate guesses remain, making additional tests necessary in order to identify the correct one. Nonetheless, if n and $|p_e - 1/2|$ are large enough, the number of steps required to retrieve the contents of register A do not significantly differ from 2^{l_A}.

Observe that Siegenthaler's technique is a variant of a guess-and-verify attack, as described in section 5.2. However, in the case of the Geffe generator, the overall running time of a correlation attack is only $2^{l_A} + 2^{l_B} + 2^{l_C}$ (as opposed to $2^{l_A + l_B} + 2^{l_C}$ steps that are required for the simple attack presented in section 5.2). This advantage is obtained at the expense of a small probability of error which can not occur with a guess-and-verify attack.

Extensions and limitations: The above attack technique can be extended to correlations between the output z and linear combinations of input bits $x^{(i)}$. If the generator consists of k LFSRs $X^{(1)}, \ldots, X^{(k)}$ and there exists an index set $I \subset [k]$ such that

$$\Pr\left(\bigoplus_{i \in I} x^{(i)} \neq z\right) < \frac{1}{2}\ , \tag{6.1}$$

then the attacker can guess the initial states of all registers $X^{(i)}$ with $i \in I$. Next, he generates the first n bits generated by each such register and calculates $x_j = \bigoplus_{i \in I} x_j^{(i)}$ for $j = 0, \ldots, n-1$. Computing the Hamming distance between $(x_0, \ldots, x_{n-1})$ and $(z_0, \ldots, z_{n-1})$, the attack proceeds as above. Note, however, that the computational effort for this phase has gone up to 2^λ steps, where $\lambda = \sum_{i \in I} l_{X^{(i)}}$.

An obvious protection against correlation attacks that guess at most r registers ($1 \leq r < k$) is the choice of a combining function g that is correlation-immune of r-th order, i.e., no set $I \subset [k]$ with $|I| \leq r$ exists that meets condition (6.1). It is known, however, that a high correlation-immunity leads to a low linear complexity, and vice versa [104]. Thus, all practical combining generators with an acceptable linear complexity will be vulnerable against correlation attacks to some extend.

6.2 Fast correlation attacks

Underlying idea: Consider again the case of a combination generator where the output a_i of a single register A is correlated with the output z_i of the generator. The attack proposed by Siegenthaler basically requires a brute force search over all possible initial states of register A, yielding an effort of $2^{|A|}$ computational steps. In [81, 82], Meier and Staffelbach proposed to use techniques from coding theory [38] in order to speed up the reconstruction of register A.

First observe that each bit of the vector $a = (a_0, a_1, \ldots, a_{n-1})$ produced by A is part of a number of linear relations. For example, if the simple feedback recurrence $a_i = a_{i-l_A} \oplus a_{i-l_A+1}$ is used, each bit a_k is contained in the three relations

$$\begin{aligned} a_k &= a_{k-l_a} \oplus a_{k-l_a+1} \\ a_{k+l_A} &= a_k \oplus a_{k+1} \\ a_{k+l_A-1} &= a_{k-1} \oplus a_k\ . \end{aligned}$$

Additional linear relations can be constructed, for example by addition of known ones. Note that the number of such relations grows in n, but that most of them will contain a large number of different variables.

Now remember that the output vector $z = (z_0, z_i, \ldots, z_{n-1})$ can be seen as the intermediate vector $a = (a_0, a_1, \ldots, a_{n-1})$, masked by an error vector $e = (e_0, e_1, \ldots, e_{n-1})$ with $\Pr(e_i = 1) < 1/2$. The basic observation is as follows: If $e = \vec{0}$, then $z = a$ and z meets all linear relations that are fulfilled

for a. If only one bit $e_k = 1$ in e, all equations containing z_k are contradictory, while all others are satisfied. But even if the Hamming weight of e increases, some linear relations in the z_i will be fulfilled. As a rule of thumb, $z_k = a_k$ holds for a given k if the share of satisfied relations amongst those that contain z_k is large. On the other hand, we expect $z_k \neq a_k$ if the share of satisfied relations is small. This simple observation can be used for a variety of reconstruction algorithms for the inner state of A.

An exponential time algorithm: Note that in order to reconstruct a, it is sufficient to reconstruct l_A bits of a. The remaining bits can be computed using systems of linear equations. A simple algorithm proposed by Meier and Staffelbach [81, 82] proceeds as follows:

1. Construct a reference set of linear relations in the z_i of equal Hamming weight.
2. For each z_i, $i = 0, \ldots, n-1$, compute the probability p^* that this bit is correct, given the number of linear relations it satisfies.
3. Choose l_A bits for a reference guess $\hat{a}$ by picking those z_i with the highest values p^*.
4. Find the correct guess by modifying $\hat{a}$ by $1, 2, \ldots$ bit and constructing the full vector a. Compute the Hamming distance between a and z. If this distance is close to the expected value, output a and stop.

The average running time of this algorithm is determined by step 4, which takes about

$$N_d = \sum_{i=0}^{d} \binom{l_A}{d}$$

trials, with d being the expected number of wrong digits in the reference guess $\hat{a}$. Since this value is clearly smaller than l_A, reconstructing register A takes $2^{c \cdot l_A}$ steps with $c < 1$.

A polynomial time algorithm: A number of improvements over the above algorithm are possible. In particular, note that when correcting the guess $\hat{a}$ in step 4, all bits are treated equal. However, it may be assumed that those bits that satisfy a large number of equations are more likely to be correct than those that satisfy less equations. Thus, a variant algorithm also proposed by Meier and Staffelbach [81, 82] proceeds as follows:

1. Construct a reference set of linear relations in the z_i of equal Hamming weight.
2. For each z_i, $i = 0, \ldots, n-1$, compute the probability p^* that this bit is correct, given the number of linear relations it satisfies.

3. Negate those bits z_i whose probability p^* is under a certain threshold. If the resulting vector z does not satisfy all relations, go back to step 2.

Note that this description is a simplified version of the full algorithm. Nonetheless, in all cases the running time for step 1 is linear in the length of the registers. Step 2 and 3 are even independent of the register length, instead, the running time is determined by the Hamming weight of the linear relations used, by the error probability $\Pr(e_i = 1)$ and by the number n of output bits available. While no closed mathematical expression for the running time could be found, it was observed that for relations of small weight (up to 8), the attack was extremely fast in practice. As a consequence, the use of LFSRs with a small number of feedback taps is strongly discouraged.

Improvements: Following the publication of [82], a number of improvements have been proposed. These can be subdivided into two categories:

- In step 1 of the above algorithm, linear relations of low degree have to be found. The efficiency of steps 2 and 3 can be increased if more care is spent on this preprocessing step. Proposals on how to find more or better linear relations were given, e.g., by Mihaljević and Golić [89], Chepyzhov and Smeets [18], and Penzhorn [95].

- In addition, the iterative decoding procedure in step 2 and 3 was improved by several proposals, such as the algorithms given by Zeng et al. [124, 126], Mihaljević and Golić [89], Chepyzhov and Smeets [18] or Živković [135]. As opposed to the original algorithm, many of these proposals also contain a proof of their convergence.

However, all of these proposals are efficient only if the feedback vectors of the LFSRs under consideration have low weight. This limitation was done away with by a set of completely different algorithms to be discovered in subsequent years. Johansson and Jönsson use convolutional codes [62, 64], turbo codes [61] or algorithms from learning theory [63] in order to reconstruct the inner state. Canteaut and Trabbia [16] proposed an algorithm to construct linear relations of low weight for arbitrary feedback vectors. Chepyzhov, Johansson and Smeets [17] approximate the LFSR output by a linear code of smaller dimension, but with higher error probability. A similar approach is chosen by Filiol [34], who proposes a d-decimating attack, considering only every d-th output bit of the LFSR.

Depending on the combination generator considered, the above attack techniques can be of varying efficiency. For the majority of generators, however, the most efficient algorithm to date is a combination of several of the above concepts, as proposed by Mihaljević, Fossorier and Imai [87, 88] and improved by Chose, Joux and Mitton [19].

6.3 Correlation attacks and memory

Correlation immunity: The efficiency of correlation attacks makes it necessary to harden combination generators against such attacks. The most obvious solution is to choose the combining function g in such a way that no correlations between the output and a linear combination of a small number of internal bits exist. More formally, a function $g : \{0,1\}^k \to \{0,1\}$ with input vector $x = (x^{(1)}, \ldots, x^{(k)})$ is said to be *correlation immune of k'-th order* if no linear combination L of up to $k' < k$ variables exists such that $\Pr(L(x) = g(x)) \neq 1/2$. The following tradeoffs, however, make it difficult to strengthen the generator in this way:

- It was shown by Siegenthaler, Xiao and Massey in [111, 122] that an increase in correlation immunity leads to a decrease in linear complexity, and vice versa. Thus, a highly correlation immune combination generator can be attacked using the Berlekamp-Massey-algorithm (see section 4.3).

- Let $\{L_i \mid 1 \leq i \leq 2^k\}$ be the set of linear functions in up to k variables. The *correlation coefficient* between g and L_i is defined as $c_i = 2 \cdot p_i - 1$, with $p_i = \Pr(L_i(x) = g(x))$. It was proven by Meier and Staffelbach [83] that

$$\sum_{i=1}^{2^k} {c_i}^2 = 1 \ . \tag{6.2}$$

  This means that if g has high correlation immunity (i.e., g is not correlated to any linear function in few variables), it is at the same time strongly correlated to linear functions with a higher number of variables. Thus, by choosing the optimal algorithm, a correlation attack is always possible.

Improved correlation immunity from nonlinear memory: In order to destroy the dependency between correlation immunity and linear complexity, Rueppel [101] introduced the generator with (nonlinear) memory. As described in section 3.3, the memory of such a generator consists of two parts: While the majority is made up of LFSRs, some bits are updated by a nonlinear function f_2. It was shown that for a good choice of f_2, such a function can achieve maximum correlation immunity while at the same time having maximum linear complexity.

However, it was proven by Meier, Staffelbach, and Golić in [84, 43, 45] that for such a generator, too, a tradeoff similar to (6.2) can be found. This time, however, several consecutive input bits from each register have to be considered, increasing the number of variables in the linear approximation function L. As a consequence, correlation attacks against combiners with memory are indeed less efficient, but not entirely impossible as was hoped for originally.

6.4 Correlation attacks and clock control

A new notion of correlation: Instead of using nonlinear memory, some LFSR-based generators use nonlinear clocking, i.e., some or all LFSRs are irregularly clocked, depending on the internal state of the generator. Note that this way, the attacker cannot see which internal bit x_i contributes to which output bit z_j. Thus, measuring the Hamming distance between $(x_1, \ldots, x_n)$ and $(z_1, \ldots, z_n)$ becomes meaningless, and correlation attacks in the above sense are no longer applicable.

However, other measures of correlation can be used. Golić and Mihaljević [50, 51] proposed to replace the Hamming distance by the so-called *Levenshtein distance*. This distance measures the minimum number of elementary operations (insertion, deletion, and substitution) required to transform one sequence into a prefix of the other. Given such a notion of distance, the standard correlation attack as defined by Siegenthaler can be deployed.

Correlation attacks: Depending on the cipher design, some edit operations may not be allowed. Thus, it may be necessary to define a so-called *Constrained Levenshtein Distance* (CLD). Note, for example, that the Hamming distance is a CLD where only substitutions are allowed. For the $\{1, 2\}$-clocked generator, only deletions are applicable with the additional constraint that no two consecutive internal bits must be deleted. In any case, an efficient dynamic programming algorithm for the computation of the CLD was given in [51]. Given a target sequence of length n, the algorithm computes the CLD in the order of $O(n^2)$ computational steps.

A number of modifications of this attack have been proposed against specific generators [134, 54, 53, 49], but the general method remains the same. In [44], Golić proposes an algorithm similar to the fast correlation attack by Meier and Staffelbach. However, step 1 of the algorithm (finding suitable linear relations) proved to be difficult, except for very special generators. Thus, a full algorithmic specification of a fast correlation attack on general irregularly clocked generators remains an unsolved research problem down to the present day.

Part III

Backtracking Attacks

Chapter 7

The Dynamic Linear Consistency Test

7.1 Generators under consideration

The linear consistency test as introduced in [125] targets pseudorandom generators (PRGs) of the type defined in chapter 3. Such generators consist of two components:

1. A *linear unit* that transforms the key bits into a sequence of intermediate bits in a linear way. In most cases, the linear unit is an LFSR.
2. A *nonlinear unit* which transforms the intermediate bits into an output stream in some nonlinear way, e.g., using nonlinear combining or clock-control functions.

Thus, each output bit of such a PRG can be written as a unique, nonlinear combination of key bits, since a) each output bit can be written as a nonlinear combination of the intermediate bits and b) each intermediate bit can be written as a linear combination of the key bits. Note, however, that in the case of clock-controlled PRGs, the size of these nonlinear representations may be too large to be actually written down.

Examples:

- For the Geffe generator, each output bit z_i can be written simply as
  - $z_0 = c_0a_0 \oplus c_0b_0 \oplus b_0$,
  - $z_1 = c_1a_1 \oplus c_1b_1 \oplus b_1$,
  - $z_2 = c_2a_2 \oplus c_2b_2 \oplus b_2$,
  - $\ldots$

- For the $\{1,2\}$-clocked generator, the first output bits z_i have algebraic representations as

  - $z_0 = a_1 \oplus c_0a_1 \oplus c_0a_2$,
  - $z_1 = a_2 \oplus c_0a_2 \oplus c_1a_2 \oplus c_0c_1a_2 \oplus c_0a_3 \oplus c_1a_3 \oplus c_0c_1a_4$,
  - $z_2 = a_3 \oplus c_0a_3 \oplus c_1a_3 \oplus c_2a_3 \oplus c_0c_1a_3 \oplus c_0c_2a_3 \oplus c_1c_2a_3 \oplus c_0c_1c_2a_3 \oplus c_0a_4 \oplus c_1a_4 \oplus c_2a_4 \oplus c_0c_1c_2a_4 \oplus c_0c_1a_5 \oplus c_0c_2a_5 \oplus c_1c_2a_5 \oplus c_0c_1c_2a_5 \oplus c_0c_1c_2a_6$,
  - ...

  It can be seen that the size of the algebraic representation is growing rapidly with increasing index i.

If for any register X, a variable x_i with $i \geq l_X$ is used in an equation (i.e., x_i is no longer a key bit), x_i can be replaced by a linear combination of key bits, making use of the feedback recurrence for register X. Thus, the number of variables in the equations is always upper bounded by the total key length l, while the number of equations is limited by the number of known output bits.

7.2 The linear consistency test revisited

As noted in section 5.2, the linear consistency test guesses part κ of the key and verifies the correctness of the guess using a system of linear equations, as described in figure 7.1. Keeping in mind the nonlinear representation of the output stream, κ should be chosen such that after an assignment to κ is known, the nonlinear equations become linear. In the above examples, it is easily seen that for both the Geffe generator and the $\{1,2\}$-clocked generator, the system of equations becomes linear if all bits generated by register C are known. Thus, it suffices to guess all possible assignments for register C in order to reconstruct the full key.

LINEAR CONSISTENCY TEST:
1. Choose a particularly useful subkey κ of length $\lambda < l$.
2. For all assignments κ' for the subkey κ:
3.   Derive the system of linear equations implied by κ'.
4.   If the system of equations is consistent:
5.   Save all solutions to the system as key candidates.
6.   Else:
7.   Discard κ'.
8. Test all key candidates by running the generator.

Figure 7.1: Linear consistency test

Running time: The running time of the LCT attack is determined by the assignment loop (steps 2 through 7) and the final testing phase (step 8). The assignment loop is iterated 2^λ times, with each loop requiring the solution of a system of equations with $l - \lambda$ variables. It was shown in [125] that if the number of equations is slightly larger than the number of variables, the number of consistent systems of equations is close to 1, and that the number of solutions to such a system is close to 1 as well. Thus, the LCT requires $O(2^\lambda \cdot (l-\lambda)^3)$ operations for steps 2 through 7 and $O(1)$ steps for step 8.

7.3 A dynamic extension

It was first pointed out by Golić in [46] that the LCT method can be applied in a more dynamic fashion. When attacking the pseudorandom generator A5/1 used in the GSM mobile phone standard, the necessary bits were guessed only one at a time. However, no generalisation of this concept to other generators was considered. This generalisation is a major contribution of this thesis and will be discussed in the subsequent chapters.

As shown in section 7.2, the running time of the LCT is mainly determined by the number of bits that have to be guessed in order to transform the nonlinear system of equations into a linear one. Thus, it is important for the attacker to keep the number of guesses as small as possible. One way to achieve this goal is to guess key bits one by one instead of guessing a subkey κ all at once. In this way, the system of linear equations can be built successively, and if an early contradiction occurs, a whole set of key candidates can be discarded at once. On the other hand, as soon as the system of equations reaches full rank, it can immediately be solved in order to find the remaining key bits.

The backtracking algorithm: More formally, this procedure can be implemented using a backtracking algorithm. A first version of such an algorithm is described in figure 7.2. Let $\pi : \{0, \ldots, l-1\} \to \{0, \ldots, l-1\}$ be a suitable permutation over the key bits, i.e., the attacker guesses the key bits in the order $k_{\pi(0)}, k_{\pi(1)}, \ldots$. The algorithm is started with an empty set M of linear equations, an empty assignment $k = (*, *, \ldots, *)$ to the key bits, and with index $d = 0$. It outputs all key candidates that are consistent with the nonlinear equations, if any exist. Note that the algorithm always terminates, since for $d = l$, all key bits have been guessed, and since each guess can be modelled as a linear equation, the condition in line 5 is always met. In most cases, however, the algorithm will terminate much earlier.

