Cryptanalysis of Lightweight Cryptographic Algorithms

By

Mohammad Ali Orumiehchiha

A thesis submitted to Macquarie University
for the degree of
Doctor of Philosophy
Department of Computing
July 2014

MACQUARIE
UNIVERSITY
FACULTY OF SCIENCE

Typeset in $\LaTeX\,2_\varepsilon$.

This thesis is submitted in fulfilment of the requirements of the degree of Doctor of Philosophy at Macquarie University. I certify that this work has not been submitted for a higher degree to any other university or institution. To the best of my knowledge, all sources of information used in the preparation of this thesis have been acknowledged and the utilization of others works, wherever applicable, has been properly cited.

Orumiehch

Mohammad Ali Orumiehchiha

Acknowledgements

I would like to express my sincere appreciation to my supervisor Professor Josef Pieprzyk for supporting of me at many levels: financial, intellectual and personal. I particularly appreciate his patience, guidance, and motivation. I would also like to thank my co-supervisor Dr. Ron Steinfeld, and other co-authors Elham Shakour, Dr. Harry Bartlett, and Qian Yu for being great collaborators and for initiating helpful ideas.

I am thankful to the organizers of ASK 2011, ASK 2012, and ASK 2013 for inviting and enabling me to collaborate with other active crypto-groups around Australia and Asia.

I am grateful to Dr. Keith Imrie, and Dr. Wendy Noble for patiently proofreading my thesis, and giving me useful comments. I am also thankful to all the staff in the Department of Computing for their excellent support.

I would also like to express my gratitude to my wife, Elham, for her continued help and encouragement. My deepest thanks go to my parents who have continuously supported me in all of the ups and downs of my life and have encouraged me to pursue my education.

List of Publications

Journal Papers:

- M. A. Orumiehchiha, J. Pieprzyk, R. Steinfeld, *Cryptanalysis of WG7 Stream Cipher*, Journal of Cryptography and Communications, vol. 4, pp. 277-285, Springer , 2012.

- M. A. Orumiehchiha, J. Pieprzyk, R. Steinfeld, H. Bartlett, *Security Analysis of Linearly Filtered NLFSRs*, Journal of Mathematical Cryptology, vol. 7, pp. 313-332, 2013.

- M. A. Orumiehchiha, J. Pieprzyk, R. Steinfeld, *Practical Attack on NLM-MAC Scheme*, Information Processing Letters, vol. 114, pp. 547-550, 2014.

Conference Papers:

- M. A. Orumiehchiha, J. Pieprzyk, R. Steinfeld, *Cryptanalysis of Hash Function Based on RC4*, The 10th Australasian Information Security Conference (AISC 2012), Melbourne, Australia, pp. 33-38, January 2012.

- Q. Yu, C.N. Zhang, M.A. Orumiehchiha, L. Hua , *RC4-BHF: An Improved RC4-Based Hash Function*, IEEE 12th International Conference on Computer and Information Technology (CIT), China, pp. 322-326, October 2012.

- M. A. Orumiehchiha, J. Pieprzyk, E. Shakour, R. Steinfeld, *Security Evaluation of Rakaposhi Stream Cipher*, 9th International Conference Information Security Practice and Experience (ISPEC 2013), China, LNCS vol. 7863, Springer, pp. 361-371, May 2013.

- M. A. Orumiehchiha, J. Pieprzyk, E. Shakour, R. Steinfeld, *Cryptanalysis of RC4(n,m)*, the 6th International Conference on Security of Information and Networks, pp. 165-172, Turkey, November 2013.

Abstract

Stream ciphers are symmetric cipher systems which provide confidentiality in many applications ranging from mobile phone communication to virtual private networks. They may be implemented efficiently in software and hardware and are a preferred choice when dealing with resource-constrained environments, such as smart cards, RFID tags, and sensor networks. This dissertation addresses cryptanalysis of several stream ciphers, and a hash function based on stream cipher. Also, the thesis investigates the design principles and security of stream ciphers built from nonlinear feedback shift registers. In a design view, any cryptographic attack shows a weak point in the design and immediately can be converted into an appropriate design criterion. Firstly, this thesis focuses on the WG-7, a lightweight stream cipher. It is shown that the keystream generated by WG-7 can be distinguished from a random sequence with a negligible error probability. In addition, a key-recovery attack on the cipher has been successfully proposed. Then, a security evaluation of the Rakaposhi stream cipher identifies weaknesses of the cipher. The main observation shows that the initialisation procedure has a sliding property. This property can be used to launch distinguishing and key-recovery attacks. Further, the cipher is studied when the registers enter short cycles. In this case, the internal state can be recovered with less complexity than exhaustive search. New security features of a specific design based on nonlinear feedback shift registers have been explored. The idea applies a distinguishing attack on linearly filtered nonlinear feedback shift registers. The attack extends the idea on linear combinations of linearly filtered nonlinear feedback shift registers as well. The proposed attacks allow the attacker to mount linear attacks to distinguish the output of the cipher and recover its internal state. The next topic analyses a new lightweight communication framework called NLM-MAC. Several critical cryptographic weaknesses leading to key-recovery and forgery attack have been indicated. It is shown that the adversary can recover the internal state of the NLM generator. The attacker also is able to forge any MAC tag in real time. The proposed attacks are completely practical and break the scheme. Another part demonstrates some new cryptographic attacks on RC4(n,m) stream cipher. The investigations have revealed several weaknesses of the cipher. Firstly, a distinguisher for the cipher is proposed. Secondly, a key-recovery attack uses a method to find the secret key in real time. Finally, the RC4-BHF hash function that is based on the well-known RC4 stream cipher is analysed. Two attacks on RC4-BHF have been developed. In the first attack, the adversary is able to find collisions for two different messages. The second attack shows how to design a distinguisher that can tell apart the sequence generated by RC4-BHF from a random one.

Keywords: Cryptography, Cryptanalysis, Symmetric cipher, Stream cipher, Hash function, Key recovery attack, Distinguishing attack, Collision attack, Forgery attack.

Contents

Acknowledgements v

List of Publications vii

Abstract ix

List of Figures xv

List of Tables xvii

1 Introduction 1

1.1 Cryptology 1
1.2 Cryptography 2
1.2.1 Hash functions 4
1.2.2 Cryptographic requirements 4
1.2.3 Hash function design 5
1.2.4 Stream ciphers 7
1.3 Cryptanalysis 9
1.3.1 Attack models 10
1.3.2 Distinguishing attacks 10
1.3.3 Related-key attacks 11
1.3.4 Generic attacks 11
1.3.5 Exhaustive search attack 11
1.3.6 Time-Memory Trade-Off attack 12
1.4 Thesis outline 12

2 Stream Ciphers 15

2.1 Stream Cipher Design 15
2.1.1 Building Blocks 15
2.1.2 Some design techniques 19

2.2 Cryptanalysis of stream ciphers 22
2.2.1 Linear Cryptanalysis 22
2.2.2 Correlation attack 23
2.2.3 Differential Cryptanalysis 25
2.2.4 Algebraic Cryptanalysis 25

3 Cryptanalysis of WG-7 stream cipher 29
3.1 Description of WG-7 30
3.2 Cryptanalysis of WG-7 30
3.2.1 Distinguishing Attack for WG-7 31
3.2.2 Key-Recovery Attack on WG-7 34
3.3 Conclusions 36

4 Security evaluation of Rakaposhi stream cipher 39
4.1 Description of Rakaposhi Stream Cipher 40
4.1.1 Initialisation Procedure 42
4.2 Cryptanalysis of Rakaposhi Stream Cipher 42
4.2.1 Properties of Rakaposhi Cipher 42
4.2.2 Related-Key Attack on Rakaposhi 43
4.2.3 Recovery of Secret Keys 45
4.3 Weak (K, IV) Pairs 47
4.3.1 Weak (K, IV) Pairs Leading to $A = \mathbf{1}$ 47
4.3.2 Weak (K, IV) Pairs Leading to $B = \mathbf{0}$ 48
4.4 Summary 49