The behaviour of backtracking algorithms is often represented by a search tree, where each call of BACKTRACK(M, k, d) is represented by a node, and the relationship between calling node and called node is modelled by a directed edge. When actually drawing such a tree, a leaf that contains a contradiction will be marked by an underscore, while a leaf containing a system of equations of full rank will be marked by a grey box.

BACKTRACK(M, k, d)
1. For all new linear equations L:
2. Add L to M.
3. If a contradiction occurs:
4. Return "Contradiction"
5. If $\texttt{rank}(M) = l$:
6. Solve the system of equations.
7. Output solution as key candidate.
8. Return "Candidate complete"
9. For $b = 0, 1$:
10. $k_{\pi(d)} \leftarrow b$
11. Backtrack$(M, k, d + 1)$

Figure 7.2: The backtracking algorithm

A toy example: Consider a Geffe generator with $l_A = l_B = 2$ and $l_C = 6$. The feedback recurrences are $a_i = a_{i-1} \oplus a_{i-2}$, $b_i = b_{i-1} \oplus b_{i-2}$, and $c_i = c_{i-4} \oplus c_{i-6}$. Assuming that output bits $(z_0, \ldots, z_5) = (1, 0, 0, 1, 0, 1)$ are available, the following equations can be constructed:

$$\begin{aligned}
1 &= z_0 = c_0(a_0 \oplus b_0) \oplus b_0 \\
0 &= z_1 = c_1(a_1 \oplus b_1) \oplus b_1 \\
0 &= z_2 = c_2(a_0 \oplus a_1 \oplus b_0 \oplus b_1) \oplus b_0 \oplus b_1 \\
1 &= z_3 = c_3(a_0 \oplus b_0) \oplus b_0 \\
0 &= z_4 = c_4(a_1 \oplus b_1) \oplus b_1 \\
1 &= z_5 = c_5(a_0 \oplus a_1 \oplus b_0 \oplus b_1) \oplus b_0 \oplus b_1
\end{aligned}$$

We guess the key bits in the order $(c_0, \ldots, c_5, a_0, a_1, b_0, b_1)$. Figure 7.3 shows levels $d = 0, 1, 2, 3$ of the resulting search tree and the systems of equations available at that stage. Note that at level $d = 3$, two nodes contain a contradictory system of equations, meaning that the partial key guesses leading to these nodes, namely $(0, 0, 0, *, \ldots, *)$ and $(1, 1, 1, *, \ldots, *)$, cannot be correct.

7.4 Computational resources

Resources per function call: The *memory* that is required by each call to the function BACKTRACK(M, k, d) is determined by the parameters M, k and d. Note that $0 \leq d \leq l$, meaning that $\lceil \log(l + 1) \rceil$ bits suffice to store d. The partial key k requires l bits, and the system of equations needs at most $l \cdot (l + 1)$ bits if only linearly independent equations are stored. Thus, the total number of memory bits required for one call to the function BACKTRACK(M, k, d) is in $O(l^2)$.

If linear equations as required by step 1 of the function BACKTRACK(M, k, d) can be constructed efficiently, the *running time* of one function call is determined

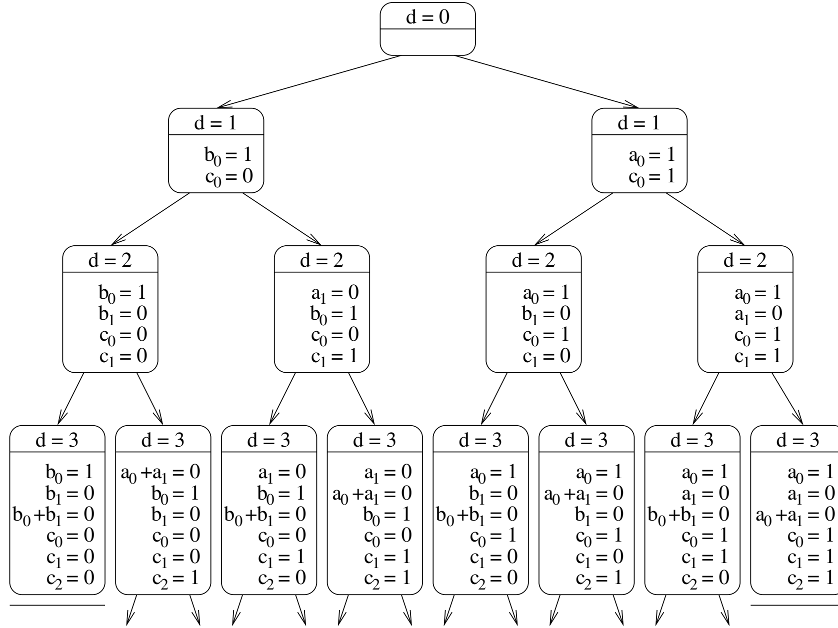


Figure 7.3: The nodes of a dynamic LCT search tree

by elementary operations on systems of binary equations (in steps 2, 3, 5, and 6). Since all of these operations can be performed in $O(l^3)$ steps (see, e.g., [23], chapter 28), the overall running time of one call to the function BACKTRACK(M, k, d) is also in the order of $O(l^3)$ computational steps.

Total resources: In order to determine the overall *memory* requirements, observe that the maximum depth of the search tree is l. This means that in the worst case, $l + 1$ recursive calls to the function BACKTRACK(M, k, d) are active at once. Even if for each call a separate copy of all parameters is used, the number of memory bits required is in $O(l^3)$.

On the other hand, the *running time* of the algorithm is determined directly by the total number of calls to function BACKTRACK(M, k, d), i.e., the size of the search tree. In the worst case (all nonlinear equations have degree l), this search tree is a complete binary tree of depth $d = l - 1$ with $2^l - 1$ nodes. On the other hand, if contradictions occur early during tree traversal, the tree will be much smaller, albeit in most cases still exponential in l. Thus, the overall running time of the dynamic LCT algorithm is in $O(l^3 \cdot 2^{cl})$, with $0 < c < 1$ being a constant that determines the efficiency of the attack.

The size of the search tree is reduced whenever a branch from the complete binary search tree is cut away, either because a contradiction has occured or because the internal system of equations has full rank. Note that contradictions

can only occur if a new equation is linearly dependent on the previous ones, while the rank of the system of equations is only increased with linearly independent equations. In practice, however, it is hard to predict whether new equations at a given depth will be linearly dependent on the previous ones.

The reason for this is that all changes to the generator, the feedback recurrences, or the variable ordering π change the form of the linear equations and thus the dependency structure. As a consequence, it seems to be impossible to find a mathematical description of the tree size or the constant c for the dynamic LCT attack in general. The best that can be done is to give estimates for certain classes of generators. Such estimates will be presented in the remaining chapters of part III. In particular, a case study against the self-shrinking generator will be discussed in chapter 8.

Handling of key candidates: So far, only the effort for traversing the search tree was considered, ignoring the handling of the key candidates produced in step 8. Note, however, that such candidates can be processed immediately by calling a suitable test procedure that runs the generator with this key, checking the correctness of the output. Thus, no additional *memory* is required.

Nonetheless, the number of key candidates can be exponentially large in l, depending on the number and the degree of the nonlinear equations available. Note that if n output bits (and associated equations) are available, the algorithm is expected to produce 1 correct and $(2^l - 1)/2^n \approx 2^{l-n}$ false key candidates. On the other hand, each leaf of the search tree can produce at most one key candidate, implying that the number of key candidates is in $O(2^{cl})$. Thus, the additional *running time* for testing the key candidates is well contained in the asymptotic running time for determining those candidates, and will be neglected from now on.

The toy example, completed: In figure 7.4, the structure of the complete search tree for the toy example is given. The following observations can be made:

- The search tree contains an overall of 83 nodes (or calls to the recursive function). This is better than the brute force effort of testing $2^{10} = 1024$ inner states, but worse than the $2^6 = 64$ tests required for a simple LCT algorithm. This is surprising; after all, the dynamic approach is meant to improve the running time of the basic LCT. Thus, it is necessary to find out in which cases the dynamic version is superior to the basic one, and vice versa.

- There exist 16 leaves in the search tree that contain a system of equations with full rank, meaning that an overall of 16 key candidates exist. This is very close to what is expected - on the average, there should be 1 correct and $1023/64 \approx 15.98$ incorrect candidates.

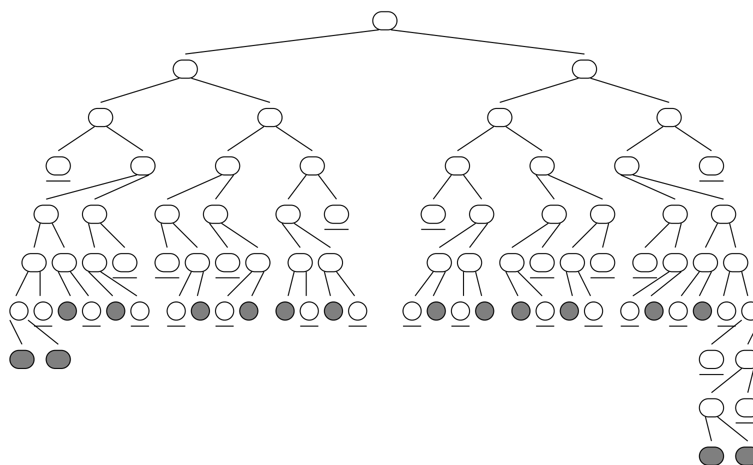


Figure 7.4: The structure of a dynamic LCT search tree

7.5 Further improvements

The role of the variable ordering: An important role for the efficiency of the attack plays the order π in which the assignments to the variables are guessed. Informally, it is important for the attacker to choose π such that he obtains as many linear equations as early as possible. In this way, he increases his chances of obtaining an early contradiction or key candidate, cutting the tree short at low depth. As an example, if the attack on the toy example is conducted using the variable ordering $\pi = (a_0, b_0, a_1, b_1, c_0, c_3, c_1, c_4, c_2, c_5)$, the size of the search tree is reduced to 57 nodes. Considering that a search tree with 16 grey leaves (i.e., key candidates) consists of at least 31 nodes, this value is much closer to the optimum than the 83 leaves of the search tree in figure 7.4.

Adaptive guessing: One step towards a smaller search tree is to abandon the statical guessing order π in favour of an adaptive one. At each level, the attacker chooses the next variable to be guessed in such a way that he obtains as many linear equations as possible. In the above example, if $(a_0, b_0) \in \{(0,0),(1,1)\}$, the attacker should use the variable ordering $\pi_1 = (a_0, b_0, a_1, b_1, c_0, c_3, c_1, c_4, c_2, c_5)$, otherwise, he should use $\pi_2 = (a_0, b_0, a_1, b_1, c_1, c_4, c_0, c_3, c_2, c_5)$. This way, the tree size reduces to a mere 49 nodes. More formally, the next variable to be guessed should depend on the values of those previously guessed. This can be modelled by a variable ordering graph (with each node being labeled by a variable name and each path by an assignment to this variable), as shown, e.g., in chapter 6 of [119] for use with BDDs.

Equation guessing: Another option for decreasing the tree size is to guess linear equations instead of single bits. Remember that for the dynamic LCT

as presented in section 7.3, not only the output equations, but also the guessed bits form equations that are added to the system. Sometimes, however, not the individual bits are of interest, but their combination. As an illustration, consider the toy example for the Geffe generator again. If the values of $a_0 \oplus b_0$ and $a_1 \oplus b_1$ are known, all equations for $(z_0, \ldots, z_5)$ become linear and can be added to the system of equations. Thus, after just two guesses, the system of equations contains up to 8 equations. If, in addition, the remaining bits are guessed in an adaptive way, the size of the search tree can be reduced to as little as 35 nodes, which is almost optimal. In chapter 9, a special case of equation guessing denoted as *clock control guessing* will be discussed in more detail.

Chapter 8

Dynamic LCT and the Self-Shrinking Generator: A Case Study

8.1 The self-shrinking generator

8.1.1 Description

The **self-shrinking generator** is a modified version of the shrinking generator [22] and was first presented by Meier and Staffelbach in [85]. It requires only one LFSR A of length $l_A = l$ and a clock-control unit. The LFSR generates an m-sequence $(a_0, a_1, \ldots)$ in the usual way. The output function requires two consecutive bits (a_{2i}, a_{2i+1}) as input and outputs a_{2i+1} iff $a_{2i} = 1$. The basic concept of the generator is displayed in figure 8.1.

Period and Linear Complexity: The period ρ of an output sequence generated by a self-shrinking generator was proven to be $2^{\lfloor l/2 \rfloor} \le \rho \le 2^{l-1}$ in [85].

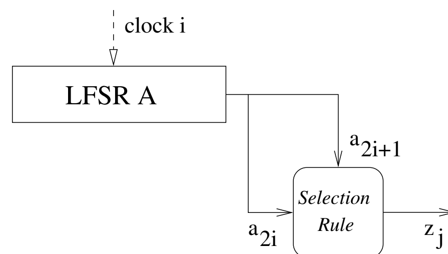


Figure 8.1: The Self-Shrinking Generator

Experiments indicate that the period always takes the maximum possible value for $l > 3$.

It was also shown that the linear complexity LC(z) is greater than $\rho/2$ for all sequences generated by a self-shrinking generator. On the other hand, LC(z) was proven in [13] to be at most $2^{l-1} - (l-2)$. If $\rho = 2^{l-1}$, it follows that LC(z) $\in \Theta(2^{l-1})$. Thus, for realistic generator sizes of $l > 100$, the Berlekamp-Massey attack (section 4.3) is computationally infeasible.

8.1.2 Previous work on cryptanalysis

Attacks using short output sequences: Even if the feedback logic of the LFSR is not known, there is a simple way of reducing the key space [85]. Consider the first two bits (a_0, a_1) of the LFSR (unknown) and the first bit z_0 of the output stream (known). Then there are only three out of four possible combinations (a_0, a_1) that are consistent with the output stream, since $(a_0, a_1) = (1, \bar{z}_0)$ is an immediate contradiction. The same rule can be applied for the next bit pair (a_2, a_3), and so on. Consequently, only

$$3^{l/2} = 2^{(log_2(3)/2) \cdot l} = 2^{0.79l}$$

possible initial values for the LFSR A are consistent with the initial bits of the output stream.

The running time required to search through the reduced key space can be further decreased on the average by considering the likelihood of the keys. Note that the following holds:

$$\begin{array}{rcl} \Pr[(a_0, a_1) = (0,0) \mid z_0] & = & 1/4 \\ \Pr[(a_0, a_1) = (0,1) \mid z_0] & = & 1/4 \\ \Pr[(a_0, a_1) = (1, z_0) \mid z_0] & = & 1/2. \end{array}$$

Thus, the entropy of the bit pair is

$$H = -(1/4)\log(1/4) - (1/4)\log(1/4) - (1/2)\log(1/2) = 3/2.$$

The total entropy of an initial state consisting of $l/2$ such pairs is $0.75l$. Thus, the effort for searching the key space is roughly $2^{0.75l}$ if the cryptanalyst starts with the most probable keys.

The most efficient attack is the BDD attack by Krause [71]. Both running time and memory requirements of this attack were asymptotically estimated to be in $O(2^{0.656l})$, using roughly $2.41l$ bit of output. Note, however, that as opposed to the above techniques, BDD attacks require significant memory. Also note that the asymptotical estimate hides significant polynomial factors from view. An implementation was given by Schleer [107], reconstructing the inner state for key sizes up to $l = 24$ bit and indicating that only for large values l, a BDD attack will be more efficient than the techniques presented in [85].

value n:	$0.25l$	$0.306l$	$0.50l$
Time:	$2^{0.75l}$	$2^{0.694l}$	$2^{0.5l}$
Bits :			
$l = 120$	$2^{8.19}$	$2^{10.17}$	$2^{65.91}$
$l = 160$	$2^{8.81}$	$2^{11.37}$	$2^{86.32}$
$l = 200$	$2^{9.30}$	$2^{13.07}$	$2^{106.64}$
$l = 240$	$2^{9.69}$	$2^{14.03}$	$2^{126.91}$
$l = 280$	$2^{10.02}$	$2^{14.94}$	$2^{147.13}$
$l = 320$	$2^{10.31}$	$2^{15.81}$	$2^{167.32}$

Table 8.1: Number N of output bits required for Mihaljević's attack

Attack using long output sequences: In [90], Mihaljević presented a faster attack that needs, however, a longer part of the output sequence. Let the length of this known part be denoted by N. Then the attacker assumes that an n-bit section of the output stream has been generated by the current inner state of the LFSR. Consequently, n out of the $l/2$ even bits of A must be equal to 1. The attacker guesses these bits and checks whether or not this guess can be correct, iterating over all n-bit sections of the output stream. It is shown that cryptanalysis is successful with high probability after 2^{l-n} steps.

Since this procedure only makes sense for $l/4 \leq n \leq l/2$, the running time can vary from $2^{0.5l}$ in the best case to $2^{0.75l}$ under less favourable circumstances. The efficiency of the attack depends mainly on the number of output bits that are available, since the value n must be chosen such that the following inequality holds:

$$N > n \cdot 2^{l/2} \cdot \binom{l/2}{n}^{-1}$$

In order to get a feeling for the number of bits required for this attack, table 8.1 gives some examples of required bitstream lengths for different register sizes l. Consider the following cases:

- In order to beat the key reconstruction algorithm by Meier and Staffelbach, $n = 0.25l$ is necessary, yielding a running time of $2^{0.75l}$ steps.

- Improving the running time to $2^{0.694l}$ (which is the performance of the algorithm to be presented in section 8.2) requires $n = 0.306l$.