5 Security analysis of linearly filtered NLFSRs 53
5.1 Description of LF-NLFSR 55
5.1.1 Attacks on LF-NLFSR 55
5.1.2 Distinguishing Attack on LF-NLFSR 56
5.2 Random LF-NLFSR Ciphers 59
5.2.1 Cryptanalysis of Random LF-NLFSR Ciphers 59
5.3 Ciphers based on LF-NLFSRs and LFSRs 61
5.3.1 Distinguishing Attack on Grain [10]. 61
5.4 Ciphers based on Linear Combination of LF-NLFSRs 62
5.4.1 Distinguishing Attack on LC-NLFSRs 62
5.5 Linear Filter Properties 64
5.5.1 Some Observations on the Grain LF-NLFSR 65
5.6 Summary 69

6 Practical attack on NLM generators **71**
6.1 Description of NLM-MAC Scheme 72
6.1.1 NLM-128 Stream Cipher 72
6.1.2 The NLM-MAC Function 74
6.2 Cryptanalysis of NLM-MAC Scheme 74
6.2.1 Cryptanalysis of NLM generator 74
6.2.2 Analysis of NLM-MAC Function 76
6.2.3 Attack on NLM Scheme 76
6.3 Summary 76

7 Cryptanalysis of RC4(n,m) stream cipher **79**
7.1 Description of RC4(n,m) Stream Cipher 82
7.1.1 Notations 83
7.2 Cryptanalysis of RC4(n,m) Stream Cipher 83
7.2.1 Weaknesses of RC4(n,m) 84
7.2.2 Distinguishing Attack on RC4(n,m) 86
7.2.3 Key-Recovery Attack on RC4(n,m) 91
7.2.4 Discussion 92
7.3 Summary 93

8 Cryptanalysis of a hash function based on RC4 **95**
8.1 Description of the RC4-BHF hash function 96
8.2 Cryptanalysis of RC4-BHF 99
8.2.1 The weaknesses of RC4-BHF 100
8.2.2 Collision attack on RC4-BHF 100
8.2.3 Other Period Properties 102
8.2.4 Finding Collisions 102
8.2.5 Randomness properties of hash digest 105
8.3 Summary 106

9 Conclusion **107**
9.1 Thesis summary 107
9.2 Future research directions 109

References **111**

List of Figures

1.1 Essential security requirements to design hash functions 5
1.2 The Merkle-Damgård construction 6
1.3 The sponge construction 7
1.4 Encryption and decryption module of a synchronous stream cipher . . . 9
1.5 Encryption and decryption module of a self-synchronising stream cipher 10

2.1 A general structure of an LFSR of length L at time t=0 16
2.2 A general structure of an NLFSR of length L at time t=0 17
2.3 A general structure of a nonlinear combination generator 20
2.4 A general structure of a filter generator 20
2.5 A general structure of an alternating step generator 21
2.6 The structure of a shrinking generator 21
2.7 KSA Function 23
2.8 PRGA Function 24

3.1 The WG-7 stream cipher scheme 30

4.1 Rakaposhi stream cipher 41

5.1 Translation of LF-NLFSR into LFSR with nonlinear filter 54
5.2 7−bit LF-NLFSR cipher 56
5.3 Grain cipher 61
5.4 LC-NLFSR 63
5.5 LC-NLFSR of Example 5.4.1 64

6.1 NLM family stream cipher 73

7.1 $RC4(n, m)$ stream cipher scheme 82
7.2 KSA*: The key-scheduling algorithm of $RC4(n, m)$ 83
7.3 PRGA*: Pseudorandom generation algorithm of $RC4(n, m)$ 84

7.4 RC4(n,m): Another perspective of KSA*. According to the Low Diffusion Property, the P-th slice of the internal state in the r-th round of KSA* (r is an arbitrary round) depends on the P-th slice of the internal state and previous slices in the initial state. Also, any random difference in the P-th slice in the initial state does not change the Q-th slice in the r-th round of KSA*. 87

8.1 H-KSA* function 97
8.2 H-PRGA* function 98
8.3 RC4-BHF scheme 99

List of Tables

3.1 Experimental results for applying the distinguishing attack on WG-7 . 34
3.2 Comparison of different algebraic attacks against WG-7 36

4.1 Shifted identical keystream outputs corresponding two related (K, IV) pairs 51

5.1 Comparison of results 69

7.1 Experimental Results and Comparison between theory and simulation 89
7.2 Comparison between the previous attacks and our proposals 94

8.1 Properties and conditions to apply a collision attack on Algorithm for other cycles 103
8.2 Example for Method 1 including M_0, M_1, M_2 and generated hash value. 104
8.3 Example for Method 2 including M_0, M_1, M_2, M_3 and generated hash values. 105

1

Introduction

1.1 Cryptology

Cryptology, from the Greek words "*kryptos*", meaning "hidden" and "*-logia*", meaning "study" [94], is the study and use of secret words. It includes two major fields: cryptography and cryptanalysis. Historically, cryptography was the art of secret writing. Today, it is the science of protecting sensitive information communicated over an insecure channel. Cryptanalysis studies the methods and procedures used to compromise the security of cryptographic schemes. Cryptography deals with design of "secure" cryptographic algorithms and protocols. The security here is understood as a guarantee that a cryptographic system achieves a well defined collection of security goals. The basic collection of security goals includes confidentiality, authentication, integrity and non-repudiation.

- **Confidentiality**: Information transmitted, stored, or processed is unintelligible to all users but the owner of the information. To achieve confidentiality, one can apply encryption.
- **Authentication**: The source of information can be identified and attributed to its owner or sender. Cryptographic message authentication codes (MACs) are normally used to achieve this goal.

- **Integrity**: The receiver of information is able to ensure it has not been modified during transmission or storing. Cryptographic hash functions are typically used to achieve integrity.

- **Non-repudiation**: The receiver can confirm that the received message is exactly what the sender sent; the sender cannot deny any part of his or her participation.

All the above goals are mainly achieved by cryptographic primitives and protocols.

1.2 Cryptography

Traditionally, encryption algorithms are the heart of cryptography. They can be divided into two categories based on employing the cryptographic key:

- secret-key encryption

- public-key encryption

Secret-key encryption (also called symmetric) applies the same key for encryption and decryption. Ciphers from this category can be further classified as stream and block ciphers.

In public-key encryption, messages are encrypted using a public key while the decryption can be done only by the holder of a secret key. It is obvious that knowledge of the public key should not allow an adversary to recover the secret key.

Modern cryptography was born when C. E. Shannon published his seminal paper "*Communication Theory of Secrecy Systems*" in which he laid down the basic principles of cryptology [127]. He formulated secrecy system foundations and transformed cryptology from an art into a science.

In 1976, Diffie and Hellman published a paper in which they showed how to agree on a secret key via public discussion [46]. They also described a concept of public-key encryption. This revolutionary concept has changed the face of cryptology; public-key cryptography was born.

Shortly after that, in 1977, Rivest, Shamir and Adleman published their public-key encryption, denoted by RSA [123], based on the factorisation problem. In addition, the Merkle-Hellman [110] and McElice [106] schemes based on the knapsack and decoding problems were the first algorithms to establish public-key cryptosystems.

In the early 1970s, the need for secure communication became urgent. Financial institutions were concerned about the security of financial data transmitted via leased public communication channels. The US National Institute for Standards and Technology (NIST) identified the need and announced a call for an encryption standard.

The researchers from academia and IBM proposed an encryption algorithm that was a modification of the IBM Lucifer encryption algorithm. The algorithm was adopted as the US standard and is called the Data Encryption Standard (DES).

The DES algorithm provides strong encryption, but the key space is relatively small. This weakness was identified by Diffie in 1975. He argued that with progress in computing technology, DES may be broken by an exhaustive search of keys. This prediction was very accurate. In the early 1995s, it was evident that DES no longer provided a sufficient level of protection. NIST eventually announced a competition for an Advanced Encryption Standard (AES) in 1997. Finally, among five finalists –RC6 [124], Mars [28], Serpent [4], Twofish [126], and Rijndael [41]– NIST selected Rijndael, which has a fixed block size of 128 bits and supports a key size of 128, 192, or 256 bits. The AES block cipher standard is broadly employed to secure communication.

In many applications in which computing resources are limited, stream ciphers play a critical role. These ciphers can be adapted to specific implementation requirements. The cryptographic community has assisted business and industry alike by providing a broad range of stream ciphers. One such cipher is RC4 [122] designed in 1987 by Ron Rivest. Because of its simplicity of design and the high speed offered by software implementation, this cipher has gained popularity in many internet applications, such as TLS/SSL and WEP. A5/1 is also a well-known stream cipher, which is proposed to provide privacy of conversations on GSM mobile phones. The cipher was designed in 1987 and kept a secret, but it was reverse engineered by Briceno et al. in 1999 [27].

In November 2004, the European Network of Excellence for Cryptology (ECRYPT) [3] announced a dedicated project called eStream [2] to recognise efficient and secure stream ciphers for a wide range of applications. For software applications, eStream emphasised stream ciphers with high throughput requirements, while the hardware profile had focused on stream ciphers suitable for restricted resources, such as limited storage, power consumption, or gate count. At the end of the project, in April 2008, four software-based designs –HC-128 [139], Rabbit [23], Salsa20/12 [13], SOSEMANUK [11]– and three hardware designs –Grain [70], Mickey [8], Trivium [30]– were selected as eStream algorithms.

In 2007, NIST announced the SHA-3 Cryptographic Hash competition [1]. The project's purpose was to standardise one or more hash functions. After two rounds of the project, in December 2010, five algorithms selected: BLAKE [7], Grøstl [58], JH [138], Keccak [15] and Skein [51]. Recently, NIST completed the selection process of the next hash standard and reported the Keccak hash function [15] as the winner of the competition [1].

This thesis investigates the security of stream ciphers and hash functions as cryptographic primitives providing confidentiality, authentication, and integrity.

1.2.1 Hash functions

A hash function maps an input of arbitrary length to a string of fixed length. Thus the output of a hash function has a fixed length but the input stream can be a string of an arbitrary length. Formally, a cryptographic hash function H is a map from variable-length input bit strings to fixed-length output bit strings,

$$H : \{0,1\}^{\star} \to \{0,1\}^{n}.$$

In particular, H can be defined as

$$H : \{0,1\}^{k} \times \{0,1\}^{\star} \to \{0,1\}^{n}.$$

where $\star$ denotes a variable-length of a string. Also, n and k indicate a fixed-length of the binary strings.

Then the hash function is utilised for authentication and it is called a *message authentication code* (MAC). A keyed cryptographic hash function has two inputs, a shared secret key and an arbitrary-length message to be authenticated, and generates a MAC (or tag). In this case, everyone who knows the shared key can sign and verify the message to detect any modification. Otherwise, a cryptographic hash function is understood as a function without a key.

1.2.2 Cryptographic requirements

A secure hash function satisfies three fundamental security properties: preimage, second-preimage, and collision resistance. Figure 1.1 graphically clarifies the cryptographic requirements of a secure hash function.

- **Preimage resistance:** It should be computationally infeasible to find any input m which hashes to a pre-specified output $h = H(m)$.

- **Second-preimage resistance:** Given m such that $h = H(m)$, it should be computationally infeasible to find another distinct input m' that hashes to the same output, that is $h = m'$.

- **Collision resistance:** It should be computationally infeasible to find distinct input messages mapping to the same output value, which means that $m \neq m'$ but $H(m) = H(m')$.

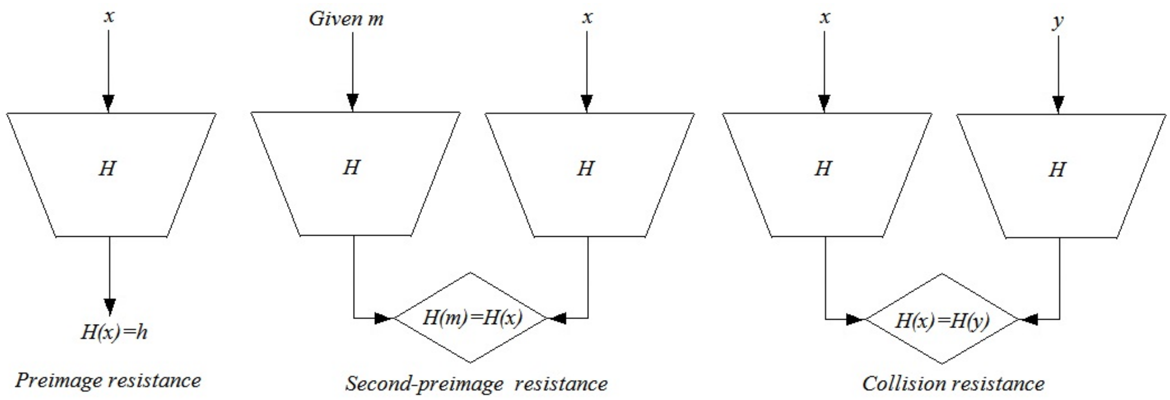

Figure 1.1: Essential security requirements to design hash functions

1.2.3 Hash function design

Cryptographic hash functions are constructed using three phases:

- domain extension (also called padding)
- iterative application of a compression function
- final output transformation.

The main goal of domain extension is to create a message whose length is a multiple of the hash function block size. Typically, the initial message (of arbitrary length) is padded by extra bits so that the message is formatted properly. The compressing phase processes the formatted message block by block. The final output transformation generates the required message digest.

The famous Merkle-Damgård (MD) construction is the basic structure of iterated hash functions [42, 111]. The construction iteratively transforms a message block and the previous chaining value as input to the next chaining value. Figure 1.2 shows the Merkle-Damgård construction, which starts from a fixed initial value (IV) and compresses the input message blocks to output the final hash digest (h). The following relations mathematically describe how the construction performs hashing progress to generate the hash digest value.

$$h_i = f(h_{i-1}, m_i), \quad h_0 = IV, \qquad i = 0, 1, \cdots, l-1$$

where $M = m_0||m_1||\cdots||m_{l-1}$ is the padded input message. A simple method to pad the message is appending a single '1' bit and '0' bits as many as required. The Merkle-Damgård construction is well understood and several generic attacks, such as

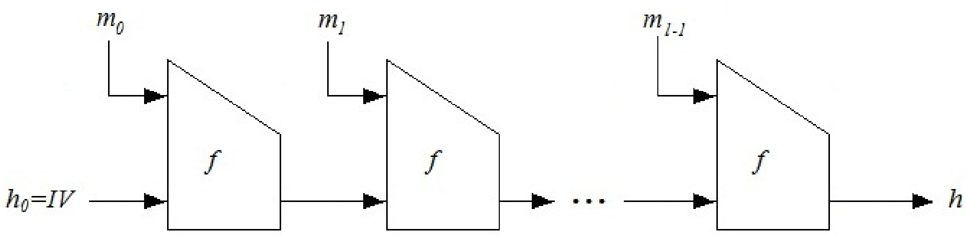

FIGURE 1.2: The Merkle-Damgård construction

differentiability [102], long-message second-preimages [44, 81], herding [80] , and multi-collisions [77] have been proposed for this construction. The strong point of the MD construction is its collision resistance. In other words, Damgård proved that finding a collision in a hash function implies finding a collision in its compression function. Unfortunately, in general, the second-preimage resistance of the MD construction deteriorates with the number of applications of the hash function.

To deal with the inherit weaknesses of the MD construction, there has been an extensive search for alternatives [14, 22, 95].

An alternative to MD is the sponge construction, proposed by Bertoni et al. [14]. The sponge scheme constructs a function with variable-length input and arbitrary output length, which makes it an alternative construction for hash functions [14] and stream ciphers [17]. This construction uses a fixed-length permutation f working on a fixed number of $b = r + c$ bits where r and c are internal parameters called the bit rate and the capacity respectively (see Figure 1.3). The sponge construction operates in two stages:

- Absorbing: The r-bit input message blocks (m_i) are linearly combined with the first r bits of the internal state. The next stage starts once all the input message blocks are injected into the function.

- Squeezing: In each round of operating the permutation f, the first r bits of the internal state are returned as output. The output can be considered as *hash digest* in a hash function or as *keystream bits* in the stream cipher mode.

Bertoni et al. [16] have proved the security of the construction by employing the indifferentiability concept proposed by Maurer et al. [102]. The designers [16] showed that to differentiate a sponge construction from a random oracle ,the success probability is upper bounded by $N^2 \cdot 2^{-(c+1)}$ where N is the number of calls to permutation and $N \ll 2^{c/2}$. It is interesting that the bound is independent of the output length. The sponge scheme is attractive from both the theoretical and practical perspectives. From the theoretical view, a random sponge function f is a random permutation or transformation. The sponge construction can be employed to design practical cryptographic hash functions and stream ciphers. As a significant example, the Keccak hash function [15] is based on the sponge structure. The Keccak has been chosen by NIST as the winner in the SHA-3 competition.