- In order to achieve the best possible running time of $2^{0.5l}$ steps, it must hold that $n = 0.5l$. Note that for realistic register lengths, the sheer amount of required data (namely, $N > \frac{l}{2} \cdot 2^{l/2}$) should make such an attack a mere theoretical possibility.

8.2 Applying the dynamic LCT attack

We will now apply the adaptive guessing technique introduced in section 7.5 against the self-shrinking generator in way of a case study, using the results previously published in [132]. In this way, we illustrate some of the difficulties encountered when trying to find an estimate for the running time of a dynamic LCT attack. Nonetheless, it will be shown that this attack is more powerful against the self-shrinking generator than the techniques by Meier/Staffelbach and Mihaljević described in section 8.1.2, almost reaching the asymptotical efficiency of Krause's attack without its memory requirements.

Variable ordering: Remember that the LFSR A of the self-shrinking generator produces the internal bitstream $(a_0, a_1, \ldots)$, using all even bits as clock control bits and all odd bits as output bits. Thus, while the attacker guesses the sequence of control bits $(a_0, a_2, a_4, \ldots)$, he learns which inner sequence bits turned into output bits. As an example, if the first output bits are $(1, 1, 0)$ and if the attacker assumes that the first clock control bits have been $a_0 = 1, a_2 = 0, a_4 = 1$, and $a_6 = 1$, it follows from the output stream that $a_1 = 1, a_5 = 1$ and $a_7 = 0$. In this way, using the variable ordering $(a_0, a_2, \ldots)$ for his guesses, the attacker conducts a dynamic LCT attack.

A useful proposition: When analysing the resources required by the attack, the following property of the key (i.e., the seed of the LFSR) can be useful[1]:

Proposition 1 *For each key $k = (a_0, \ldots, a_{l-1})$ with $a_0 = 0$, there exists an equivalent key $k' = (a'_0, \ldots, a'_{l-1})$ with $a'_0 = 1$.*

Proof: Consider the sequence $(a_i)_{i \geq 0}$ generated by the inner state k. Suppose the first '1' on an even position appears in position $2s$. Then clock the register by $2s$ steps, deriving the new inner state $k' = (a_{2s}, \ldots, a_{2s+l-1})$. Obviously, both inner states yield the same output sequence, since in transforming k to k', no output is generated. □

It is thus safe to assume that $a_0 = 1$ and $a_1 = z_0$. In this way, we will reconstruct a key that is not necessarily equal to the original key, but it is equivalent in a sense that it will create the same output sequence.

From now on, the attacker has to guess the even bits of the sequence $(a_i)_{i \geq 0}$. In this way, he obtains two different types of equations as follows:

- Every guess can be represented by a linear equation $a_{2d} = b_d$. These equations will be referred to as being of *type 1*.
- If $a_{2d} = 1$, he obtains a second equation, namely $a_{2d+1} = z_j$, where $j = \sum_{c=0}^{d} a_{2c}$. These equations will be denoted as being of *type 2*.

The first nodes of the corresponding search tree are shown in figure 8.2.

[1] The same property also holds for the shrinking generator. In this context, it was discussed in [114].

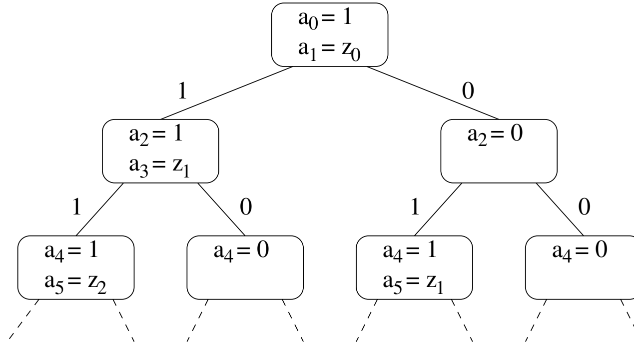


Figure 8.2: The Tree of Guesses

Structure of the tree: In the next section, the size of this search tree shall be evaluated. Note that as long as $d \leq \lfloor l/2 \rfloor - 1$, the development of the tree is straightforward. There are exactly two new equations whenever a '1' branch is followed, and exactly one new equation when following a '0' branch. All of these equations are linearly independent, since no variable a_i appears more than once. Thus, the search tree is a complete binary tree of height $\lfloor l/2 \rfloor - 1$.

Beyond that point, however, the tree becomes irregular, since the indices of the new equations (both of type 1 and 2) become larger than $l - 1$. Thus, the feedback recurrence must be used to convert the simple equations into a representation using only $a_0, \ldots, a_{l-1}$. Depending on the equations that are already known, there is an increasing probability that the new equations are linearly dependent on the earlier ones. If this leads to a contradiction, the tree is cut short at this point, reducing the tree size. The same holds if the system of equations obtains full rank, which can happen at some point between $d = \lceil l/2 \rceil - 1$ and $d = l - 1$.

8.3 Upper bounding the running time

In this section, an asymptotical upper bound on the running time of the algorithm is established. First, an upper bound C_l for the number of *consistent leaves* in the tree of guesses is given (sections 8.3.1 to 8.3.3).[2] Then, in section 8.3.4, an upper bound for the number N_l of *nodes* in the tree is derived, followed by the conclusion that the total running time of the algorithm can be upper bounded by $O(l^4 \cdot 2^{0.694l})$.

[2]A consistent leaf is a leaf that contains a linear equation system of full rank, as opposed to an inconsistent leaf which contains a contradictory equation system.

8.3.1 Well-formed vs. malformed trees

Let T_ℓ denote a tree of guesses such that ℓ linearly independent equations are still missing in the root to allow the solving of the system of equations. Note that for the search tree given in section 8.2, it holds that $\ell = l - 2$.

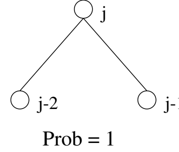
In order to formally prove the maximum number C_ℓ of consistent leaves in T_ℓ, each node is labelled by the number of linearly independent equations still needed in order to solve the system of equations. The root is thus labelled by ℓ. For technical reasons, we allow a consistent leaf of the tree to take both the labels 0 and -1, both meaning that the system is completely specified.

Assumption 1 *For the following average case analysis, assume that an equation that is linearly dependent on its predecessors will lead to a contradiction with probability 1/2.*

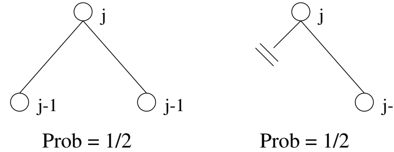
This assumption is reasonable, since the bits a_i are generated by an m-LFSR, meaning that a variable takes values 0 and 1 with (almost) equal probability.

Now consider an arbitrary node V of depth $d-1$, $d \geq 1$, and its two children, V_0 and V_1 (reached by guessing $a_{2d} = 0$ or $a_{2d} = 1$, resp.). Let V be labelled by j. The labelling of the child nodes depends on whether a_{2d} or a_{2d+1} are linearly dependent on the previous equations:

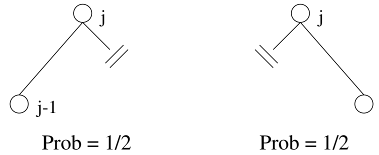
A) *Both are independent.* In this case, no contradiction occurs. The left child is labelled $j-2$, and the right child is labelled $j-1$.



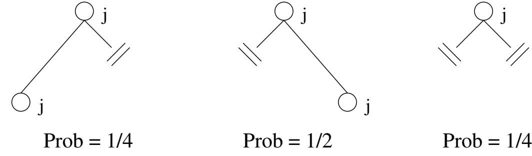
B) a_{2d} *is independent,* a_{2d+1} *is not.* Both children are labelled $j-1$. However, a contradiction occurs in V_1 with probability 1/2.



C) a_{2d} *is dependent,* a_{2d+1} *is not.* The left child is labelled $j-1$, while the right child is labelled j. However, a contradiction occurs either in V_1 or in V_0, with equal probability.



D) *Both are dependent.* In this case, both child nodes have the same label as the parent node. Due to the linear dependency of a_{2d} there occurs a contradiction in either V_1 or V_0, with equal probability. Also note that there is an additional probability of 1/2 that a_{2d+1} leads to a contradiction in V_1.



Definition 1 *A* ***well-formed tree*** T^*_ℓ *is a binary tree where only branchings of type A occur, i.e., for every node that is not a leaf, the following rule holds: If the label of the node is* j*, then the label of its left child is* $j-2$ *and the label of its right child is* $j-1$*.*
A ***malformed tree*** *is an arbitrary tree of guesses that contains at least one branching of type B, C or D.*

Essentially, the notion of a well-formed tree describes the tree of guesses under the assumption that all linear equations (of both type 1 and 2) are linearly independent. Note that such a tree is highly unlikely for large ℓ. Nonetheless, the well-formed tree plays an important role in establishing the overall number of consistent leaves for the tree of guesses.

Consider the following experiment in order to generate a tree of guesses. We start at the root (labelled ℓ) and generate the lower levels recursively with the help of an adversary G as follows:

- If $l \geq 1$, let G choose one of the roles *A*-*D*. The labels of the child nodes are determined probabilistically, as described above. For each child node, this algorithm is repeated recursively.

- If $l \leq 0$, mark the current node as leaf and backtrack.

Note that the tree generated by this experiment has the same structure as a tree generated while running the dynamic LCT attack on the self-shrinking generator. Now, it can be proven that whichever roles the adversary G chooses during the course of the experiment, on the average, a malformed tree has at most the same number of consistent leaves as a well-formed tree.

Theorem 1 *Let* C^*_ℓ *denote the number of consistent leaves of a well-formed tree* T^*_ℓ*. Let* C_ℓ *denote the maximum number of consistent leaves in a tree* T_ℓ *that was generated according to the above experiment. Then for all behaviours of* G*, it holds that* $C_\ell \leq C^*_\ell$ *on the average.*

Proof: The proof is by induction. Obviously, the inequality holds for C_{-1} and C_0, since trees T_{-1} and T_0 consist only of a root without a child. Thus, $C_{-1} = C^*_{-1} = 1$ and $C_0 = C^*_0 = 1$.
Now consider C_ℓ, $\ell \geq 1$. First note that since the theorem holds for $C_{\ell-1}$ and $C_{\ell-2}$, it follows that

$$C_{\ell-1} + C_{\ell-2} \leq C^*_{\ell-1} + C^*_{\ell-2} = C^*_\ell. \tag{8.1}$$

Also note that even in the worst possible branching case, it follows that

$$C_\ell \leq 2 \cdot C_{\ell-1}. \tag{8.2}$$

for all ℓ. Using these two facts, an upper bound for C_ℓ can be proven by distinguishing the following cases (identical to the behaviours of G in the above experiment):

A) Let the tree T^A_ℓ be composed of a subtree with at most $C_{\ell-2}$ consistent leaves and a subtree with at most $C_{\ell-1}$ such leaves. It follows for the maximum number C^A_ℓ of consistent leaves in such a tree that

$$C^A_\ell \leq C_{\ell-2} + C_{\ell-1} \leq C^*_\ell.$$

B) The tree T^B_ℓ is composed of either one or two subtrees, having at most $C_{\ell-1}$ consistent leaves each. Consequently, $C^B_\ell \leq 1/2 \cdot C_{\ell-1} + C_{\ell-1}$. Using (8.2), it follows that

$$C^B_\ell \leq C_{\ell-2} + C_{\ell-1} \leq C^*_\ell.$$

C) The tree T^C_ℓ is composed of only one subtree with at most $C_{\ell-1}$ or C_ℓ consistent leaves, resp. (with equal probability). It holds that $C^C_\ell \leq 1/2 \cdot (C_{\ell-1} + C_\ell)$, and using (8.2), it follows that

$$C^C_\ell \leq \frac{1}{2}(2C_{\ell-2} + 2C_{\ell-1}) = C_{\ell-2} + C_{\ell-1} \leq C^*_\ell.$$

D) The tree T^D_ℓ has one of the forms given in case D. Then, for the average number C^D_ℓ of consistent leaves in this tree, it holds that $C^D_\ell \leq \frac{3}{4} \cdot C_\ell$. Using (8.2) repeatedly, it follows that

$$C^D_\ell \leq \frac{3}{2} \cdot C_{\ell-1} = C_{\ell-1} + \frac{1}{2}C_{\ell-1} \leq C_{\ell-1} + C_{\ell-2} \leq C^*_\ell.$$

Since $C_\ell = \max(C^A_\ell, C^B_\ell, C^C_\ell, C^D_\ell)$, if follows that $C_\ell \leq C^*_\ell$. □

8.3.2 Size of a well-formed tree

We have shown that on the average, the number C_ℓ of consistent leaves in an arbitrary tree of guesses is not bigger than the number C^*_ℓ of consistent leaves in a well-formed tree. In the next section, an estimate for C^*_ℓ and thus an upper bound for C_ℓ will be proven.

Theorem 2 *Let C^*_ℓ denote the size of a well-formed tree T^*_ℓ. Then we have $a^\ell \leq C^*_\ell \leq \frac{2}{a}a^\ell$ for all $L \geq 1$, where $a = \frac{1+\sqrt{5}}{2} \approx 2^{0.6942419}$.* [3]

Proof: Note that for all $\ell \geq -1$, C^*_ℓ satisfies the recursion $C^*_{\ell+2} = C^*_{\ell+1} + C^*_\ell$ with $C^*_{-1} = C^*_0 = 1$.
Let a be the unique positive solution of $x^2 = x + 1$, i.e., $a = \frac{1+\sqrt{5}}{2}$. In this case, the function $F(\ell) = a^\ell$ also satisfies the recursion $F(\ell+2) = F(\ell+1) + F(\ell)$ for all $\ell \geq 0$. Since $C^*_0 = F(0)$ and $C^*_1 = \frac{2}{a}F(1)$, it follows that $a^\ell \leq C^*_\ell \leq \frac{2}{a}a^\ell$ for all $\ell \geq 0$. □

Note that $\frac{2}{a} \approx 1.236068$. Thus, the upper bound of the average search tree is $C_\ell \leq \frac{2}{a} \cdot 2^{0.694\ell} \approx 2^{0.694\ell+0.306}$.

8.3.3 Worst case considerations

The above result can be applied directly to the tree of guesses in section 8.2. Remembering that such a search tree actually has a root labelled $\ell = l - 2$, the average number of consistent leaves is upper bounded by $C_l \leq 2^{0.694l-0.918}$.

This upper bound seems to hold even for the worst case, provided that l is large enough. Remember that assumption 1 stated that in case of a linearly dependent equation, a contradiction occurs with probability 1/2. Now remember from section 8.2 that linearly dependent equations do not occur before depth $\lfloor \frac{l}{2} \rfloor$ is reached. This, in turn, means that for large l there exists a large number of nodes labelled j for each $j < l - \lfloor \frac{l}{2} \rfloor$. Thus, the law of large numbers can be applied, stating that the actual number of contradictions is very close to the expected number of contradictions. Thus, the number of consistent leaves should be close to the above bound not only for the average case, but for almost any tree of guesses.

In order to give some more weight to this rather informal argument, we will provide some empirical evidence for this conjecture in section 8.4.

8.3.4 Total running time

It remains to establish an upper bound for the maximum number of *nodes* in the tree. Since the tree will be malformed, it contains nodes that have only one child. It is thus impossible to upper bound the number of nodes by $2 \cdot C_l - 1$, as could be done for a binary tree with inner nodes of fixed outdegree 2. However, it can be proven that the maximum depth of the search tree is $l - 1$.

Proposition 2 *If the linear recurrent sequence $(a_i)_{i\geq 0}$ is an m-sequence, then the tree has maximum height of $l - 1$.* [4]

[3] The proof of this theorem is due to M. Krause.

[4] Note that this proposition only holds for m-sequences. The use of shorter sequences, however, would be a breach of elementary design principles, since it would facilitate a number of other attacks. It does not seem to increase resistance against our attack either, it just makes the proof harder.

Proof: Any node of depth d contains exactly $d+1$ equations of type 1 (and a varying number of equations of type 2). Thus, at depth $l-1$, there are exactly l such equations, namely for $a_0, a_2, \ldots, a_{2l-2}$.
By a theorem on m-sequences (see, e.g., [56], p. 76), there exists an s such that the following holds:

$$(a_s, a_{s+1}, \ldots, a_{s+l-1}) = (a_0, a_2, \ldots, a_{2l-2})$$

Since $a_s, \ldots, a_{s+l-1}$ are linearly independent, the same holds for $a_0, a_2, \ldots, a_{2l-2}$. Consequently, there are l linearly independent equations of type 1 in any node of depth $l-1$, allowing us to solve the system and derive a key candidate. Thus, no node of the tree will have depth $\geq l$. □

This fact can be used to upper bound the number of nodes. Consider the largest binary tree (w.r.t. the number of nodes) with height $l-1$ and C_l consistent leaves. This tree is a complete binary tree from depth 0 to $p := \lfloor \log C_l \rfloor$. From depth $p+1$ to depth $l-1$, the tree has constant width of C_l.

Let N_l denote the number of nodes in a search tree. It follows that N_l is at most the size of this worst possible tree.

$$N_l \leq (2^{p+1} - 1) + (l - p - 1) \cdot C_l$$

Note that both 2^{p+1} and C_l are in $O(C_l)$. Ignoring all constant summands and factors to C_l, if follows that:

$$\begin{aligned} N_l &\in O((l-p) \cdot C_l) \\ &= O(0.306\, l \cdot 2^{0.694\, l - 0.918}) \\ &= O(0.162\, l \cdot 2^{0.694\, l}) \end{aligned}$$

Remembering that in each node, one or two linear equations have to be inserted into a system of equations, and ignoring constant factors again, we derive a total asymptotic running time in $O(l^4 \cdot 2^{0.694\, l})$, requiring roughly l output bits instead of the large number of bits necessary for Mihaljević's attack as presented in section 8.1.2.

8.4 Experimental results

Results on the number of consistent leaves: In section 8.3, it was proven that the number of consistent leaves in the search tree is upper bounded by $2^{0.694l-0.918}$ in the average case. This result leaves a number of open questions. Since only an upper bound was derived: How close is this value to the average number of consistent leaves that do occur in an actual search?[5] And what about the conjecture in section 8.3.3? Is C_l also an upper bound for the worst case, for large l?

[5]We must take care not to confuse the average case of the analysis with the average number of consistent leaves in the search tree; they are quite different mathematical objects.

	Number of leaves			Number of nodes		
l	C_{avg}	C_{max}	C_{bound}	N_{avg}	N_{max}	N_{bound}
3	$2^{1.00}$	$2^{1.00}$	$2^{1.16}$	$2^{1.58}$	$2^{1.58}$	$2^{1.04}$
4	$2^{1.55}$	$2^{1.58}$	$2^{1.86}$	$2^{2.57}$	$2^{2.81}$	$2^{2.15}$
5	$2^{2.29}$	$2^{2.58}$	$2^{2.55}$	$2^{3.28}$	$2^{3.46}$	$2^{3.17}$
6	$2^{2.85}$	$2^{3.17}$	$2^{3.25}$	$2^{4.21}$	$2^{4.64}$	$2^{4.12}$
7	$2^{3.53}$	$2^{3.81}$	$2^{3.94}$	$2^{4.92}$	$2^{5.43}$	$2^{5.04}$
8	$2^{4.23}$	$2^{4.64}$	$2^{4.63}$	$2^{5.61}$	$2^{5.93}$	$2^{5.93}$
9	$2^{4.88}$	$2^{5.29}$	$2^{5.33}$	$2^{6.35}$	$2^{6.79}$	$2^{6.79}$
10	$2^{5.53}$	$2^{5.88}$	$2^{6.02}$	$2^{7.05}$	$2^{7.55}$	$2^{7.64}$
11	$2^{6.22}$	$2^{6.57}$	$2^{6.72}$	$2^{7.75}$	$2^{8.24}$	$2^{8.47}$
12	$2^{6.87}$	$2^{7.26}$	$2^{7.41}$	$2^{8.46}$	$2^{8.89}$	$2^{9.29}$
13	$2^{7.56}$	$2^{7.92}$	$2^{8.10}$	$2^{9.16}$	$2^{9.73}$	$2^{10.10}$
14	$2^{8.25}$	$2^{8.56}$	$2^{8.80}$	$2^{9.85}$	$2^{10.20}$	$2^{10.90}$
15	$2^{8.92}$	$2^{9.23}$	$2^{9.49}$	$2^{10.56}$	$2^{11.26}$	$2^{11.69}$
16	$2^{9.61}$	$2^{9.90}$	$2^{10.19}$	$2^{11.25}$	$2^{11.64}$	$2^{12.48}$

Table 8.2: Empirical Results

In order to answer those questions, the key reconstruction algorithm from section 8.2 has been implemented and tested against all keys and all m-LFSRs of lengths $l = 3, \ldots, 16$. The main results of this simulation are given in the left part of table 8.2. Here, C_{avg} and C_{max} denote the average and maximum number of consistent leaves encountered in the experiments. $C_{bound} = 2^{0.694l-0.918}$ denotes the upper bound as calculated in section 8.3. For ease of comparison, all values are given in logarithmical notation.

First observe that values C_{avg} and C_{max} are very close; they differ by a factor ϕ with $1 < \phi < 1.33$. Of course, this may or may not hold for larger values of l, but for small l, the maximum number of consistent leaves does not stray very far from the average.

Also observe that for $l > 8$, C_{bound} seems to be a proper upper bound not only for the average case, but also for the maximum number of consistent leaves in the search tree. Note especially that for $l > 8$, the gap between C_{max} and C_{bound} seems to be widening with increasing l. Nonetheless, additional empirical or mathematical evidence for larger l might be necessary before our conjecture from section 8.3.3 can be considered confirmed.

Results on the number of nodes: In the right half of the table, the results on the number of nodes are given. Again, N_{avg} and N_{max} denote the average and maximum values encountered in the experiments, while $N_{bound} = 0.162\, l \cdot 2^{0.694\, l}$ denotes the mathematical bound as given in section 8.3.4.

It seems that for $l > 7$, N_{bound} is an upper bound for the number of nodes in the worst possible case. As with the results on the number of consistent leaves, the gap between N_{max} and N_{bound} seems to be widening with increasing l, but

again, more data for larger l would be helpful. Also note that N_{avg} and N_{bound} are very close to each other.

An interesting side observation is that $N_{avg} \approx 2 \cdot C_{bound}$, i.e., the average number of nodes appears to be almost exactly twice the mathematical upper bound for the number of consistent leaves as derived in section 8.3.3. This is not apparent from the mathematical analysis in section 8.3 and may thus be an interesting starting point for future research.

8.5 Conclusions

General observations: Using the example of the self-shrinking generator, it was demonstrated that the dynamic LCT attack can be an efficient way of attacking a PRG, especially if only a small number of output bits is available. Nonetheless, for realistic key sizes between 120 and 200 bits, the attack is currently not feasible in practice.

When analysing the running time of the algorithm, it became apparent that estimating the size of the search tree requires some thought. For many other generators, a similar effort is necessary in order to determine the computational resources required by the attack.

Design recommendations: For a modern key size of 120 bits, the search tree has a size of more than 2^{83} nodes, making the dynamic LCT attack infeasible in practice. Note, however, that the attack is easily parallelised, allowing an adversary to use as many parallel processors at once as he can afford. Since each processor can operate on its own segment of the tree (without any need of communication), k processors can reduce the running time by a factor of k. Thus, a generator using a shorter register is in real danger of being compromised. It can be concluded that 120 bit should be the **minimum length** of a self-shrinking generator.

Note that our attack relies on the feedback logic of the register to be known. If this is not the case, the attack has to be repeated for all m-LFSRs of length l, yielding an additional working factor of $\phi(2^l - 1)/l$, where ϕ denotes the Euler function.[6] Security of the self-shrinking generator can thus be increased significantly by following the proposal given in [22, 85]: Use a **programmable feedback logic** and make the actual feedback recurrence a part of the key.

Finally, observe that the use of sparse feedback recurrence makes our attack slightly more effective. If the more significant bits depend on only a few of the less significant bits, the probability of linearly dependent equations increases, yielding a tree of guesses that is more slender than the average case tree considered above. However, as stated in section 8.4, the sizes of worst case and best case trees seem to differ by less than a factor 2. Nonetheless, **sparse feedback recurrences** should be avoided in designing most PRGs, the self-shrinking generator being no exception.

[6] Again, refer to [86] for definitions.

Chapter 9

Dynamic LCT and Clock-Controlled Generators

9.1 Introduction

As described in section 3.3, clock control is an important technique for transforming the output of one or more LFSRs into a nonlinear bitstream. Many PRGs are based on this principle. In this chapter, it will be shown how the dynamic LCT can be applied against a class of such clock-controlled generators, and how a general upper bound for the resulting search trees can be obtained. Originally, these topics have been discussed in [128].

The class of generators: Remember that S_t denotes the inner state of a generator at time t, where S_0 is the initial state or key. Each inner state S_t determines uniquely a clock control behaviour ξ_t (sometimes referred to as "clocking") that leads to the inner state S_{t+1}.

$$S_0 \xrightarrow{\xi_0} S_1 \xrightarrow{\xi_1} S_2 \xrightarrow{\xi_2} \ldots$$

Also remember that from the inner states $S_0, S_1, \ldots$, the generator derives the output stream $z = (z_0, z_1, \ldots)$. In the following, clock control generators with the following properties are considered:

1. *The output bit depends on the inner state of the generator in some linear way.*
   For each clock cycle t and each assignment to the output bit z_t, a linear equation L can be given such that the inner state S_t generates output bit z_t iff S_t is a solution to L.

2. *The behaviour of the clock control depends on the inner state of the generator in some linear way.*
   For each clock cycle t and each assignment to the clock control behaviour ξ_t, a set M' of linear equations can be given such that the inner state S_t generates the clock control value ξ_t iff S_t is a solution to M'.

3. *The number of possible behaviours of the internal clock is small.*

Clock control guessing: Given a generator that has properties 1-3, the attacker can modify the dynamic LCT attack introduced in chapter 7 by guessing the clock control behaviour ξ_t for $t = 0, 1, \ldots$. Since condition 2 holds, he obtains a set of linear equations for each such guess, making the attack an application of the equation guessing technique introduced in section 7.5. From condition 1 and his knowledge of the LFSR clockings so far, he also obtains one equation per guess that depends on the output stream z.

In simple cases like the $\{1, 2\}$-clocked generator, clock control guessing is identical to simple bit guessing. Guessing the behaviour of the clock control is equivalent to guessing the bits $c_0, c_1, \ldots$. Thus, for this particular generator, clock control guessing turns out to be the dynamical LCT attack as described in section 7.3. There are, however, more contrieved clock control designs like that of A5/1, which will be introduced in section 9.3. In the next section, a simple technique will be presented for estimating the running time of clock control guessing against all generators having properties 1-3, no matter how simple or complicated their clock control rule is.

9.2 On the efficiency of clock control guessing

Estimating the running time: As mentioned in section 7.4, a precise estimate of the running time (i.e., the number of nodes in the search tree) is not possible without paying close attention to the details of the cipher considered. The length of the registers, the sparseness of the feedback recurrences, the positions of the output and clock control bits, the choice of the output and clock control function, and even the values of the output bits all determine the efficiency of the attack.

In case of the clock-controlled generators meeting conditions 1-3, however, a general upper bound for the size of the search tree can be proven. In order to do this, the generator is assumed to meet the following additional condition:

4. *The number of seeds S_0 that are consistent with the first d output bits ($d \leq l$) is approximately 2^{l-d}.*

Note that this condition is met by all properly designed pseudorandom generators, since otherwise, successful statistical tests could be developed. Now the maximum width of the search tree can be estimated, using an elegant technique proposed by Krause in [71]. To this end, consider the following simple observations.

Observation 1: Consider a node v in the search tree at depth d. Such a node is reached by a sequence $b_0, b_1, \ldots, b_{d-1}$ of guesses for the clock control behaviour. It contains a system M of linear equations derived on the path from the root to the node by using properties 1 and 2 of the generator. The set of solutions to M has the following properties:

a) All solutions to M produce the clock control sequence $b_0, b_1, \ldots, b_{d-1}$.

b) All solutions to M produce the output sequence $z_0, z_1, \ldots, z_{d-1}$.

c) If M is consistent, there is at least one solution to M.

We say that the node v *represents* all inner states that are solutions to M, and that v is consistent if M is consistent. As a consequence of property a, no two nodes at depth d represent the same inner state, since different nodes imply different behaviours of the clock control. On the other hand, no node v represents an inner state that is inconsistent with the output bits $z_0, \ldots, z_{d-1}$. From property 4 of the generator, it follows that there are approximately 2^{l-d} solutions represented by all nodes at depth d. Since by property c, there are no empty consistent nodes, there can be at most 2^{l-d} consistent nodes at depth d. For low values of d, however, the number of consistent nodes is going to be a lot smaller since each node represents a huge number of inner states.

Observation 2: On the other hand, the number of nodes in the tree at depth d can never be larger than k^d, where k is the number of possible behaviours of the clock control. For small values of d, this estimate will usually be exact, while for larger values of d, the actual tree contains a lot less nodes than indicated by this number.

Width of the search tree: Observe that the function 2^{l-d} is constantly decreasing in d, while k^d is constantly increasing. Since the number of consistent nodes in the tree is upper bounded by both of these functions, the maximum number of nodes at a given depth d is upper bounded by $\min\{2^{l-d}, k^d\}$. Writing $k^d = 2^{\log(k)\cdot d}$ for convenience, the maximum number of nodes must be smaller than 2^w with $w = l - d$, yielding

$$\begin{aligned} 2^w &= 2^{\log(k)\cdot(l-w)} \\ \Leftrightarrow \quad w &= \log(k)\cdot(l-w) \\ \Leftrightarrow \quad w &= \frac{\log(k)}{\log(k)+1} l \end{aligned}$$

Thus, the number of consistent nodes in the widest part of the search tree cannot exceed $2^{c\cdot l}$ with $c = \frac{\log(k)}{\log(k)+1}$. Note that this is not an asymptotical result; it is perfectly valid to use concrete values for k and l and to calculate the upper bound.

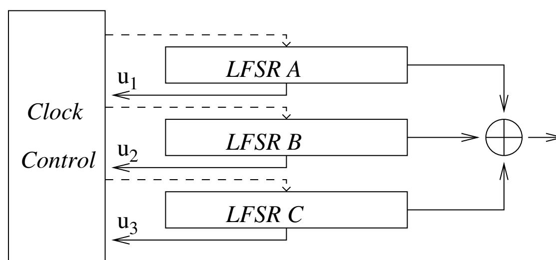


Figure 9.1: The A5/1 Generator

Total running time: After obtaining an upper bound on the width of the search tree, the total running time is easily determined. Observing that

- there are at most two layers with width 2^w, that
- all layers above those two have at most 2^w consistent nodes amongst them, and that
- all layers below those two have at most 2^w consistent nodes amongst them,

it follows that the tree has at most $4 \cdot 2^w$ consistent nodes. Observing further that there must be less than k non-consistent nodes for each consistent node, the number of recursive function calls is limited by $4 \cdot (k+1) \cdot 2^w \in O(2^w)$. Thus, the overall running time must be in $O(l^3 \cdot 2^{c \cdot l})$ with $c = \frac{\log(k)}{\log(k)+1}$.

9.3 Application: attacking A5/1

Description of the cipher: A5/1 is the encryption algorithm used by the GSM standard for mobile phones; it was described in [15]. The core building block is a PRG, consisting of three LFSRs with a total length of 64 bit. First, the output is generated as the sum (mod 2) of the least significant bits of the three registers. Then the registers are clocked in a stop-and-go fashion according to the following rule:

- Each register delivers one bit to the clock control. The position of the clock control tap is fixed for each register.
- A register is clocked iff its clock control bit agrees with the majority of all clock control bits.

An illustration of the generator is given in figure 9.1, where dotted lines denote the clock control and straight lines denote the LFSR outputs.

Clock control guessing: Against the A5/1 generator, the clock control guessing attack was discussed earlier by Zenner [127], Golić [48], and Pornin and Stern [97]. First observe that the A5/1 generator produces 1 output bit per master clock cycle, and that there are 4 different behaviours of the clock control. Let u_1, u_2 and u_3 denote the contents of the clock control bits for a given clock cycle. Table 9.1 gives the dependency between u_1, u_2, u_3 and the behaviour ξ of the clock control. Note that equivalent linear equations are easily constructed. Thus, it follows that the A5/1 algorithm meets all prerequisites for a success-

ξ	Equation
(011)	$u_1 \neq u_2 = u_3$
(101)	$u_1 \neq u_2 \neq u_3$
(110)	$u_1 = u_2 \neq u_3$
(111)	$u_1 = u_2 = u_3$

Table 9.1: Clock control and linear equations

ful clock control guessing attack. The attacker guesses the behaviour of the clock control for each output bit, derives the linear equations and checks for consistency.

Upper bounding the running time: Applying our estimation technique to the A5/1, two facts can be observed:

1. The seed is generated in such a way that only $\frac{5}{8} \cdot 2^{64}$ states are in fact possible. The impossible states can be excluded by a number of simple linear equations (for details, see [46]). Thus, the efficient key length of the inner state is only $64 + \log(\frac{5}{8}) \approx 63.32$ bit.

2. Furthermore, the first output bit is not yet dependent on the clock control. Thus, the efficient key length of the inner state prior to any clock control guessing is further reduced by 1 bit, yielding $l \approx 62.32$.

For each master clock cycle, 4 possible behaviours of the clock control are possible. Thus, $k = 4$ and $\log(k) = 2$. Using the estimate from section 9.2, it follows that the search tree has a maximum width of $2^{(2/3) \cdot 62.32} \approx 2^{41.547}$ nodes.

This result coincides with the maximum number of leaves as given by Golić in [48], derived from a more involved analysis. Also note that in the same work, the average number of leaves was estimated to be $2^{40.1}$, as was to be expected: By paying close attention to important details of the generator such as the position of the feedback taps or the lengths of the registers, an estimate for the tree size can be derived that in most cases will be lower than the general upper bound. Nonetheless, this upper bound gives a first indication of a PRG's strength by ruling out some weak generators without further effort.

Test run on a small version: In order to demonstrate the difference between the proven upper bound and the actual running time, a 40-bit version of the A5/1 was implemented, featuring the details given in table 9.2.

LFSR	length	feedback recurrence	clock control tap
A	11	$a_i = a_{i-9} + a_{i-11}$	a_6
B	14	$b_i = b_{i-9} + b_{i-11} + b_{i-13} + b_{i-14}$	b_7
C	15	$c_i = c_{i-3} + c_{i-11} + c_{i-13} + c_{i-15}$	c_8

Table 9.2: 40-bit version of the A5/1 generator

Again, observe that the first output bit is not yet dependent on the clock control, yielding 2^{39} candidates for the seed or an efficient key length of $l = 39$ bit.[1] Thus, the bounding functions are 4^d and 2^{39-d}, yielding a maximum search tree width of 2^{26}.