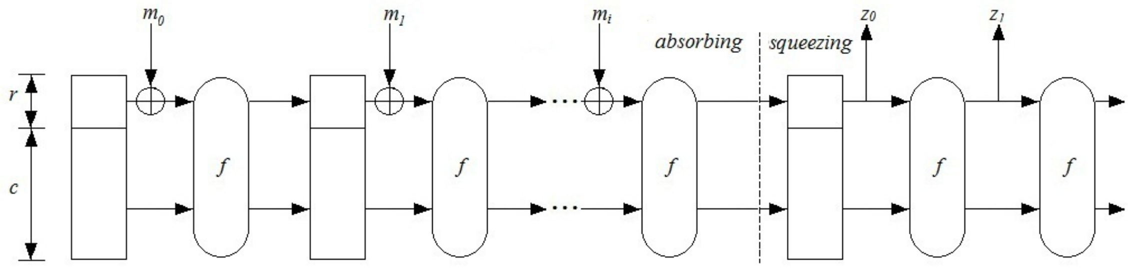

Figure 1.3: The sponge construction

1.2.4 Stream ciphers

Stream ciphers are symmetric cipher systems which provide confidentiality in many applications ranging from mobile phone communication to private virtual networks. They may be implemented efficiently in software and hardware and are a preferred choice when dealing with an environment that has restricted computing resources, such as smart cards and RFID tags. The inner workings of stream ciphers are controlled normally by two parameters: an initialisation vector (IV) and a key (K). The initialisation vector is public and the key is secret and shared between the sender and receiver. Normally the encryption and decryption algorithms are identical and produce a keystream of arbitrary length. The keystream bits are used to encrypt message bits (or decrypt ciphertext bits) by a simple XOR operation. A stream cipher typically consists of two parts: the initialisation algorithm and key generation algorithm. In many applications, the stream cipher needs to produce pseudorandom keystreams for different sessions with the same secret key. An initialisation algorithm mixes the secret key and IV securely, and provides initial state to generate keystream output bits.

Initialisation Algorithm: The inputs are a secret key K and a known initialisation value IV. The secret key and IV are mapped to the initial state of the stream cipher (S_0) at time $t = 0$ by an initialisation function denoted by F. Formally,

$$S_0 = F(K, IV) \tag{1.1}$$

Key Generation Algorithm: At time t, the stream cipher consists of

- internal state S_t,
- variable Var: it might include secret key, IV, plaintext P_t, ciphertext C_t or combination of them,
- update function g,
- output function f.

Generally, the operations on the encryption module are as follows:

$$\begin{cases} Z_t = g(S_t, Var) \\ C_t = Z_t \oplus P_t \\ S_{t+1} = f(S_t, Var) \end{cases} \tag{1.2}$$

At time t, a stream cipher generates output Z_t (keystream) to encrypt the plaintext P_t to produce the ciphertext C_t. Similar to 1.2, the decryption operations are:

$$\begin{cases} Z_t = g(S_t, Var) \\ P_t = Z_t \oplus C_t \\ S_{t+1} = f(S_t, Var) \end{cases} \tag{1.3}$$

Based on the variable Var involved in the update and output functions f and g, stream ciphers are categorised into two main classes:

- Synchronous, and
- Self-synchronising stream ciphers

In a synchronous stream cipher, the keystream is generated independently of the plaintext and ciphertext. So, the encryption transformation is:

$$\begin{cases} Z_t = g(S_t) \\ C_t = Z_t \oplus P_t \\ S_{t+1} = f(S_t) \end{cases} \tag{1.4}$$

and the decryption transformation becomes:

$$\begin{cases} Z_t = g(S_t) \\ P_t = Z_t \oplus C_t \\ S_{t+1} = f(S_t). \end{cases} \tag{1.5}$$

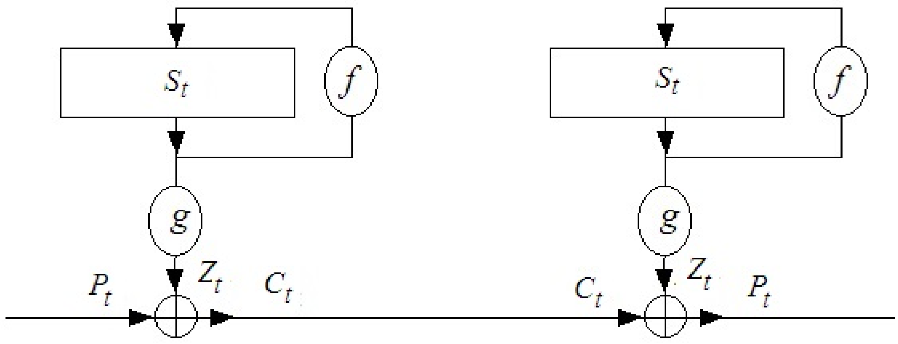

FIGURE 1.4: Encryption and decryption module of a synchronous stream cipher

A synchronous stream cipher is shown in Figure 1.4. A self-synchronising stream cipher generates the keystream Z_t depending on the plaintext or the ciphertext. By replacing $Var = P_t$ or $Var = C_t$ in Equations 1.2 and 1.3, two variations of self-synchronising stream ciphers can be defined. Equation 1.6 presents a typical encryption function in a self-synchronising stream cipher (See Figure 1.5).

$$\begin{cases} Z_t = g(S_t) \\ C_t = Z_t \oplus P_t \\ S_{t+1} = f(S_t, C_t) \end{cases} \tag{1.6}$$

Consequently, the decryption operations are:

$$\begin{cases} Z_t = g(S_t) \\ P_t = Z_t \oplus C_t \\ S_{t+1} = f(S_t, C_t) \end{cases} \tag{1.7}$$

The designer can add extra input variables, such as a secret key, IV, and ciphertext, into the functions g and f to make new stream ciphers.

1.3 Cryptanalysis

In the 19th century, Auguste Kerckhoffs, a Dutch linguist and cryptographer, identified six design principles that a cryptosystem has to satisfy. One of them states that the system/algorithm must be known to an adversary. The unknown element must be a secret key only [82]. In fact, the security of a cipher system must rely only on the secret key. According to this axiom, every cryptographic attack on a cipher intends to recover the secret key, which is known as *key-recovery attack*.

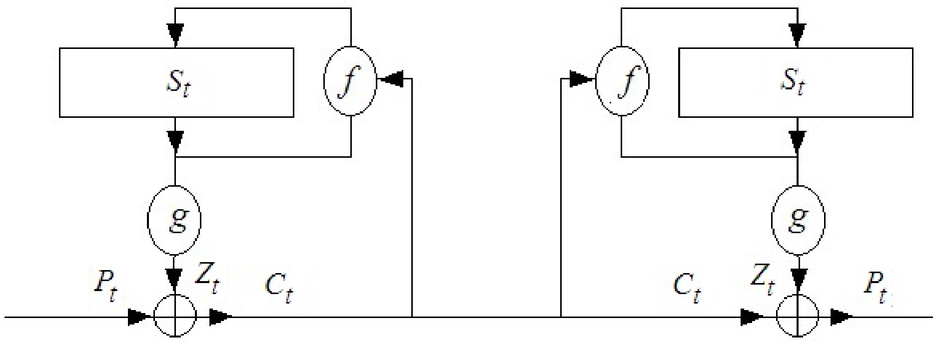

FIGURE 1.5: Encryption and decryption module of a self-synchronising stream cipher

1.3.1 Attack models

Assuming that an adversary can access the full communication between sender and receiver, the attack models are classified into four main cryptanalytic attacks:

- **ciphertext only attack:** The adversary is able to intercept an arbitrary number of ciphertexts.
- **Known plaintext attack:** The adversary is able to observe an arbitrary number of pairs consisting of a message and the corresponding ciphertext.
- **Chosen plaintext attack:** The adversary has access to the encryption device, but cannot see the secret key. This access is called the midnight attack. They can choose arbitrary messages and see the corresponding ciphertext.
- **Chosen ciphertext attack:** The adversary has access to the decryption device, but cannot see the secret key. They can choose arbitrary ciphertexts and see the corresponding messages.

In addition, there are several possible scenarios, such as *adaptively chosen plaintext attack* and *adaptively chosen ciphertext attack*, in which the attacker chooses the plaintext or ciphertext depending on the revealed information from previous (plaintext, ciphertext) pairs. Also in some cipher systems, there are other public input parameters, such as the initial value in stream and block ciphers from which the adversary can choose or adaptively choose to obtain (plaintext, ciphertext) pairs.

1.3.2 Distinguishing attacks

Apart from a key-recovery attack, which is the ultimate goal in cryptanalysis, another possible attack is known as a *distinguishing attack*. A distinguisher is an algorithm which distinguishes the output of a cipher from a truly random sequence. Distinguishing attacks are weak tests of cryptosystem security. However, once a distinguishing attack is developed, then it may be possible to convert it into a partial key-recovery attack. Identifying a correct guess from wrong ones to recover parts of the contents of the cipher is a common example of the benefits of distinguishing attacks.

1.3.3 Related-key attacks

In a related-key attack, an attacker tries to analyse an cryptographic algorithm by invoking the cipher with several secret keys which satisfy some known, or even chosen, relation. For example, the adversary attempts to find any relation between cipher outputs which have been encrypted or decrypted by related-secret keys[18]. The related-secret keys are typically several different keys whose exact values are unknown, but the adversary knows there is some mathematical relationship connecting the keys. For instance, the adversary might know that the number of ones of the related keys are always the same, although they do not know the contents of the keys. The model seems to be unrealistic, but in some practical applications the scenario can be applicable.

1.3.4 Generic attacks

Generic attacks do not exploit any specific property or internal structure of a cipher. Instead, the cipher is treated as a black box. They are useful for deriving security bounds so the user can choose large enough parameters (for cryptographic keys or the internal state length). They also help cryptographers to compare the general security aspects of ciphers.

To characterise the efficiency of cryptographic attacks, there are the following measures:

- Time complexity - this is the time needed to achieve the goal of the attack.
- Memory complexity - the storage required to launch the attack.
- Data complexity - the number of plaintexts and ciphertexts which are required.
- Probability of success - the expected probability of a successful attack.

1.3.5 Exhaustive search attack

An exhaustive search is a trivial approach to recovering the secret key of a cipher. Given a plaintext (P) and ciphertext (C) pair, the adversary searches through all possible keys, and marks the keys for which the observed plaintext is correctly encrypted. The secret key which encrypts P and gives C is the correct one. The attack is expected to recover the correct key after searching about half of the keys, on average.

1.3.6 Time-Memory Trade-Off attack

A time-memory trade-off (TMTO) attack is a generic method that can be used to invert one-way functions, such as an encryption. The attack, presented first by Martin Hellman [72] in 1980, recovers a key in $N^{\frac{2}{3}}$ operations, which uses $N^{\frac{2}{3}}$ words of memory where N is the number of all possible keys in the cryptosystem. The basic attack can be summarised as follows:

1. Choose M initial points randomly. Each point is a possible state of the cipher.

2. Compute a chain, by applying a function formed by the cipher, from each initial point to reach a final point.

3. Sort the initial and final point pairs based on the final point and store in a Table of size M.

4. Given a point, namely an output of the cryptosystem, build a chain of points and look them up in the Table.

5. Once a matching happens, recover the input point which has made the given point.

The first three items introduce the pre-computation phase and the last two items are an online phase which tries to find a secret key with complexity T. To get success in the attack, the values of M (memory complexity) and T (time complexity) should satisfy in the curve $T \cdot M^2 = N^2$ [72].

1.4 Thesis outline

The thesis investigates cryptanalysis of stream ciphers and hash functions based on stream ciphers with special attention to lightweight algorithms. The rest of the thesis is structured as follows.

Chapter 2 gives a brief introduction to the design and cryptanalysis of stream ciphers. The chapter explores several key techniques and primitive elements used to design stream ciphers, and cryptographic methods used to analyse these ciphers.

Chapter 3 focuses on the WG-7 stream cipher. This cipher is designed to be used for lightweight applications, such as RFID tags. It is shown that the keystream generated by WG-7 can be distinguished from a random sequence with about $2^{13.5}$ keystream bits and with a negligible error probability. A successful key-recovery attack on the cipher can be applied with time complexity of about $O(2^{27})$. The results were published in the Journal of Cryptography and Communications in December 2012.

Chapter 4 is a security evaluation of the Rakaposhi stream cipher, intended for uses in restricted computing resources, such as smart cards and sensor networks. The study identifies the weaknesses and properties of the cipher. The main observation is that the initialisation procedure has the so-called sliding property. This property can be used to launch distinguishing and key-recovery attacks. The distinguisher needs four observations of the related-secret key and initial value pairs. The key-recovery algorithm allows discovery of the 128-bit secret key after 2^9 initialisation operations. Further, the cipher is studied when the registers enter short cycles. In this case, the internal state can be recovered with less complexity than an exhaustive search. This result was published in the 9th International Conference on Information Security Practice and Experience (ISPEC 2013).

Chapter 5 discovers new security features of a specific design based on nonlinear feedback shift registers. The idea applies a distinguishing attack on linearly filtered nonlinear feedback shift registers. The attack also extends the idea to linear combinations of linearly filtered nonlinear feedback shift registers. The proposed attacks allow the attacker to mount a linear attack to distinguish the output of the cipher and recover its internal state. This approach indicates how invulnerable the modified version of the Grain stream cipher is against distinguishing attacks. This study has been accepted in the Journal of Mathematical Cryptology.

Chapter 6 analyses a new lightweight communication framework using authenticated encryption, called NLM-MAC, in wireless sensor networks. This chapter indicates several critical cryptographic weaknesses leading to key-recovery and forgery attacks. The internal state of the NLM-n generator can be recovered with time complexity of about $n^{\log_2 7 \times 2}$ where the total length of the internal state is $2 \cdot n + 2$ bits. The attack needs about n^2 keystream bits. It is shown that the attacker is able to forge any MAC tag in real time by having only one pair (MAC tag, ciphertext). The proposed attacks are completely practical and break the scheme with negligible error probability.

Chapter 7 demonstrates some new cryptographic attacks on RC4(n,m) stream cipher. The cipher is a modification of the original RC4 cipher proposed by Rivest. The investigations have revealed some weaknesses of the RC4(n, m) stream cipher. Firstly, a distinguisher for the cipher has been proposed. Secondly, a key-recovery attack uses a method to find the L-bit secret key with time complexity $(L/8).2^n$. When implemented on a standard PC, the attack is able to recover the secret key of RC4(8,32) in less than one second. This study has been accepted to present at the 6th International Conference on Security of Information and Networks Conference (SIN 2013).

Chapter 8 analyses the RC4-BHF hash function that is based on the well-known RC4 stream cipher. Two attacks on RC4-BHF have been developed. In the first attack, the adversary is able to find collisions for two different messages with time complexity around of 2^{13}, so it is very practical. The second attack shows how to design a distinguisher that can tell apart the sequence generated by RC4-BHF from a random one. The results were presented at the Australian Information Security Conference (AISC 2012).

Chapter 9 concludes the thesis and outlines directions for future work.

2
Stream Ciphers

In many applications where computing resources are limited, the cryptographic algorithms of choice are stream ciphers. They offer high speed and can be well adapted to specific implementation requirements. The cryptographic community has assisted business and industry alike by providing a wide range of stream ciphers. Various design strategies can help designers to ensure the specific criteria, in terms of security and performance, will be satisfied. This chapter describes the necessary background of design and cryptanalysis of stream ciphers, which are the main foci of this thesis.

2.1 Stream Cipher Design

Historically the first stream ciphers were based on very simple transformations. The driving force in the design was efficiency. Although this is still the case, the developed analytical techniques allow us to avoid designs that are insecure. Several basic early constructions are reviewed here.

2.1.1 Building Blocks

The main task for a stream cipher is to generate a keystream output whose statistical distribution is uniform and indistinguishable from a truly random distribution. In this view, building blocks to design secure ciphers play a critical role. In fact, strong building blocks can lead designers to construct reliable designs.

Linear feedback shift register

The linear feedback shift register (LFSR) is broadly utilised as a basic building block which generates good statistical distributions of outputs. Although a single LFSR is extremely vulnerable against cryptographic attacks, designers can exploit the uniform probability distribution and other statistical properties of LFSRs to design mathematically strong schemes.