An total of 120 experiments was conducted, and the results are shown in figure 9.2. The figure shows the average widths of the search trees that were

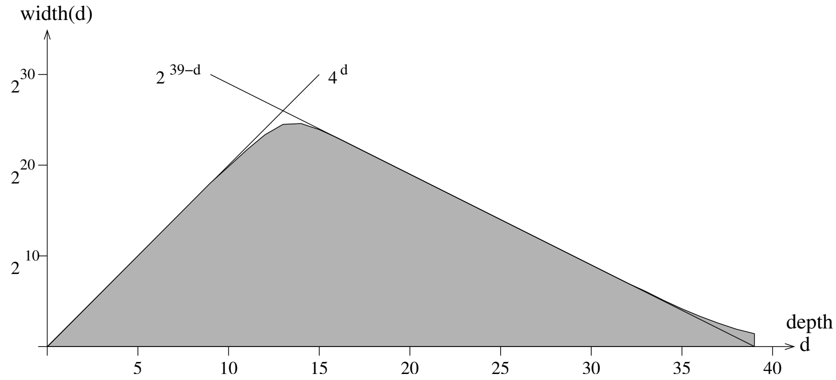


Figure 9.2: Width of search tree for 40-bit A5/1 generator

found in the experiments. It also gives the bounding functions 4^d and 2^{39-d} for convenience. The following observations can be made:

- The tree width at depth d matches the predicted value of $\min(4^d, 2^{39-d})$ surprisingly well.
- In the widest part of the tree ($d = 14$), the actual number of nodes is smaller than the predicted upper bound, which was to be expected.
- In the lowest part of the tree ($d > 34$), the actual number of nodes is larger than predicted by the function 2^{39-d}. This is due to the fact that for the

[1] For simplicity's sake, we ignore the fact that only $\frac{5}{8} \cdot 2^{40}$ inner states are actually possible.

A5/1 generator, there is a chance that several inner states map onto the same output sequence, i.e., assumption 4 does not hold for high values of d. This, however, does not affect the performance of the algorithm, since the running time is almost exclusively determined by the widest part of the tree.

In our experiments, an average of 1.758 inner states that produce the same output were found. Judging from the empirical data as given in table 9.3, it seems that the probability of an output stream (generated from a random seed) having m generating keys is approximately 2^{-m} for small values of m. Whether or not this assumption is correct and whether or not it also holds for the full version of A5/1 remains an open problem.

equivalent keys	1	2	3	4	5	6	7
frequency	64	33	17	2	3	-	1

Table 9.3: Frequency of equivalent keys for 40-bit A5/1 generator

9.4 Other generators

In this section, some other generators from the literature will be reviewed and some dos and don'ts when using the above attack and the associated technique for upper bounding the efficient key length will be pointed out.

Alternating step generator: As a first example, consider the alternating step generator [57]. The generator consists of three LFSRs C, A and B. For each clock t, the output c_t of LFSR C is determined. If $c_t = 0$, clock LFSR A, else clock LFSR B. Finally, add the current output bit of LFSRs A and B (modulo 2) and append it to the output stream.

Noting that there are only two options for the clock control, it follows that $\log(k) = \log(2) = 1$ and thus $w = l/2$. Consequently, there is an absolute upper bound of $0.5l$ bit on the efficient key size of this kind of generator. This holds independently of the the choice of all other paramters. In particular, while increasing the length of LFSR C at the expense of LFSRs A and B improves protection against simple LCT attacks, this measure remains useless against dynamic LCT using clock control guessing.

Stop-and-go generator: The $\{0, 1\}$-clocked generator (also denoted as stop-and-go generator [8]) consists of two LFSRs C and A, where the output bit is taken as the least significant bit of LFSR A. While LFSR C is clocked regularly and outputs $c_0, c_1, \ldots$, LFSR A is clocked iff $c_t = 1$. As a consequence, the output sequence y has a probability of 3/4 that the condition $z_t = z_{t-1}$ holds. Thus, certain output sequence prefixes are much more likely than others, contradicting property 4. Thus, even though the clock control guessing attack

can be implemented against the stop-and-go generator, the estimate cannot be used without further thought.

$\{1,2\}$-clocked generator: Simply by changing the clocking delivered by LFSR C to $\{1,2\}$ instead of $\{0,1\}$ while leaving the rest of the design unchanged, the output anomaly of the stop-and-go generator disappears. Since the behaviour of the clock control can be described as for the alternating step generator and since there are only 2 possible behaviours of the clock control, the upper bound for the efficient key length of the step1-step2 generator must be $0.5\,l$, independently of the individual parameters.

[1..D]-decimating generator: More generally, a generator might pick some bits from LFSR C and interpret them as a positive number $\xi \in \{1,\ldots,D\}$. Then, register A is clocked ξ times before delivering the next output bit. Such a generator is called [1..D]-decimating generator [55]. If it meets conditions 1-4, a clock control guessing attack is possible and has an efficient key length of at most $\frac{\log(D)}{\log(D)+1}\,l$ bit.

Cascade generator: A [1..D] decimating generator can be further generalised by turning it into a cascade, using s LFSRs $X_1,\ldots,X_s$ instead of just 2. In [55], Gollmann and Chambers describe some possible constructions for cascade generators obtaining good statistical bitstream properties.

A typical example is a cascade of stop-and-go generators where the output bit of LFSR X_i controls the clocking of LFSR X_{i+1} and is also added to the output of LFSR X_{i+1}. Since the basic clock-control mechanism (stop-and-go) meets conditions 1-3, the cascade generator can be attacked using clock control guessing. Since the cascade (as opposed to the simple stop-and-go generator) meets assumption 4, the above technique can be used to derive an upper bound on the effective key length. Note that there are $k = 2^{s-1}$ possible behaviours for the clock control, yielding $\log(k) = s-1$ and an efficient key length of at most $\frac{s-1}{s}\,l$.

Note that this is not identical to the naïve LCT attack of guessing the contents of the uppermost $s-1$ registers and deriving the content of the lowest LFSR from the output stream. This naïve attack has computational cost in the order of $O(2^{l-l_X})$, where l_X is the length of the final LFSR. If $l_X < \frac{l}{s}$, the clock control guessing attack will usually be more efficient than the simple LCT attack.

Shrinking generator: Note that the shrinking generator, too, can be viewed as a clock-controlled generator, where register A is clocked once with probability 1/2, twice with probability 1/4 a.s.o. before producing one bit of output. Thus, the number of possible clock control behaviours is rather large (up to l_C different possibilities), the property 3 is violated, and the attack is not applicable in a straightforward manner. In this case, the bit guessing attack presented in chapter 8 seems to obtain better results.

9.5 Conclusions

Design recommendations: We have shown that while running time estimates for dynamic LCT attacks against general PRGs are not easily found, an upper bound can be given for a class of clock-controlled generators. Most generators proposed in the literature that belong to this class have rather simplistic clock control rules, often yielding $k = 2$ and thus cutting the efficient key length down to $l/2$ without any further analysis. If this is not acceptable, any of the following design changes increases resistance against our attack:

- Increase the number of possible behaviours for the clock control. As a consequence, the search tree expands rather rapidly, making the search more difficult.
- Choose a non-linear function for the clock control.
- Choose a non-linear function for the output bit extraction.

The LILI generator: A generic example of a clock-controlled pseudorandom generator that can be designed to follow all of those design criteria is the LILI generator [113]. The generator consists of two LFSRs C and A, where C determines the clock control and A the output. The clock control c_t is determined from the inner state of LFSR C by a bijective function $f_c : \{0,1\}^m \to \{1,\ldots,2^m\}$, and the output bit y_t is computed from the inner state of LFSR A using a Boolean function $f_d : \{0,1\}^n \to \{0,1\}$. If the values m and n are chosen large enough and if the functions f_c and f_d are non-linear, the generator should be safe from clock control guessing attacks[2].

As mentioned before, however, security against one class of attacks does not necessarily imply security of the generator in general. In the case of the LILI generator, correlation attacks proved to be fatal [65], as did time-memory trade-off attacks [2, 105]. In way of a conclusion, note again that good cipher designs have to resist all known cryptanalytic techniques, with clock control guessing being just one of them.

[2]The mapping $f_c(x_1,\ldots,x_k) = 1 + x_1 + 2x_2 + \ldots + 2^{k-1}x_k$ that was proposed by the authors is easily modelled using linear equations. This should not be a problem, as long as the other design criteria are met. For paranoia's sake, however, a non-linear permutation might be considered instead.

Part IV

The Role of the Inner State

Chapter 10

Deployment of PRGs in Stream Ciphers

10.1 Motivation

Seed and key, revisited: So far, when considering a pseudorandom generator, it was assumed that the seed S_0 was equal to the key k (see section 3.1). There are, however, a number of reasons why for practical ciphers, seed and key are different concepts:

- Remember from section 2.1 that in most cases, sender and receiver want to exchange more than one message before exchanging new keys. When using a PRG, this means that two messages m, m' would be encrypted to ciphertexts c, c' using the same PRG output stream z. In this case, the following property holds for all $i = 0, 1, \ldots$:

$$\begin{aligned} c_i \oplus c'_i &= (m_i \oplus z_i) \oplus (m'_i \oplus z_i) \\ &= m_i \oplus m'_i \end{aligned}$$

  In other words, the bitwise sum of the ciphertexts equals the bitwise sum of the plaintexts, enabling the attacker to mount a ciphertext-only attack using the plaintext statistics. In order to prevent such a scenario, the seed S_0 is computed not only from the key k, but also from a so-called *nonce* value[1]. This value is publicly known, but changes for every message which is to be transmitted.

- An alternative solution would be for both sender and receiver to memorise the current inner state S_i after transmitting message m, resuming encryption with state S_{i+1} for the subsequent message m'. In practice, however, designers face the so-called *synchronisation problem*: When bits are lost or replicated in transmission, sender and receiver obtain different

[1] Nonce stands for a "number used once", cf., e.g., [33], p. 72.

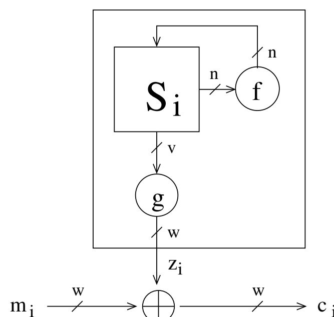


Figure 10.1: General model of a pseudorandom generator

values for the counter i. In order to restore synchronisation, the PRG is reset to an initial value after a certain number of bits. Thus, keeping an ongoing counter i over a longer period is not a viable option; instead, frequent re-initialisation with varying seed values is necessary.

- Another advantage of separating key and seed for practical systems is the possibility to choose between different key lengths. In some countries, limits on the allowable key sizes exist, forcing cryptographic products that are exported to different countries to support different key sizes. In this case, it is advantageous if the PRG (including the seed size) is the same for all products, while the key size can be chosen as desired.

- Finally, the security of a PRG-based encryption scheme can be improved by choosing the seed size larger than the key size. This will be elaborated on in section 11.2.

10.2 Extending the basic model

Basic model: Remember the model of a PRG from section 3.1. In a slight modification of the initial description, the generator is now allowed to produce w bit of output at once. In addition, a set $\mathcal{S}$ of valid inner states is defined, yielding a PRG G with the following components (see figure 10.1):

(a) An inner state $S_i \in \mathcal{S}$ with $\mathcal{S} \subseteq \{0,1\}^n$,

(b) an update function $f : \mathcal{S} \rightarrow \mathcal{S}$ that modifies the inner state with each clock, and

(c) an output function $g : \{0,1\}^v \rightarrow \{0,1\}^w$, $w \leq v \leq n$, that uses the inner state to compute w output bits with each clock.

Deployment in stream ciphers: A *stream cipher* is an encryption scheme where each block of plaintext is encrypted in a time-dependend fashion, i.e., the method used to encrypt m_i differs from the method used for m_{i+1}. Obviously, adding the output of a PRG bitwise to the plaintext is a simple form of a stream cipher. More generally, when using a PRG in a stream cipher, the following additional components are required:

(A) A secret *key* $k \in \{0,1\}^l$ that is not necessarily identical to the seed S_0,

(B) an *initialisation function* $h : \{0,1\}^l \times \{0,1\}^m \to \mathcal{S}$ that derives the seed S_0 from the key k and an m-bit nonce value N, and

(C) the *xor-function* $\oplus : \{0,1\}^w \times \{0,1\}^w \to \{0,1\}^w$ which adds the PRG output bitwise modulo 2 to the plaintext, generating the ciphertext[2].

Attacker model: Recall from section 2.1 that the attacker knows all about the stream cipher with the exception of k. In particular, apart from being aware of the inner workings of the PRG, he also knows the initialisation function h and the nonce value N. For a rigorous analysis, N is often even assumed to be under the control of the attacker, i.e., he may choose any value for N and can obtain some output bits generated by the PRG under N and the unknown key k.

In section 3.1, the attacker was considered successful if he either reconstructs the set $\mathcal{M}'$ of consistent messages (*prediction attack*) or the set $\mathcal{K}'$ of consistent keys (*key reconstruction attack*). If $\{z_{i_1}, \ldots, z_{i_s}\}$ is the set of known output bits and $G_i(S_0)$ denotes the i-th output bit of the generator G initialised with seed S_0, these definitions have to be adjusted as follows:

$$\begin{aligned} z' \in \mathcal{Z}' &\quad\Leftrightarrow\quad z'_i = z_i \;\; \forall i \in \{i_1, \ldots, i_s\} \;\; \text{and} \;\; \exists k' \in \mathcal{K} : G(h(k', N)) = z' \\ k' \in \mathcal{K}' &\quad\Leftrightarrow\quad G_i(h(k', N)) = z_i \;\; \forall i \in \{i_1, \ldots, i_s\} \end{aligned}$$

In addition, the attacker will also be considered successful if he can reconstruct the set $\mathcal{S}'$ of consistent seed values, which is defined as

$$S_0 \in \mathcal{S}' \quad\Leftrightarrow\quad G_i(S_0) = z_i \;\; \forall i \in \{i_1, \ldots, i_s\} \;\; \text{and} \;\; \exists k' \in \mathcal{K} : h(k', N) = S_0 \; .$$

This type of attack is denoted as *state reconstruction attack*. Note that the set $\mathcal{S}'$ is smaller than the set of all seed values producing $z_{i_1}, \ldots, z_{i_s}$, which may or may not be an advantage when cryptanalysing the stream cipher. Also note that finding $\mathcal{K}'$ implies finding $\mathcal{S}'$, which in turn implies finding $\mathcal{Z}'$.

Distinguishing attacks, revisited: Remember from section 2.2.2 that in the asymptotical security model, an attacker is considered successful if he can distinguish the encryption function from a truly random function with significant probability. Applying this concept to a PRG-based stream cipher, an

[2] In fact, other functions like addition modulo 2^w are also possible, but rarely used.

attacker would be successful if he can tell a random bit stream apart from an output stream generated by the generator.

In the asymptotical model, a distinguishing attack is equivalent to a prediction attack [123].[3] Transferring the concept into the empirical security model is possible, but not all distinguishers running in less than 2^l steps imply the existence of an efficient prediction algorithm.

As a consequence, the practical relevance of distinguishing attacks against stream ciphers is disputed [99]. Nonetheless, a successful distinguishing attack may indicate a weakness of the stream cipher under consideration. For this reason, security against distinguishing attacks will be required from the stream ciphers considered in the next sections.

10.3 Outlook

Problem statement: While a large body of literature exists on the design of pseudorandom generators (cf. [103, 60], or chapters 2-6 of this thesis), the deployment of a PRG as a stream cipher is less well researched. Only few guidelines exist for the choice of important parameters like key length, inner state size, or the number of bits produced before re-keying. The same uncertainty exists with respect to the initialisation function h.

The consequences in practical stream cipher design are twofold. On one hand, an increasing number of stream ciphers is broken not by attacking the PRG, but by attacking the initialisation function (e.g., RC4 as used in the WEP protocol [116], or A5/1 from the GSM standard [32]). There exist a few general attack techniques against weak setup functions for stream ciphers (e.g., resynchronisation attacks [27, 52]), but no design criteria for good initialisation functions. Considering recent research progress on related key attacks for pseudorandom functions (see [9] and subsequent work), more problems for stream ciphers designed in an ad-hoc manner are to be expected in the future.

On the other hand, when a cipher is successfully attacked, a common solution is to change the parameters while keeping the general design intact. Examples include increasing the inner state size (e.g., for LILI-II [20]) or decreasing the security level (e.g., for Sober-128 [58]). For some ciphers, huge security margins for the parameters are used in the first place (e.g., more than 33,000 bit of inner state for SEAL [98]), making the stream cipher unsuitable for resource-restricted applications.

The role of the inner state: The remaining contents of part IV are taken from [130, 131]. They consider the construction of a stream cipher from a PRG and a matching initialisation function. Note that the inner state of the cipher forms the interface between those two primitives. While a large inner state is advantageous for the security of the PRG, it makes the task of the initialisation

[3] If the generator output can be predicted, it can obviously be distinguished from random. If distinguishing is possible, the next bits can be guessed and checked for correctness using the distinguisher.

algorithm more difficult. Thus, the inner state should be chosen as large as necessary, but as small as possible.

The goal of chapter 11 is to improve the understanding of the necessity and the limitations of the inner state. To this end, a formal definition of the inner state size is given in section 11.1, along with an illustration why such a definition is not as trivial as it may seem. Section 11.2 discusses the cryptographic relevance of inner states, giving lower bounds on the minimum size as well as a construction for a secure stream cipher when inner state size and initialisation time are not critical. The chapter is concluded by section 11.3, highlighting some of the disadvantages associated with allowing such large inner states.