An LFSR consists of several memory cells in which each cell is able to hold one single bit or word at a time. To update the contents, a new value is computed from some linear combination of the cells and the register is shifted one position and then the last value is discarded. The new value is placed in the first position. If the shifted value is a single bit, the LFSR is called bit-oriented, otherwise it is a word-oriented LFSR. Using larger alphabets in word-oriented LFSRs can generate more bits per iteration, and give better performance than the bit-oriented LFSRs. Since using words generally speeds up the keystream generation without a security penalty, most of the recent stream ciphers are constructed using word-oriented LFSRs. [11, 49, 114] are some stream cipher designs based on word-oriented LFSRs.

Definition 2.1.1 *The content of the cells at time t is called the internal state of the LFSR at time t and is denoted by $S_t = (s_{t+L-1}, s_{t+L-2}, \cdots, s_t)$, where s_{t+i} is the content of cell i at time t. The state S_0 is called the initial state of the LFSR.*

Figure 2.1 displays the general structure of an LFSR of length L. The feedback poly-

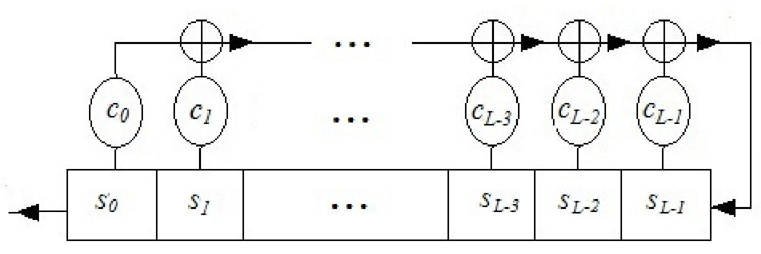

FIGURE 2.1: A general structure of an LFSR of length L at time *t=0*

nomial $f(x) = c_0 + c_1x + c_2x^2 + \cdots + c_{L-1}x^{L-1} + x^L$ defines all the properties of the LFSR. The polynomial $f(x)$ is irreducible over $\mathbb{F}_2[x]$ if it cannot be represented as the product of two polynomials in $\mathbb{F}_2[x]$ which both have a positive degree [109].

Definition 2.1.2 *An irreducible polynomial f(x) of degree n is called a primitive polynomial if the smallest positive integer j such that f(x) divides* $x^j - 1$ *is* $j = 2^L - 1$ *[109].*

Since every LFSR includes a limited number of cells, any sequence which has been produced by the LFSR must be repeated after a finite number of clocks.

Definition 2.1.3 *There is a positive integer* T*, called the period, which satisfies the relation* $z_t = z_{t+T}$ *where* z_t *is* t^{th} *of bits of a sequence and* $t \geq 0$.

Theorem 1 *If the feedback polynomial f(x) over* $\mathbb{F}_2[x]$ *of a linear feedback shift register is a primitive polynomial of degree L, then for every non-zero initial state the period of the LFSR is* $2^L - 1$.

Proof 1 *The proof can be found in [109].*

Definition 2.1.4 *The linear complexity of a sequence is the length of the shortest LFSR that generates the sequence.*

To compute the linear complexity of a given sequence, there is an efficient algorithm, the Berlekamp-Massey [12, 99], which finds the shortest LFSR of length l, given at least $2 \times l$ bits from the sequence. It is a necessary condition that a secure stream cipher must produce keystream with large linear complexity.

Nonlinear feedback shift register

By definition an LFSR uses a linear recursion to modify its internal state. The internal state can also be modified using a nonlinear recursion. This kind of register is called a nonlinear feedback shift register (NLFSR). Figure 2.2 shows the general structure of an NLFSR of length L. While the mathematics behind LFSRs is well understood, the

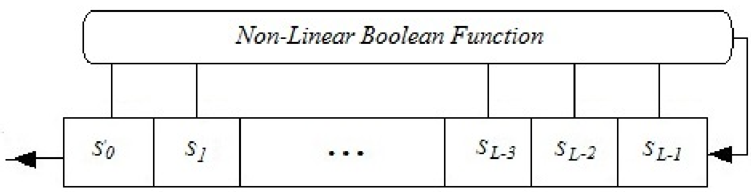

FIGURE 2.2: A general structure of an NLFSR of length L at time *t=0*

theory of NLFSRs is in its infancy stage. There are still many basic open problems related to NLFSRs [63]. For example, we do not know how to determine the period, identify different cycles, or find out the linear complexity of NLFSRs. The lack of understanding of the mathematics behind NLFSRs does not prevent a proliferation of stream cipher designs based on NLFSRs. For instance, the two eStream finalists, the Trivium [30] and Grain [70] ciphers, exploit one or several NLFSRs combined with LFSRs. Other ciphers, such as Achterbahn [56], Rakaposhi [35], KATAN/KTANTAN [29], and the NLM generator [90], use the NLFSRs as a building block to protect the designs from the basic linear and algebraic attacks.

Boolean function

A Boolean function f is a mapping $f : \mathbb{F}_{2^n} \to \mathbb{F}_2$ which inputs n bits and outputs one bit (*e.g.* $f(x) = f(x_0, x_1, \cdots, x_{n-1})$). There are two main representations of a Boolean function: truth table (TT) and algebraic normal form (ANF). The truth table form of Boolean function f on $\mathbb{F}_{2^n}$ is a binary vector of length 2^n, so that the first element is $f(0) = f(0, 0, \cdots, 0)$ and the last one is $f(2^n - 1) = f(1, 1, \cdots, 1)$ and every element in $\mathbb{F}_2$ corresponds to the output of the f function with a $n-$bit input. The Boolean function f is denoted balanced if the number of ones and zeros in TT are equal. Another representation of a Boolean function is the ANF. An ANF of a Boolean function on $\mathbb{F}_{2^n}$ is a polynomial of the following form:

$$f(x_0, x_1, \cdots, x_{n-1}) = \bigoplus_{i=(i_0,i_1,\cdots,i_{n-1})\in\mathbb{F}_{2^n}} a_i x_0^{i_0} x_1^{i_1} \cdots x_{n-1}^{i_{n-1}}$$

where $a_i \in \mathbb{F}_2$. The algebraic degree of f, called $deg(f)$, is the maximum number of variables involved in the terms of the ANF of f.

Definition 2.1.5 *A boolean function f is called an affine function if $deg(f) \leq 1$. The set of affine functions is denoted by $A(n)$.*

An affine function with $deg(f) = 0$ (*e.g.* $f(x) = 0$ or $f(x) = 1$) is called a constant function. An affine function without the constant term is called a linear function.

The nonlinearity of an n-variable Boolean function is the minimum distance from the set $A(n)$ of all n-variable affine functions:

$$nl(f) = \min_{g(x)\in A(n)} |\{x \in \mathbb{F}_{2^n} : f(x) \neq g(x)\}|$$

The correlation immunity of a Boolean function indicates the degree to which its outputs are uncorrelated with some subset of its inputs. The Boolean function $f(x)$ is called correlation immune with respect to the subset $K \subset \{1, 2, \cdots, n\}$ if the probability for f to get any value from $\{0, 1\}$ is not changed, so that $x_i, i \in K$ are fixed and other variables are chosen independently at random. A function is said to be r^{th} order correlation immune if $f(x)$ and any set of r or fewer variables in x are statistically independent. Also, if $f(x)$ is correlation immune of order r, then it is correlation immune for any order less than r. A balanced r^{th} order correlation immune function is called $r-$resilient.

The Walsh Hadamard transform of $f(x)$ is a real valued function over $\mathbb{F}_{2^n}$ that can be defined as

$$W_f(\omega) = \sum_{x \in \mathbb{F}_{2^n}} (1)^{f(x)+x\cdot\omega}$$

where $\omega = (\omega_0, \omega_1, \cdots, \omega_{n-1}) \in \mathbb{F}_{2^n}$ and $x \cdot \omega = x_0 \cdot \omega_0 \oplus x_1 \cdot \omega_1 \oplus \cdots \oplus x_{n-1} \cdot \omega_{n-1}$.

A function $f(x_0, x_1, \cdots, x_{n-1})$ is r^{th} order correlation immune iff its Walsh transform satisfies $W_f(\omega) = 0$, for $1 \leq wt(\omega) \leq m$ where $wt(\omega)$ is the Hamming weight or the number of ones in ω. The Boolean function f is balanced iff $W_f(0) = 0$. The function $f(x_0, x_1, \cdots, x_{n-1})$ is r^{th} resilient iff its Walsh transform satisfies $W_f(\omega) = 0$, for $0 \leq wt(\omega) \leq m$. The relationship between the nonlinearity of $f(x)$ and the Walsh Hadamard Transform is

$$nl(f) = \frac{1}{2}(2^n - WHT_{max})$$

where WHT_{max} is the maximum absolute value of the Walsh Hadamard Transform [141].

2.1.2 Some design techniques

Shift register based stream ciphers

Linear feedback shift registers are broadly used to design stream ciphers because they are well-suited for software and hardware implementation, have a large period, and good statistical properties. However, these registers are vulnerable against linear attack when their output is used alone [109]. To destroy the linearity inherent in LFSRs, designers use several approaches: nonlinear combination, filter, and clock control generators.

A nonlinear combination generator mixes outputs of n parallel LFSRs by using a highly nonlinear boolean function f, known as the combining function, to generate keystream output. Figure 2.3 presents the construction. Suppose that n LFSRs of lengths $L_1, L_2, \cdots, L_n$, are combined by a nonlinear function $f(x_1, x_2, \cdots, x_n)$. Then the keystream z_t is the output of f at time t as follows:

$$z_t = f(s_t^1, s_t^2, \cdots, s_t^n)$$

where s_t^i is the output of the i^{th} LFSR at time t.

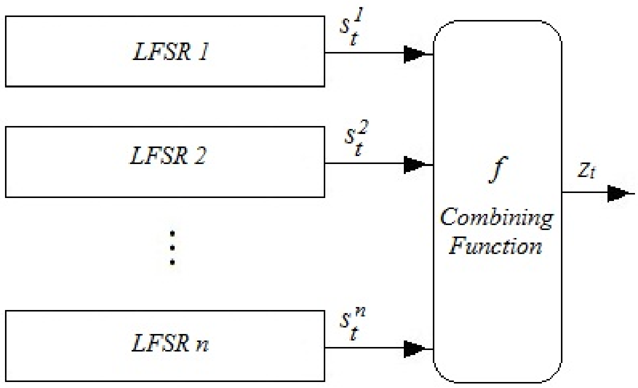

FIGURE 2.3: A general structure of a nonlinear combination generator

Another general construction is a filter generator which uses a single LFSR and a nonlinear function, called a filtering function, to generate keystream outputs. Figure 2.4 illustrates an overall view of the filter generator. Suppose that an LFSR of length L is filtered by a nonlinear function $f(x_1, x_2, \cdots, x_n)$. Then the keystream z_t is the output of f at time t as follows:

$$z_t = f(s_{t+i_0}, s_{t+i_1}, \cdots, s_{t+i_{n-1}})$$

where s_{t+i} is the i^{th} bit of the internal state of the LFSR at time t and $0 \leq i_0, i_1, \cdots, i_{n-1} \leq L-1$.

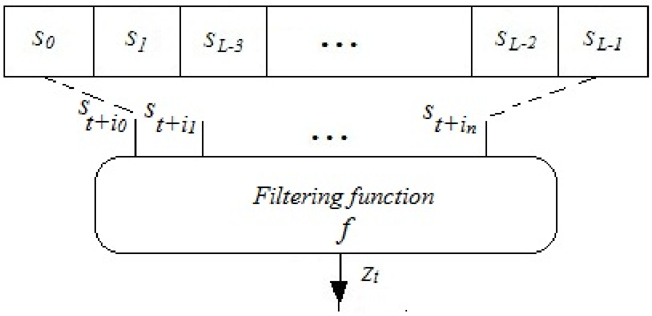

FIGURE 2.4: A general structure of a filter generator

Unlike nonlinear combination generators and nonlinear filter generators in which LFSRs are clocked regularly, clock-controlled generators use irregular clocking to remove linearity of outputs. The main idea with these generators is to use an LFSR to control movement of other LFSRs. Two well-known clock-controlled generators are the alternating step generator and the shrinking generator [109].

The alternating step generator utilises an LFSR R_1 to control the clocking of two LFSRs, R_2 and R_3. Figure 2.5 depicts the alternating step generator. Suppose that the output sequences of LFSR R_i are $r^i_0, r^i_1, r^i_2, \cdots$ where r^i_k is the k^{th} bit produced by LFSR r^i for $0 \leq i \leq 2$. Then the keystream generated by the cipher is

$$z_t = r^1_{c(t)} \oplus r^2_{t-c(t)-1}$$

where $c(t) = (\sum_{i=0}^{t} r^0_i) - 1$ for $t \geq 0$.

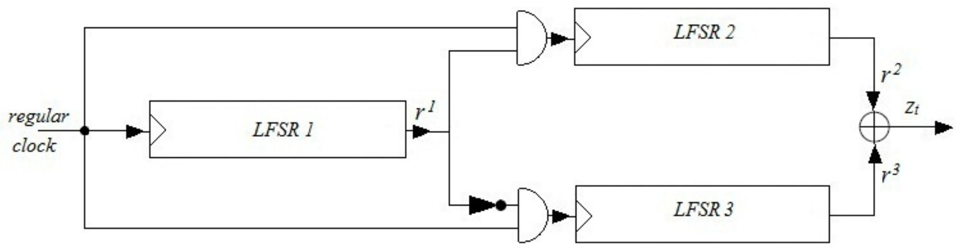

Figure 2.5: A general structure of an alternating step generator

The shrinking generator uses two LFSRs: one generates sequences and another one decides whether to output the sequences as keystreams or discard them. Figure 2.6 shows a view of a shrinking generator. Suppose that the output sequences of LFSR R_i are $r^i_0, r^i_1, r^i_2, \cdots$ where r^i_k is the k^{th} bit produced by LFSR r^i for $0 \leq i \leq 1$. Then the keystream generated by the generator is

$$z_t = r^1_{c(t)}$$

for $t \geq 0$; $c(t)$ is the position of the t^{th} '1' in the sequence $r^2_0, r^2_1, r^2_2, \cdots$.

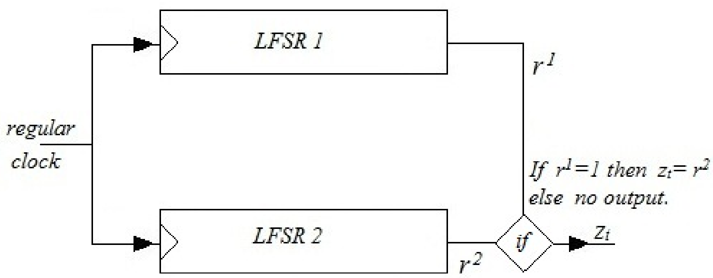

Figure 2.6: The structure of a shrinking generator

Array and modular addition based ciphers

Many modern stream ciphers utilise large arrays and simple operations, such as modular additions, rotations and exclusive-or. RC4 [122], Py [20], HC-256 [139], and RC4(n, m) [65] use word based arrays and modular addition to produce pseudorandom keystreams. Addition modulo 2^n ($f : \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2^n$) is a nonlinear mapping over $\mathbb{F}_2$ which provides fast computing in most implementation environments.

One of the most widely used ciphers is RC4, designed in 1987 by Ron Rivest. The cipher uses a large internal state that is stored in an array of words. Because of its simplicity of design and the high speed offered by software implementation, the cipher has gained popularity in many internet applications. RC4 is a family of stream ciphers indexed by an integer n that indicates the size of the word in bits. The internal state is an array S of 2^n words.

RC4 consists of two algorithms. The first is a key-scheduling algorithm (KSA) which initialises the internal state. The second is a pseudorandom generation algorithm (PRGA). It generates the output keystream. The KSA algorithm takes an array S and a secret key K and produces the initial state or a secret permutation of $\{0, 1, 2, \ldots, 2^n - 1\}$. The PRGA algorithm accepts the initial state S and produces a sequence of words consisting of one word per clock. A popular instantiation of RC4 is for $n = 8$. In this instantiation, words are 8 bits long and the array S contains $2^8 = 256$ entries.

The blocks in question are:

- KSA (key-scheduling algorithm) – this function takes as an input a $l-$byte secret key array $K = (K[0], \ldots, K[l-1])$ and initialises the internal state $\langle S \rangle$, where $S = (S[0], \ldots, S[255])$ is a 256-byte sequence. The function is fully described in Figure 2.7.