In chapter 12, a design criterion denoted as *efficient inner state size* is introduced which measures the contribution of the inner state size to the security of the stream cipher. After a formal definition is given in section 12.1, a number of practical stream ciphers is surveyed (section 12.2). The results are compared (section 12.3), leading to the conclusion that not unexpectedly, large inner states do not make strong stream ciphers as long as the underlying PRG is cryptographically weak.

Chapter 11

On the Role of the Inner State Size

11.1 Defining the inner state size

11.1.1 Problem illustration

Surprisingly, defining the size of the inner state is not as trivial as it may seem. Consider the following examples as an illustration.

SOBER-128: This nonlinear filter generator proposed by Hawkes and Rose [58] is an almost ideal case. The inner state consists of

- an LFSR with 17 words taken from GF(2^{32}),[1] producing an m-sequence with period $2^{544} - 1$, and
- a key-dependend constant of 32 bit length.

Assume that the cipher is used with a large key ($l \gg 32$) and a nonce value. In this case, for all inner states S with the exception of the all zero LFSR assignments, there exists a key k, nonce value N, and count i such that $S_0 = h(k, N)$ and $f^i(S_0) = S$. Thus, there is little doubt that the inner state size is 576 bit.

RC4: The algorithm that is generally assumed to be Rivest's RC4 generator [75] takes some more thought. At first glance, the 8-bit version uses an inner state table that consists of 256 bytes, along with two 8-bit variables, yielding an inner state of $2048 + 16 = 2064$ bit.

[1] Note that while in section 3.2, LFSRs were defined over GF(2), a theory exists to construct m-LFSRs over any finite field (see, e.g., [73, 100]).

However, this table is used to store permutations over $\mathbb{Z}_{256}$, which reduces the number of possible table states to $256! \approx 2^{1684}$. Thus, it could be argued that the inner state is about $1684 + 16 = 1700$ bit long.

If, however, the size is defined by the number of states that can actually be attained, things get even more complicated. The initial values of the 8-bit variables are key-independent, and it was demonstrated by Finney [35] that an easily characterised class of inner states can never occur. Thus, the inner state size lies somewhere between 1684 (derived from the number of valid starting states) and 1700 (derived from the number of representable states), where none of the bounds is tight.

SEAL 3.0: The generator proposed by Coppersmith and Rogaway [98] uses two different kinds of inner states. On one hand, there are 8 32-bit variables and 12 counter bits that change constantly over time. Since they can in principle attain all possible values, they contribute 268 bit to the inner state size.

On the other hand, however, the algorithm uses huge lookup tables R, S and T with a total size of 32,768 bits. These tables are generated from the 160-bit key and a 32-bit nonce using a hash function in counter mode.[2] Since they are never modified during output stream generation, only 2^{192} different assignments to the tables are possible. One might be tempted to state that the tables contribute only 192 bit to the size of the inner state, but then again, no efficient algorithm is known that distinguishes a valid table setting from an invalid one. This means that in practice, a possible attacker faces the full state space of 33,036 bits.

Adding to the conceptional confusion, one table contains values that are used only once during the encryption process. Thus, given enough processing time, the corresponding entries could be calculated as need arises, making it possible to replace a 8,192 bit table by a simple 6-bit counter in an algorithmic implementation. This raises the question of whether or not the inner state size is reduced, too.

11.1.2 Autonomous finite state machines

A naïve candidate for the inner state size is the length n of the inner state representation, as described in section 10.2. There is, however, the obvious problem that the same generator may be represented in different ways, yielding different values of n depending on the concrete implementation.

Instead, in order to derive a unique definition of the inner state size, consider an *autonomous finite state machine* (AFSM) implementing the generator. Such an AFSM consists of a set $\mathcal{S}$ of inner states, and for each inner state $S \in \mathcal{S}$, there exists

- a *transition rule* that defines the next state $f(S)$ for S, and
- a *label* defining the output $g(S)$ generated from S.

[2] For definitions, see any textbook on cryptography, e.g., [86].

In addition, each finite state machine needs a set $\mathcal{S}_0$ of valid *starting states*.

Note that there exists an infinite number of AFSMs describing a given generator. In particular, the size of the AFSMs (i.e., the number of inner states) can vary arbitrarily. Thus, in order to find a unique value for the number of inner states, the notion of the *minimal AFSM* describing the generator is introduced.

An AFSM is said to *generate* an (infinite) output sequence $z = (z_0, z_1, \ldots)$ if there exists a starting state $S_0 \in \mathcal{S}_0$ such that $z_i = g(f^i(S_0))$ for all $i = 0, 1, \ldots$. Two AFSMs A and B are said to be *equivalent* if all (infinite) output sequences produced by A are also produced by B, and vice versa. As a consequence, all AFSMs that describe a given PRG are equivalent. An AFSM is said to be *minimal* if no equivalent AFSM of smaller size exists. Thus, if a minimal AFSM for a given generator can be found, its size yields the minimal number of inner states required to implement the generator.

11.1.3 Valid starting states

The size of a minimal AFSM for a given generator depends on the set $\mathcal{S}_0$ of valid starting states. Consider, as a toy example, a 2-bit version of the RC4 generator, where the inner state consists of two 2-bit variables and a table representing a permutation over $\{0, 1, 2, 3\}$.

1. If all assignments to the two 2-bit variables are allowed and all assignments to a 4×2-bit table as initial states, then the minimal AFSM has $2^{4+8} = 4096$ inner states, all of which are starting states.

2. If all assignments to the variables are allowed, but the table entries are restricted to correct permutations, the minimal AFSM will have $2^4 \cdot 4! = 384$ inner states, all of which are starting states.

3. If the variables are initialised to zero (as we should for a correct RC4 implementation), there are only $4! = 24$ starting states left, and exactly 24 states are no longer reachable. Thus, the size of the minimal AFSM drops to 360.

4. If the initialisation function h and the key length l are taken into account, the number of starting states may even be smaller, which in turn may or may not affect the size of the minimal AFSM.

Note that the inner state size should depend only on the PRG. Thus, the initialisation function must not be considered when defining $\mathcal{S}_0$, discarding case 4. However, what should be known is the interface between the PRG and the initialisation function: A set of conditions that the output of *any* initialisation function must meet in order to guarantee the correct working of the generator. In the case of 2-bit RC4, those conditions would be the ones described in case 3: Both variables must be set to zero, and the table must contain an arbitary permutation over $\{0, 1, 2, 3\}$.

11.1.4 Final definition

Basic model, extended: A PRG consists of an inner state space $\mathcal{S}$, an update function f, and an output function g, as described in conditions (a) to (c) in section 10.2. However, it must also have an additional component, namely

(d) a Boolean predicate $C : \mathcal{S} \to \{0, 1\}$, such that an inner state S is a valid starting state iff $C(S) = 1$.

Analogously, condition (B) in section 10.2 must be corrected such that a stream cipher contains

(B) an *initialisation function* $h : \{0, 1\}^l \times \{0, 1\}^m \to \mathcal{S}$ that derives a starting state S_0 from the key k and an m-bit nonce value N, such that $C(S_0) = 1$.

Inner state size: Given the above definitions of a PRG, a (minimal) autonomous finite state machine and its size, the unique inner state size of the generator can be defined as follows:

Definition 2 *Let G be a PRG as defined above, and let A be a minimal AFSM implementing G. Then the inner state size of the generator G is defined as $\hat{n} := \lceil \log(|A|) \rceil$, where $|A|$ is the number of inner states of A.*

Note that for many generators, only an upper bound on the inner state size can be given. Only in a few cases (like LFSR), it can be proven that all presumed inner states are actually reachable from a valid starting state.

11.2 Advantages

11.2.1 The necessity of large inner states

Remember that l denotes the key length, $\hat{n}$ the inner state size and n the length of the inner state representation ($n \geq \hat{n}$). For most practical stream ciphers, it can be observed that $n > l$ holds. In this subsection, it will be shown that this is in fact a necessary condition for secure stream ciphers.

First lower bound: The main design goals of practical stream ciphers are security and efficiency. In order to achieve the efficiency goal, the functions f, g and h are chosen to be as simple as possible. In particular, $g : \{0, 1\}^v \to \{0, 1\}^w$ is often constructed such that $w \leq v < \min\{l, n\}$.

Lemma 1 *Let the output function g depend on $v < l$ inner state bits and let the output be balanced. Then the PRG cannot be secure if $n < l + w$.*

Proof: For such a generator, a guess-and-verify attack can be developed: The attacker guesses all v bits of the inner state representation that are input to g (since $v < l$, this is feasible in our attack model). He then verifies whether the output of g matches the observed value z_0. Since g is balanced, only 2^{-w} of all assignments meet this criterion, strongly reducing the search space. The attacker can now mount a complete search over the remaining assignments, yielding an attack in 2^{n-w} steps. If $n < l + w$, this attack would be more efficient than brute force search over the key space of the stream cipher. □

Since the value n depends on the implementation and is thus not under the control of the cipher designer, the inner state size must be chosen such that the above attack becomes infeasible for all implementations.

Corollary 1 *If $v < l$, a necessary condition for a secure PRG is $\hat{n} \geq l + w$.*

Note that for many ciphers, this attack can be extended using a backtracking approach like the one described in part III, yielding an even greater lower bound on the minimum size of the inner state.

Second lower bound: The requirement for a large inner state gets even stronger if the attacker has a large amount of output bits at his disposal. In this case, time-memory-data tradeoff attacks as described in section 5.5 have to be taken into account, as follows.

Lemma 2 *Let L be the number of output bits available to the attacker. Then the PRG cannot be secure if $n < l + \log(L)$.*

Proof: A general time-memory-data tradeoff for $w = 1$ works as follows:

- *Precomputation phase:* The attacker draws a large sample (say, $2^{l-\epsilon}$) of inner states at random from $\mathcal{S}$. For each sample state S^i, the generator is run to produce an l-bit output z^i. The tuple (z^i, S^i) is stored in a table, indexed by z^i.
- *Attack phase:* The attacker segments the known output stream into roughly L overlapping frames $\tilde{z}^j$ of l bits[3]. For each frame, he checks whether $\tilde{z}^j$ is contained in the table, and if yes, he extracts the inner state S.

By the birthday paradox, there is high probability for a collision between the set of samples z^i in the table and the set of observations $\tilde{z}^j$ in the output stream if $2^{l-\epsilon} \cdot L \approx 2^n$. Since this attack requires $2^{l-\epsilon}$ precomputations and L table lookups, it is feasible for the attacker if $n \approx l + \log(L) - \epsilon$, where ϵ is small. Note that this proof can be generalised for arbitrary values of w by using frame lengths that are multiples of w, yielding the same result. □

Again, the cipher designer cannot control n, but only the inner state size $\hat{n}$. Remembering that an attacker who is restricted to 2^l operations can read at most $L = 2^l$ output bits, the following lower bound is obtained:

[3]To be exact, there are $L - l + 1$ such frames.

Corollary 2 *If the generator produces arbitrarily large output streams, a necessary condition for a secure PRG is* $\hat{n} \geq 2l$.

11.2.2 A generic construction

We have seen that for efficient and secure stream ciphers, the inner state size $\hat{n}$ must be strictly larger than the key size l. An obvious question is: What happens if $\hat{n}$ is increased even further? It can be shown that a large inner state can be used to make up for the deficiencies of a relatively weak PRG design. To this end, a cryptographic primitive denoted as preimage resistant hash function can be used.

Definition 3 *[86] A hash function* $H_n : \{0,1\}^* \to \{0,1\}^n$ *is said to be preimage resistant if given a value* $y \in \{0,1\}^n$*, it is infeasible to find a value* $x \in \{0,1\}^*$ *such that* $H_n(x) = y$.

Constructing the stream cipher: Let $H = \{H_n \mid n \in \mathbb{N}\}$ be a family of preimage resistant hash functions $H_n : \{0,1\}^* \to \{0,1\}^n$. Let $G = \{G_n \mid n \in \mathbb{N}\}$ be a family of PRGs with $n = \hat{n}$.[4] Furthermore, let the generator be such that the mapping from state space to the first n output bits is bijective. Finally, assume that there exists a known parameter c, $0 < c < 1$, such that for any generator $G_n \in G$ and given n bits of output stream, predicting additional output bits will require at least 2^{cn} computational steps for all but $O(1)$ cases.

Given these building blocks, a stream cipher with security level l can be constructed as follows. First, n is chosen such that $c \cdot n > l$, and use G_n as PRG. The n bits of inner state for generator G_n are initialised using the matching hash function $H_n : \{0,1\}^l \times \{0,1\}^m \to \{0,1\}^n$.

Security against key reconstruction: It can be shown that such a stream cipher is secure against inversion attacks, as long as no assumption about G_n and H_n is violated.

Lemma 3 *If for the stream cipher* (G_n, H_n)*, the key can be reconstructed in less than* 2^l *steps, then the hash function* H_n *can be inverted in less than* $2^l + n$ *steps.*

Proof: Assume that there exists an attacker A who, given the description of (G_n, H_n) and at least n bit of cipher output z, can reconstruct the key in less than 2^l steps. Then an inverter A' can be constructed who, given a valid output y of the hash function H_n, finds a corresponding input x such that $H_n(x) = y$.

- A' runs the PRG on inner state representation y, producing n bit of cipher output $z = G_n(y)$.

[4] Many PRGs are of that kind, e.g., most generators based on the principles described in section 3.3.

- A' invokes A with input z and obtains a key k with $G_n(H_n(k)) = z$.
- A' outputs k.

Note that k meets the condition $G_n(H_n(k)) = z$. Since G_n is injective, there exists only one intermediate value y with $G_n(y) = z$, implying that $H_n(k) = y$. Thus, A' has inverted the hash function, using $2^l + n$ computational steps. □

Security against prediction: Analogously, it can be shown that the stream cipher is secure against prediction attacks, as long as the output of PRG G_n cannot be predicted in less than 2^{cn} computational steps in all but a small number of cases.

Lemma 4 *Let H_n such that for all $x, y \in \{0,1\}^l$, it holds that $H_n(x) \neq H_n(y)$. Then if the stream cipher (G_n, H_n) can be predicted with probabilty p in less than 2^l steps, then the PRG G_n can be predicted with the same probabilty p in less than 2^l steps in at least 2^l out of 2^n cases.*

Proof: Assume that there exists an attacker A who, given the description of (G_n, H_n) and output bits $(z_0, \dots, z_{n-1})$, can predict output bits $(z_n, \dots, z_{n+d-1})$ correctly in less than 2^l steps. Then a trivial predictor A' can be constructed who, given a valid output $(z_0, \dots, z_{n-1})$ of G_n, can predict the subsequent output bits $(z_n, \dots, z_{n+d-1})$ in at least 2^l different cases.

- A' runs A on input $(z_0, \dots, z_{n-1})$ and obtains bits $(z_n, \dots, z_{n+d-1})$.
- A' outputs $(z_n, \dots, z_{n+d-1})$.

Note that due to the injectivity of G_n, $(z_0, \dots, z_{n-1})$ was generated from a unique starting state S_0. For the analysis, we have to distinguish two cases:

(a) If S_0 is a possible output of H_n, the sequence $(z_0, \dots, z_{n-1})$ is a correct output of the stream cipher (G_n, H_n). Thus, if A predicts correctly for the stream cipher, A' predicts correctly for the generator.

(b) If, however, no key k exists such that $H_n(k) = S_0$, the behaviour of A (and thus of A') is undefined - the prediction may or may not be correct.

In any case, the running time of A' is identical to the running time of A, yielding an effort of less than 2^l steps. Note that the algorithm is always right if case (a) occurs, yielding a correct prediction in at least 2^l (out of 2^n) cases. □

11.3 Disadvantages

In fact, practical stream ciphers often use a relatively weak PRG and rely on the inner state size and the initialisation function for security. Since constructing a cipher in the above way is tempting, why not use it as a general design rule?

With all their advantages as demonstrated in section 11.2, large inner states also have a number of drawbacks:

1. Memory is not for free. While on a modern PC, sufficient memory should be available for all reasonable PRG designs, other platforms like encryption hardware, smartcards, sensor networks, or RFID transponders may require a more economical use of resources.

2. Cryptographic memory must be protected from observation (both on general purpose and specialised hardware). Thus, an increase in memory size increases the options of an attacker, e.g., for side-channel attacks (see [69, 70] and subsequent work).

3. As mentioned in section 10.1, most stream ciphers are frequently re-initialised. This can be due to synchronisation problems, but also for cryptographical reasons.[5] Thus, initialisation should be fast, which gets increasingly difficult with growing inner state size.

4. On the other hand, initialisation should be secure, i.e., a good mixing of key and nonce into the starting state should be obtained. This, too, is difficult to obtain if the inner state is large.[6]

As a consequence, the inner state size should be as large as necessary (see section 11.2), but at the same time as small as possible. To this end, in the following chapter, a new measure of security will be introduced, and some PRG-based stream ciphers will be surveyed with respect to the inner state size required to obtain practical security.

[5] Remember that once the number of subsequent output bits available to the attacker gets large, most PRGs become vulnerable to a wide range of cryptanalytic techniques, like time-memory-data tradeoffs, correlation attacks, or algebraic attacks.

[6] This line of research was pointed out by W. Meier [80].

Chapter 12

Efficient Inner State Size

12.1 Definition

In chapter 11, it was shown that the inner state size should be

a) large enough to guarantee security of keystream generation, but

b) small enough to allow for efficient and secure initialisation.

It was also discussed that for a generator to be secure independently of the number of output bits produced, an inner state size of $2l$ bit is necessary if a security level of l bit is to be obtained. As with all lower bounds, however, this does not prove that secure generators of inner state size $2l$ can actually be constructed. On the other hand, no keystream generator of size $\hat{n} < 2^l$ has been proven to be secure against a system-theoretic attacker. Thus, there is no upper bound on the necessary inner state size.

We can, however, survey a number of proposed PRGs and known attacks, giving us an indication of what inner state sizes have led to what security levels. Note that practical generators tend to be designed for fixed key lengths and inner state sizes. For the same reason, the computational effort for the attacks is a fixed value instead of being a function in l. Thus, in order to make different generators comparable, a security measurement is required that is independent of the actual key and inner state size.

Definition 4 *Let G be a PRG, and let A be the best known attack against G. The efficient inner state size of G is a number $\sigma \in \mathbb{R}$ such that executing A takes as many computational steps as a brute force search over 2^σ starting states of G.*[1] *The quotient $\gamma = \sigma/\hat{n}$ is denoted as the inner state efficiency and is a measure for the quality of the PRG G.*

[1]Note that the efficient inner state size is defined in analogy to the so-called efficient key size. An encryption system has efficient key size τ if the best known attack has a work effort equivalent to a brute force search over 2^τ keys.

12.2 Survey of fielded generators

In the following, the inner states and best known attacks for a number of PRGs are discussed. Note that as with the examples in section 11.1, no full description of the generators is presented, but a reference to the specification is given instead. Also note that as opposed to the PRGs mentioned in parts I to III of this thesis, not all generators discussed here are based on LFSRs. Finally, observe that all PRG proposals under consideration are at least 3 years old in order to allow some time for cryptanalysis.

For most generators, the inner state can be subdivided into a linear part (i.e., the update function is linear), a nonlinear part, and key-dependent S-boxes which may or may not be bijective.[2]

A5/1: This stream cipher is part of the GSM mobile phone standard, its details were reverse engineered by Briceno et al. [15]. As can be seen from the description in section 9.3, its inner state consists of LFSRs with a total length of 64 bit. Since all initial assignments to these registers are possible, the inner state size is indeed 64 bit.
Numerous attacks have been proposed against the full A5/1 stream cipher, all taking into account that in practice, only a small number of output bits is available to the attacker [48, 12, 10, 97, 128]. If, however, an arbitrary amount of output bits is available, the generic time-memory-tradeoff attack as described in section 5.5 is most efficient, yielding $\sigma = 32$.

E_0: This cipher encrypts data in the Bluetooth communication standard [14]. Its inner state is composed of a 128 bit linear part and an additional 4 bit of memory to destroy the linearity.
Currently, an algebraic attack proposed by Courtois [24] using equations developed by Armknecht and Krause [1] is the most efficient method of cryptanalysis published, requiring roughly 2^{49} computational steps. Note, however, that the attack requires more consecutive output bits than the cipher produces between two re-initialisations. Thus, while it successfully attacks the PRG, it does not endanger the security of the Bluetooth standard itself.

Leviathan: This cipher was proposed by McGrew and Fluhrer [79] as a contribution to the NESSIE competition.[3] Its inner state consists of a 48-bit counter and 4 permutation tables over $\{0, 1\}^8$. Thus, the overall inner state size is $48 + 4 \cdot 1,684 = 6,784$ bit.

[2] A substitution box (or S-box) in cryptography implements a nonlinear mapping. For an introduction on the design of S-boxes, cf., e.g., [28].

[3] NESSIE stands for "New European Schemes for Signatures, Integrity, and Encryption" and was a EU-funded project with the objective of finding strong cryptographic algorithms. The NESSIE competition started in 2000 and ended in 2003 with the announcement of the accepted candidates. In the section "Stream Ciphers", no candidate was selected as satisfying both security and efficiency conditions [91].

The best known attack against Leviathan is a distinguisher by Crowley and Lucks [26], requiring 2^{39} bits of output and a similar work effort.

LILI-128: This pseudorandom generator designed by Dawson et al. [29] was also a NESSIE submission. Its inner state consists of two independent linear states of sizes 39 and 89 bit, respectively, yielding a total inner state size of 128 bit.
Given the construction of the cipher, a security level of 128 bit (the security goal for the NESSIE project) was not achievable in the first place due to lemma 1. As a consequence, a number of attacks on LILI-128 have been published, the most efficient one being a specialised time-memory attack by Saarinen [105] that requires roughly 2^{48} computational steps. Note that an attack proposed by Courtois in [24] formally requires less computational steps, but needs 2^{60} output bits.

RC4 (8-bit version): Officially, the stream cipher RC4 designed by Rivest is still a trade secret. Nonetheless, the design presented in [75] is widely believed to be identical to RC4. As described in section 11.1, its inner state consists of two 8-bit state variables and a table that implements a permutation $\{0,1\}^8$ that changes over time. Normally, this would yield an inner state size of $16 + 1,684 = 1,700$ bit. However, the starting values for the state variables are key-independent, and it was shown by Finney [35] that a fraction of 1/256 states can never be reached. Experiments on smaller versions of RC4 seem to indicate that the fraction of non-reachable states is even larger but still small enough that $1,700$ is a good approximation of the inner state size.
Numerous attacks against RC4 have been described. A particularly strong attack against its PRG was proposed by Golić [47] and improved by Fluhrer and McGrew [37]. The attack is a distinguisher that requires $2^{30.6}$ output bits and a similar amount of work.

Seal 3.0: As discussed in section 11.1, the generator proposed by Rogaway and Coppersmith [98] uses a 12 bit counter, 8 32-bit state words, and a set of lookup tables consisting of 1024 32-bit words, contributing up to $32,768$ bit to the inner state. Thus, the inner state size of the generator is $33,036$ bit.
While the state words are re-initialised every $2^6 \cdot 2^7 = 2^{13}$ output bits, the tables are re-initialised once every 2^{19} output bits. Thus, SEAL has two initialisation functions h_1 and h_2, and can be considered as a stream cipher $(H, G) = ((h_1, h_2), g)$. Note that the best known attack - a distinguisher by Fluhrer [36] that requires rougly 2^{43} computational steps - is only applicable if (h_2, g) is considered as the PRG.[4]

[4] Note that the inner generator g works in the way of a a one-time pad: It masks the state words using words from the lookup tables, none of which is used more than once. Thus, only by observing the working of $h2$, the generator can be attacked, otherwise, it would be secure in an information-theoretical sense.

Generator		$l_{\max}$	$\hat{n}$	σ		γ
A5/1	[15]	64	64	32.0		0.5000
Lili-128	[29]	128	128	48.0		0.3750
E_0	[14]	128	132	49.0		0.3712
Sober-t32	[59]	256	576	158.0	D	0.2743
SNOW 1.0	[30]	256	576	100.0	D	0.1736
RC4 (8bit)	[75]	256	1,700	30.6	D	0.0180
Leviathan	[79]	256	6,784	39.0	D	0.0057
Seal 3.0	[98]	160	33,036	43.0	D	0.0013

Table 12.1: Pseudorandom generators of fielded stream ciphers

Snow 1.0: This cipher is another NESSIE contribution, designed by Ekdahl and Johansson [30]. Its linear part contributes 16 32-bit words to the inner state, while the nonlinear part adds another 2 32-bit words, yielding a total inner state size of 576 bit.
Amongst the attacks proposed against Snow 1.0, the most efficient is a distinguisher by Coppersmith et al. [21], requiring about 2^{100} computational steps.

Sober-t32: The Sober family of stream ciphers by Rose and Hawkes has a long genealogy, this particular candidate being another NESSIE submission [59]. As pointed out in section 11.1, the inner state consists of 17 32-bit words and a 32-bit constant. Thus, the inner state size is 576 bit.
The most efficient attack against full Sober-t32 is a distinguisher presented by Babbage et al. [3], requiring $2^{153+5} = 2^{158}$ output bits and a similar work effort.

12.3 Comparison of results

From the time-memory-data tradeoff presented in subsection 11.2.1. it follows that the efficient inner state size of all generators is restricted by $\hat{n}/2$. Thus, we have $0 \leq \gamma \leq 0.5$ for all generators. For the generators discussed in the last section, table 12.1 compares the inner state size and inner state efficiency. We denote by $l_{\max}$ the maximum key length of the overall stream cipher and by σ the most efficient attack published against the PRG only. Distinguishing attacks are marked by a 'D'.

For a number of reasons, such a comparison has to be taken with a grain of salt, since

- running time estimates of attacks only give a rough indication of the work effort actually involved,
- older ciphers have been analysed for a longer time,
- famous ciphers like RC4 have received more cryptanalytic attention than less well-known designs,

- some attacks reconstruct the seed while others are only distinguishers, and
- some attacks require large amounts of output bits while others don't.

Nonetheless, it can be observed that the stream ciphers with particularly large internal states have very low inner state efficiencies. But even if comparison is restricted to those generators that have only distinguishing attacks standing against them, ciphers with large inner states do not seem to enjoy a real advantage over ciphers with small values for $\hat{n}$.

From this comparison, it seems reasonable to assume that values of $\gamma > 0.1$ should be achievable for practical pseudorandom generators. Note that more recent versions of Sober [58] and SNOW [31] exist that correct previous problems and have not been successfully attacked thus far. Thus, there is hope that the inner state efficiency can be brought up very close to the boundary of $\gamma = 0.5$, even for practical stream ciphers. A formal proof of this conjecture, however, seems to be beyond the current state of the art in cryptographic research.

Chapter 13

Conclusion

In modern cryptography, a number of elementary building blocks like block ciphers, stream ciphers, or hash functions are used. Stream ciphers are often based on pseudorandom generators (PRGs) that are used to transform a small initial value into a long sequence of seemingly random bits. Many PRG designs are in turn based on linear feedback shift registers (LFSRs), which can be constructed in such a way as to have optimal statistical and periodical properties.

In order to understand the security needs of a cryptographic building block, it is unavoidable to delve into cryptanalysis, which is the activity of searching for security weaknesses of cryptographic algorithms. The underlying goal of cryptanalysis is not destructive, but constructive: Only by improving the understanding of possible problems, it is possible to propose new design criteria for cryptographic systems. Thus, this thesis discussed both construction principles and cryptanalytic attacks against LFSR-based PRGs.

In part I, we introduced the basic notions and concepts required for the subsequent chapters. In particular, we introduced formal models for the encryption system and the attacker, and we gave a definition of when the system can be considered secure. We also discussed the use of pseudorandom generators in cryptography and their construction from LFSRs.

In part II, we surveyed the state of the art in cryptanalysis of pseudorandom generators. We described techniques both against unknown designs and against generators whose specifications are known to the attacker. We updated earlier surveys (e.g. [103]) by discussing new attacks and by adding examples and resource estimates.

In part III, we gave an in-depth discussion of backtracking attacks, a particular cryptanalytic technique applicable against LFSR-based pseudorandom generators. After giving a general introduction to the basic method denoted as dynamic linear consistency test, its potential was demonstrated against the self-shrinking generator, and an upper bound on the running time was proven and experimentally verified. A variant of the attack was successfully applied against a whole class of clock-controlled generators and again, an upper bound on the security of such generators was proven mathematically and confirmed in

a trial implementation.

In part IV, we analysed the necessary inner state size for pseudorandom generators to be deployed in encryption algorithms. After introducing the necessary terminology, the inner state size was formally defined and its advantages and disadvantages were highlighted. In particular, lower bounds on the necessary size were are obtained, and the security potential of increasing the inner state size was demonstrated. While proving a formal upper bound is beyond the current state of cryptographic research, a survey of fielded pseudorandom generators was given, leading to the conclusion that in practice, secure generators with inner state sizes very close to the theoretical lower bounds should be obtainable.

Bibliography

[1] F. Armknecht and M. Krause. Algebraic attacks on combiners with memory. In D. Boneh, editor, *Proc. Crypto 2003*, volume 2729 of *LNCS*, pages 162–175. Springer, 2003.

[2] S. Babbage. Cryptanalysis of LILI-128. Technical report, Nessie project, 2001. `https://www.cosic.esat.kuleuven.ac.be/nessie/reports/`.

[3] S. Babbage, C. De Cannière, J. Lano, B. Preneel, and J. Vandewalle. Cryptanalysis of Sober-t32. In T. Johansson, editor, *Proc. Fast Software Encryption 2003*, volume 2887 of *LNCS*, pages 111–128. Springer, 2003.

[4] H. Beker and F. Piper. *Cipher Systems - The Protection of Communications.* Northwood Books, London, 1982.

[5] M. Bellare. Practice-oriented provable security. In I. Damgård, editor, *Lectures on Data Security*, volume 1561 of *LNCS*, pages 1–15. Springer, 1999.

[6] M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. Stinson, editor, *Proc. Crypto '93*, volume 773 of *LNCS*, pages 232–249. Springer, 1993.

[7] E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, IT-24(3):384–386, May 1978.

[8] T. Beth and F. Piper. The stop-and-go generator. In T. Beth, N. Cot, and I. Ingemarsson, editors, *Proc. Eurocrypt '84*, volume 209 of *LNCS*, pages 88–92. Springer, 1985.

[9] E. Biham. New types of cryptanalytic attacks using related keys. In T. Helleseth, editor, *Proc. Eurocrypt '93*, volume 765 of *LNCS*, pages 398–409. Springer, 1993.

[10] Eli Biham and Orr Dunkelman. Cryptanalysis of the A5/1 GSM stream cipher. In B. K. Roy and E. Okamoto, editors, *Proc. Indocrypt 2000*, pages 43–51, 2000.

[11] A. Biryukov and A. Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In T. Okamoto, editor, *Proc. Asiacrypt 2000*, volume 1976 of *LNCS*, pages 1–13. Springer, 2000.

[12] A. Biryukov, A. Shamir, and D. Wagner. Real time cryptanalysis of A5/1 on a PC. In B. Schneier, editor, *Proc. Fast Software Encryption 2000*, volume 1978 of *LNCS*, pages 1–18. Springer, 2001.

[13] S.R. Blackburn. The linear complexity of the self-shrinking generator. *IEEE Transactions on Information Theory*, 45(6):2073–2077, September 1999.

[14] *Bluetooth Specification v1.1*, 1999. www.bluetooth.com.

[15] M. Briceno, I. Goldberg, and D. Wagner. A pedagogical implementation of A5/1.
`http://www.scard.org/gsm/a51.html`.

[16] A. Canteaut and M. Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In B. Preneel, editor, *Proc. Eurocrypt 2000*, volume 1807 of *LNCS*, pages 573–588. Springer, 2000.

[17] V. Chepyzhov, T. Johansson, and B. Smeets. A simple algorithm for fast correlation attacks on stream ciphers. In B. Schneier, editor, *Proc. Fast Software Encryption 2000*, volume 1978 of *LNCS*, pages 181–195. Springer, 2000.

[18] V. Chepyzhov and B. Smeets. On a fast correlation attack on certain stream ciphers. In D. Davies, editor, *Proc. Eurocrypt '91*, volume 547 of *LNCS*, pages 176–185. Springer, 1991.

[19] P. Chose, A. Joux, and M. Mitton. Fast correlation attacks: An algorithmic point of view. In L. Knudsen, editor, *Proc. Eurocrypt 2002*, volume 2332 of *LNCS*, pages 209–221. Springer, 2002.

[20] A. Clark, E. Dawson, J. Fuller, H.-J. Lee J. Dj. Golić, W. Millan, S.-J. Moon, and L. Simpson. The LILI-II keystream generator. In L. Batten and J. Seberry, editors, *Proc. ACISP 2002*, volume 2384 of *LNCS*, pages 25–39. Springer, 2002.

[21] D. Coppersmith, S. Halevi, and C. Jutla. Cryptanalysis of stream ciphers with linear masking. In M. Yung, editor, *Proc. Crypto 2002*, volume 2442 of *LNCS*, pages 515–532. Springer, 2002.

[22] D. Coppersmith, H. Krawczyk, and Y. Mansour. The shrinking generator. In D.R. Stinson, editor, *Advances in Cryptology - Eurocrypt '93*, volume 773 of *LNCS*, pages 22–39. Springer, 1993.

[23] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms.* MIT press, Cambridge (Mass.), 2nd edition, 2001.

[24] N. Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In D. Boneh, editor, *Proc. Crypto 2003*, volume 2729 of *LNCS*, pages 176–194. Springer, 2003.

[25] N. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In B. Preneel, editor, *Proc. Eurocrypt 2000*, volume 1807 of *LNCS*, pages 392–407. Springer, 2000.

[26] P. Crowley and S. Lucks. Bias in the LEVIATHAN stream cipher. In M. Matsui, editor, *Proc. Fast Software Encryption 2001*, volume 2355 of *LNCS*, pages 211–218. Springer, 2002.

[27] J. Daemen, R. Govaerts, and J. Vandewalle. Resynchronisation weakness in synchronous stream ciphers. In T. Helleseth, editor, *Proc. Eurocrypt '93*, volume 765 of *LNCS*, pages 159–167. Springer, 1994.

[28] J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer, Berlin, 2002.

[29] E. Dawson, A. Clark, J. Golić, W. Millan, L. Penna, and L. Simpson. The LILI-128 keystream generator. `http://www.isrc.qut.edu.au/resource/lili/lili_nessie_workshop.pdf`.

[30] P. Ekdahl and T. Johansson. SNOW - a new stream cipher. `http://www.it.lth.se/cryptology/snow/`. NESSIE project submission.

[31] P. Ekdahl and T. Johansson. A new version of the stream cipher SNOW. In H. Heys and K. Nyberg, editors, *Proc. SAC 2002*, volume 2595 of *LNCS*, pages 47–61. Springer, 2002.

[32] P. Ekdahl and T. Johansson. Another attack on A5/1. *IEEE Trans. Information Theory*, 49(1):284–289, 2003.

[33] N. Ferguson and B. Schneier. *Practical Cryptography*. Wiley, 2003.

[34] E. Filiol. Decimation attack of stream ciphers. In B. Roy and E. Okamoto, editors, *Proc. Indocrypt 2000*, volume 1977 of *LNCS*, pages 31–42. Springer, 2000.

[35] H. Finney. An RC4 cycle that can't happen. Newsgroup post to sci.crypt, September 1994.

[36] S. Fluhrer. Cryptanalysis of the SEAL 3.0 pseudorandom function family. In M. Matsui, editor, *Proc. Fast Software Encryption 2001*, volume 2355 of *LNCS*, pages 135–143. Springer, 2002.

[37] S. Fluhrer and D. McGrew. Statistical analysis of the alleged RC4 keystream generator. In B. Schneier, editor, *Proc. Fast Software Encryption 2000*, volume 1978 of *LNCS*, pages 19–30. Springer, 2001.

[38] R. Gallager. *Low-density Parity Check Codes.* MIT Press, Cambridge, 1963.

[39] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.

[40] P. Geffe. How to protect data with ciphers that are really hard to break. *Electronics*, 46(1):99–101, Jan. 1973.

[41] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness.* Springer, 1999.

[42] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[43] J. Golić. Correlation via linear sequential circuit approximation of combiners with memory. In R. Rueppel, editor, *Proc. Eurocrypt '92*, volume 658 of *LNCS*, pages 113–123. Springer, 1993.

[44] J. Golić. Towards fast correlation attacks on irregularly clocked shift registers. In L. Guillou and J.-J. Quisquater, editors, *Proc. Eurocrypt '95*, volume 921 of *LNCS*, pages 248–262. Springer, 1995.

[45] J. Golić. Correlation properties of a general binary combiner with memory. *Journal of Cryptology*, 9(2):111–126, 1996.

[46] J. Golić. Cryptanalysis of alleged A5 stream cipher. In W. Fumy, editor, *Proc. Eurocrypt '97*, volume 1233 of *LNCS*, pages 239–255. Springer, 1997.

[47] J. Golić. Linear statistical weakness of alleged RC4 keystream generator. In W. Fumy, editor, *Proc. Eurocrypt '97*, volume 1233 of *LNCS*, pages 226–238. Springer, 1997.

[48] J. Golić. Cryptanalysis of three mutually clock-controlled stop/go shift registers. *IEEE Trans. Inf. Theory*, 46(3):1081–1090, May 2000.

[49] J. Golić. Correlation analysis of the shrinking generator. In J. Kilian, editor, *Proc. Crypto 2001*, volume 2139 of *LNCS*, pages 440–457. Springer, 2001.

[50] J. Golić and M. Mihaljević. A noisy clock-controlled shift register cryptanalytic concept based on sequence comparison approach. In I. Damgard, editor, *Proc. Eurocrypt '90*, volume 473 of *LNCS*, pages 487–491. Springer, 1990.

[51] J. Golić and M. Mihaljević. A generalized correlation attack on a class of stream ciphers based on the Levenshtein distance. *Journal of Cryptology*, 3(3):201–212, 1991.

[52] J. Golić and G. Morgari. On the resynchronization attack. In T. Johansson, editor, *Proc. Fast Software Encryption 2003*, volume 2887 of *LNCS*, pages 100–110. Springer, 2003.

[53] J. Golić and L. O'Connor. Embedding and probabilistic attacks on clock-controlled shift registers. In A. De Santis, editor, *Proc. Eurocrypto '94*, volume 950 of *LNCS*, pages 230–243, Berlin, 1995. Springer.

[54] J. Golić and S. Petrović. A generalized correlation attack with a probabilistic constrained edit distance. In R. Rueppel, editor, *Proc. Eurocrypt '92*, volume 658 of *LNCS*, pages 472–476. Springer, 1993.

[55] D. Gollmann and W. Chambers. Clock-controlled shift registers: A review. *IEEE J. Selected Areas Comm.*, 7(4):525–533, May 1989.

[56] S. Golomb. *Shift Register Sequences.* Aegean Park Press, Laguna Hills (CA), revised edition, 1982.

[57] C. Günther. Alternating step generators controlled by de Bruijn sequences. In D. Chaum and W. Price, editors, *Proc. Eurocrypt '87*, volume 304 of *LNCS*, pages 88–92. Springer, 1988.

[58] P. Hawkes and G. Rose. Primitive specification for Sober-128. `http://www.qualcomm.com.au/Sober128.html`.

[59] P. Hawkes and G. Rose. Primitive specification and supporting documentation for Sober-t32. NESSIE project submission, October 2000.

[60] S. Jiang and G. Gong. Cryptanalysis of stream cipher - a survey. Technical Report CORR-2002-29, University of Waterloo, 2002.

[61] T. Johansson and F. Jönsson. Fast correlation attacks based on Turbo Code techniques. In M. Wiener, editor, *Proc. Crypto '99*, volume 1666 of *LNCS*, pages 181–197. Springer, 1999.

[62] T. Johansson and F. Jönsson. Improved fast correlation attacks on stream ciphers via convolutional codes. In J. Stern, editor, *Proc. Eurocrypt '99*, volume 1592 of *LNCS*, pages 347–362. Springer, 1999.

[63] T. Johansson and F. Jönsson. Fast correlation attacks through reconstruction of linear polynomials. In M. Bellare, editor, *Proc. Crypto 2000*, volume 1880 of *LNCS*, pages 300–315. Springer, 2000.

[64] T. Johansson and F. Jönsson. Theoretical analysis of a correlation attack based on convolutional codes. In *Proc. International Symposium on Information Theory*, Sorrento, Italy, June 2000.

[65] F. Jönsson and T. Johansson. A fast correlation attack on LILI-128. Technical report, Lund University, Sweden, 2001.

[66] A. Kerckhoffs. La cryptographie militaire. *Journal des Sciences Militaires*, pages 161–191, Feb. 1883.

[67] A. Kipnis and A. Shamir. Cryptanalysis of the HFE public key cryptosystem by relinearization. In M. Wiener, editor, *Proc. Crypto '99*, volume 1666 of *LNCS*, pages 19–30. Springer, 1999.

[68] D. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*. Addison-Wesley, 2 edition, 1981.

[69] P. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Proc. Crypto '96*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.

[70] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. Wiener, editor, *Proc. Crypto '99*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.

[71] M. Krause. BDD-based cryptanalysis of keystream generators. In L. Knudsen, editor, *Proc. Eurocrypt '02*, LNCS. Springer, 2002.

[72] X. Lai and J. Massey. A proposal for a new block encryption standard. In I. Damgard, editor, *Proc. Eurocrypt '90*, volume 473 of *LNCS*, pages 389–404, Berlin, 1991. Springer.

[73] R. Lidl and H. Niederreiter. *Introduction to finite fields and their application*. Cambridge University Press, Cambridge (UK), revised edition, 1994.

[74] S. Lucks and E. Zenner. Kryptographie - eine Geheimwissenschaft wird öffentlich. *Forschung Universität Mannheim - FoRUM 2003*, pages 14–17, 2003.

[75] Itsik Mantin. Analysis of the stream cipher RC4. Master's thesis, Weizmann Institute of Science, Rehovot, Israel, November 2001.

[76] J. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, IT-15:122–127, Jan. 1969.

[77] U. Maurer. A universal statistical test for random bit generators. In A. Menezes and S. Vanstone, editors, *Proc. Crypto 1990*, volume 537 of *LNCS*, pages 409–420. Springer, 1991.

[78] U. Maurer. Conditionally-perfect secrecy and a provably-secure randomized cipher. *Journal of Cryptology*, 5(1):53–66, 1992.

[79] D. McGrew and S. Fluhrer. The stream cipher Leviathan. NESSIE project submission, October 2000.

[80] W. Meier. personal communication, August 2003.

[81] W. Meier and O. Staffelbach. Fast correlation attacks on stream ciphers. In C. Günther, editor, *Proc. Eurocrypt '88*, volume 330 of *LNCS*, pages 301–314. Springer, 1988.

[82] W. Meier and O. Staffelbach. Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1(3):159–167, 1989.

[83] W. Meier and O. Staffelbach. Nonlinearity criteria for cryptographic functions. In J.-J. Quisquater and J. Vandewalle, editors, *Proc. Eurocrypt '89*, volume 434 of *LNCS*, pages 549–562. Springer, 1990.

[84] W. Meier and O. Staffelbach. Correlation properties of combiners with memory in stream ciphers. *Journal of Cryptology*, 5(1):67–86, 1992.

[85] W. Meier and O. Staffelbach. The self-shrinking generator. In A. De Santis, editor, *Proc. Eurocrypt '94*, volume 950 of *LNCS*, pages 205–214, Berlin, 1995. Springer.

[86] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1997.

[87] M. Mihaljević, M. Fossorier, and H. Imai. A low-complexity and high-performance algorithm for the fast correlation attack. In B. Schneier, editor, *Proc. Fast Software Encryption 2000*, volume 1978 of *LNCS*, pages 196–212. Springer, 2001.

[88] M. Mihaljević, M. Fossorier, and H. Imai. Fast correlation attack algorithm with list decoding and an application. In M. Matsui, editor, *Proc. Fast Software Encryption 2001*, volume 2355 of *LNCS*, pages 196–210. Springer, 2002.

[89] M. Mihaljević and J. Dj. Golić. A fast iterative algorithm for a shift register initial state reconstruction given the noisy output sequence. In J. Seberry and J. Pieprzyk, editors, *Proc. Auscrypt '90*, volume 453 of *LNCS*, pages 165–175. Springer, 1990.

[90] M.J. Mihaljević. A faster cryptanalysis of the self-shrinking generator. In J. Pieprzyk and J. Seberry, editors, *Proc. ACISP '96*, volume 1172 of *LNCS*, pages 182–189, Berlin, 1996. Springer.

[91] New european schemes for signatures, integrity, and encryption (NESSIE). `https://www.cosic.esat.kuleuven.ac.be/nessie/`.

[92] National Institute of Standards and Technology. Data Encryption Standard (DES). FIPS PUB 46-3, October 1999.

[93] National Institute of Standards and Technology. Advanced Encryption Standard (AES). FIPS PUB 197, November 2001.

[94] National Institute of Standards and Technology. Security requirements for cryptographic modules. FIPS PUB 140-2, December 2002.

[95] W. Penzhorn. Correlation attacks on stream ciphers: Computing low-weight parity checks based on error-correcting codes. In D. Gollmann, editor, *Proc. Fast Software Encryption '96*, volume 1039 of *LNCS*, pages 159–172. Springer, 1996.

[96] V. Pless. *Introduction to the Theory of Error-Correcting Codes.* John Wiley, 1982.

[97] T. Pornin and J. Stern. Software-hardware trade-offs: Application to A5/1 cryptanalysis. In Ç. Koç and C. Paar, editors, *Proc. CHES 2000*, volume 1965 of *LNCS*, pages 318–327. Springer, 2000.

[98] P. Rogaway and D. Coppersmith. A software-optimized encryption algorithm. *Journal of Cryptology*, 11(4):273–287, Fall 1998.

[99] G. Rose and P. Hawkes. On the applicability of distinguishing attacks against stream ciphers. 3rd NESSIE workshop, available at `http://www.qualcomm.com.au/PublicationsDocs/StreamAttack.pdf`, Nov. 2002.

[100] R. Rueppel. *Analysis and Design of Stream Ciphers.* Springer, 1986.

[101] R. Rueppel. Correlation immunity and the summation generator. In H. Williams, editor, *Proc. Crypto '85*, volume 218 of *LNCS*, pages 260–272. Springer, 1986.

[102] R. Rueppel. Security models and notions for stream ciphers. In H. Beker and F. Piper, editors, *Proc. 2nd IMA Conf. on Cryptography and Coding*, pages 213–230. Oxford University Press, 1989.

[103] R. Rueppel. Stream ciphers. In G. Simmons, editor, *Contemporary Cryptology - The Science of Information Integrity*, pages 65–134. IEEE Press, 1992.

[104] R. Rueppel and O. Staffelbach. Products of sequences with maximum linear complexity. *IEEE Transactions on Information Theory*, IT-33(1):124–131, 1987.

[105] M.-J. Saarinen. A time-memory tradeoff attack against LILI-128. In J. Daemen and V. Rijmen, editors, *Proc. Fast Software Encryption 2002*, volume 2365 of *LNCS*, pages 231–236. Springer, 2002.

[106] S. Sarma, S. Weis, and D. Engels. Radio-frequency identification: Security risks and challenges. *RSA Laboratories Cryptobytes*, 6(1):2–9, 2003.

[107] F. Schleer. Einsatz von OBDDs zur Kryptanalyse von Flusschiffren. Master's thesis, University of Mannheim, 2002.

[108] M. Schmidt. Blauzahnlücken. *C'T Magazin für Computertechnik 11/03*, pages 186–189, 2003.

[109] C. Shannon. A mathematical theory of communication. *Bell Systems Techn. Journal*, 27:623–656, 1948.

[110] C. Shannon. Communication theory of secrecy systems. *Bell Systems Techn. Journal*, 28:656–715, 1949.

[111] T. Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Transactions on Information Theory*, IT-30(5):776–780, 1984.

[112] T. Siegenthaler. Decrypting a class of stream ciphers using ciphertext only. *IEEE Transactions on Information Theory*, C-34(1):81–85, January 1985.

[113] L. Simpson, E. Dawson, J. Golić, and W. Millan. LILI keystream generator. In D. Stinson and S. Tavares, editors, *Proc. SAC 2000*, volume 2012 of *LNCS*, pages 248–261. Springer, 2001.

[114] L. Simpson, J. Golić, and E. Dawson. A probabilistic correlation attack on the shrinking generator. In C. Boyd and E. Dawson, editors, *Proc. ACISP '98*, volume 1438 of *LNCS*, pages 147–158, Berlin, 1998. Springer.

[115] D. Stegemann. FBDD-basierte Kryptanalyse des A5/1-Schlüsselstromgenerators. Master's thesis, University of Mannheim, 2004.

[116] A. Stubblefield, J. Ioannidis, and A. Rubin. Using the Fluhrer, Mantin and Shamir attack to break WEP. Technical Report TD-4ZCPZZ, AT&T labs, August 2001.

[117] J. van Lint. *Introduction to Coding Theory*. Springer, 1982.

[118] G. Vernam. Cipher printing telegraph system for secret wire and radio telegraphic communications. *Journal of American Institute of Electrical Engineers*, 45:109–115, 1926.

[119] I. Wegener. *Branching Programs and Binary Decision Diagrams*. Monographs on Discrete Mathematics and Applications. SIAM, 2000.

[120] D. Wheeler and R. Needham. TEA - a tiny encryption algorithm. Technical report, Computer Laboratory, University of Cambridge, 1995.

[121] D. Wheeler and R. Needham. TEA extensions. Technical report, Computer Laboratory, University of Cambridge, 1997.

[122] G. Xiao and J. Massey. A spectral characterization of correlation-immune combining functions. *IEEE Transactions on Information Theory*, 34(3):569–571, May 1988.

[123] A. Yao. Theory and applications of trapdoor functions. In *Proc. 25th FOCS*, pages 80–91, 1982.

[124] K. Zeng and M. Huang. On the linear syndrome method in cryptanalysis. In S. Goldwasser, editor, *Proc. Crypto '88*, volume 403 of *LNCS*, pages 469–478. Springer, 1990.

[125] K. Zeng, C. Yang, and T. Rao. On the linear consistency test (LCT) in cryptanalysis with applications. In G. Brassard, editor, *Proc. Crypto '89*, volume 435 of *LNCS*, pages 164–174. Springer, 1990.

[126] K. Zeng, C. Yang, and T. Rao. An improved linear syndrome algorithm in cryptanalysis with applications. In A. Menezes and S. Vanstone, editors, *Proc. Crypto '90*, volume 537 of *LNCS*, pages 34–47. Springer, 1991.

[127] E. Zenner. Kryptographische Protokolle im GSM-Standard - Beschreibung und Kryptanalyse. Master's thesis, University of Mannheim, 1999.

[128] E. Zenner. On the efficiency of clock control guessing. In P. J. Lee and C. H. Lim, editors, *Proc. ICISC '02*, volume 2587 of *LNCS*, pages 200–212. Springer, 2003.

[129] E. Zenner. Cryptanalysis of LFSR-based pseudorandom generators - a survey. Technical Report Informatik TR-04-004, University of Mannheim (Germany), January 2004. available at
`http://www.informatik.uni-mannheim.de/techberichte/html/`.

[130] E. Zenner. On the role of the inner state size in stream ciphers. Technical Report Informatik TR-04-001, University of Mannheim (Germany), January 2004. available at
`http://www.informatik.uni-mannheim.de/techberichte/html/`.

[131] E. Zenner. On the role of the inner state size in stream ciphers. In E. Fernández-Medina, J. Hernández Castro, and L. García Villalba, editors, *Proc. WOSIS '04*, pages 237–250. INSTICC Press, 2004.

[132] E. Zenner, M. Krause, and S. Lucks. Improved cryptanalysis of the self-shrinking generator. In V. Varadharajan and Y. Mu, editors, *Proc. ACISP '01*, volume 2119 of *LNCS*, pages 21–35. Springer, 2001.

[133] E. Zenner, R. Weis, and S. Lucks. Sicherheit des GSM-Verschlüsselungsstandards A5. *DuD - Datenschutz und Datensicherheit*, 24(7):405–407, 2000.

[134] M. Zivković. An algorithm for the initial state reconstruction of the clock-controlled shift register. *IEEE Transactions on Information Theory*, 37(5):1488–1490, Sep. 1991.

[135] M. Zivković. On two probabilistic decoding algorithms for binary linear codes. *IEEE Transactions on Information Theory*, 37(6):1707–1716, Nov. 1991.