- PRGA (pseudorandom generation algorithm) – this function takes the internal state $\langle S \rangle$ as the input and updates it to generate keystream bytes (*output*). The pseudocode of the function is given in Figure 2.8.

2.2 Cryptanalysis of stream ciphers

2.2.1 Linear Cryptanalysis

Linear cryptanalysis was first proposed to investigate the security of block ciphers. The first attacks were applied to the FEAL cipher [128] and DES [45] in the early 1990s [100, 101]. In the analysis, nonlinear functions are approximated by linear functions.

1. **Input:** Secret key K (l bytes).
2. **Output:** Internal State $\langle S \rangle$.
3. **for** $i = 0$ **to** 255
4. $S[i] = i$;
5. **end for**
4. **for** $i = 0$ **to** 255
5. $j = (j + S[i] + K[i \mod l]) \mod 256$;
5. $swap(S[i], S[j])$;
9. **end for**

Figure 2.7: KSA Function

Clearly, it is desirable to find linear functions that are very "close" to their nonlinear siblings. The quality of approximation is measured by the probability of finding the correct outputs.

Linear distinguishing attack

A distinguisher constructs a linear relation which involves only keystream outputs so that

$$Pr[\bigoplus_{i \in \rho} z_{t+i}] = \frac{1}{2} + \epsilon$$

where ρ represents certain coefficients making the distinguisher based on finding the bias (ϵ) in the keystream bits and $t \geq 0$. The required outputs should be in order ϵ^{-2} to distinguish the cipher from truly random sequences. The linear distinguishing attack has been applied on a large variety of stream ciphers, such as LFSR based ciphers [24, 34, 69, 116] and array based stream ciphers [75, 79, 117, 119, 120].

2.2.2 Correlation attack

The correlation attack uses a correlation between the output sequence and the internal state of the cipher. The first proposed attack, introduced by Siegenthaler [129–131], seeks a statistical bias between one or more LFSRs and the keystream bits in a nonlinear combination generator. Suppose that a correlation between the first LFSR of length L_1

Input: Internal State $\langle S \rangle$

Output: Updated Internal State $\langle S, j \rangle$, keystream bytes ($Output$)

$i = 0; \quad j = 0;$

Output loop

$\quad i = i + 1 \mod 256;$

$\quad j = (j + S[i]) \mod 256;$

$\quad swap(S[i], S[j]);$

$\quad Output = S[(S[i] + S[j]) \mod 256;$

FIGURE 2.8: PRGA Function

and the keystream z_t has been found in the nonlinear combination generator mentioned in the previous section (see Figure 2.3). We show the bias as $Pr[s_t^1 = z_t] = \frac{1}{2} + \epsilon$. Then, by testing all possible 2_1^L initial states, and checking the correlation between the output keystream and the internal state of the LFSR, the attacker can find the most probable initial state of the LFSR. This means the attacker does not need to search all states of LFSRs, and he can recover each LFSR in a separate process. The main result found by Siegenthaler is that the combining function should have a high correlation immunity.

The time complexity of the correlation attack is exponentially related to the length of the LFSRs. Also the attacker can apply more sophisticated methods. Amongst many published papers, the fast correction attack deserves to be mentioned due to its efficiency and low data complexity. In [31, 76, 107], the role of feedback polynomials to prevent the correlation attacks has been examined. As a result, designers should use an LFSR with a large state with high-weight feedback functions, and high correlation immune combining functions to protect the design against correlation and fast correlation attacks.

2.2.3 Differential Cryptanalysis

A differential attack exploits leaked information where a difference in the state spreads through the state during initialisation and key-generation phases. [19] provides a framework to apply differential attacks on stream ciphers. For synchronous stream ciphers, the attacker divides the cipher into three distinct parts to find differentials which may help to recover the secret key as follows by:

- Finding a difference in the secret key or (and) the IV to generate a difference in the internal state.
- Tracking the difference which propagates through the internal state-update function.
- Discovering the difference in the internal state, which produces a keystream difference.

Once the attacker finds a difference in the secret key or the IV, which generates a keystream difference, then he can apply a distinguishing attack and even a key-recovery attack on the cipher.

2.2.4 Algebraic Cryptanalysis

Algebraic attacks on stream ciphers are powerful cryptanalytic techniques to analyse the algebraic character of the ciphers. Basically, an algebraic attack can be divided into two main phases: computing a system of equations corresponding to the keystream outputs, and solving the system to recover the internal state.

To explain the main idea of the algebraic attack, we consider a general scheme of filter generators: an LFSR of length L bits and a filtering function f illustrated in Figure 2.4. Let z_t, $t \geq 0$ be the keystream generated by the cipher, and f be a nonlinear map defined from $GF(2)^n \rightarrow GF(2)$ with algebraic degree d. The keystream can be written by:

$$f(T(s_0, ..., s_{L-1})),$$

where $T(s_0, ..., s_{L-1})$ extracts the L-bit content of the register. So, the system of relations can be determined as follows:

$$\begin{cases} z_0 = f(T(s_0, ..., s_{L-1})) \\ z_1 = f(T(P(s_0, ..., s_{L-1}))) \\ \ldots \\ z_t = f(T(P^t(s_0, ..., s_{L-1}))) \end{cases} \tag{2.1}$$

where $f(T(P^t(s_0, ..., s_{L-1})))$ indicates the output keystream at the clock t, generated by nonlinear filtering of the internal state $(s_0, ..., s_{L-1})$ under feedback polynomial P. Now the cryptanalytic problem converts into the problem of solving a system of nonlinear equations [5, 36–38, 40].

The simplest scenario to solve System (2.1) is known as the linearization technique [38, 39]. The number N of monomials of degree smaller than or equal to d is

$$N = \sum_{i=1}^{d} \binom{L}{i} \approx \binom{L}{d}.$$

Each of these monomials can be considered as a new variable and then the attacker can solve the nonlinear system with $\approx N$ equations and time complexity $\approx N^{\log_2 7}$ by the Gaussian elimination method.

The important idea to improve the efficiency of the algebraic attack is to reduce the degree of the equations. For this purpose, the attacker needs to find an annihilator function so that $f \cdot g = 0$ and $\deg g < \deg f$. The steps to apply the attack can be described as follows:

1. Find an annihilator g of f or $f \oplus 1$ with a low degree $\hat{d}, \hat{d} < d$.
2. Given multivariate equations of a low degree $\hat{d}$ on the initial state bits, there are $\hat{N} = \sum_{i=1}^{\hat{d}} \binom{L}{i}$ monomials of degree no bigger than $\hat{d}$, where L is the length of the internal state. In the linearisation method, the time complexity to solve the nonlinear system is $\hat{N}^{\log_2 7}$. The memory complexity of the attack is about $\hat{N}$.

This means that the attacker reduces the time complexity from $N^{\log_2 7}$ to $\hat{N}^{\log_2 7}$, and memory complexity from N to $\hat{N}$.

Fast algebraic attacks

Fast algebraic attacks [36, 38, 68] on LFSR based stream ciphers are based on equations of type $zX^e + X^d$ with $e < d$. This is shorthand to describe that at least one equation of type

$$z \cdot g(s_0, \cdots, s_{L-1}) + h(s_0, \cdots, s_{L-1}) = 0 \tag{2.2}$$

exists, where g and h are some multivariate polynomials of degree e and d $(e < d)$ respectively, and $z = f(s_0, \cdots, s_{n-1})$. The attack can be summarised as follows:

$$\sum_{i=t}^{t+D} \alpha_{t+i}.z_i.g(P(L^i(s_0, \cdots, s_{160}))) \tag{2.3}$$

for some linear combination $(\alpha_0, \cdots, \alpha_{D1}) \in GF(2)^D$, where $D = \sum_{i=1}^{d} \binom{n}{i}$. The same equation applies to each window of D consecutive steps and it can be written E times, for E overlapping intervals, with $E = \sum_{i=1}^{e} \binom{n}{i}$. This is because we need to get the final system of degree e that is solvable by linearisation (with complexity $E^{\log_2 7}$). This approach is discussed in [5, 6, 38, 68]. The steps of the improved attack are summarised as follows:

1. Relation step: One searches g and h with small degrees such that $f \cdot g = h$. The lower bound on the complexity of solving a linear system with $D + E$ equations is $O((D + E)^{\log_2 7})$. In general one considers $e < d$.

2. Pre-computation step: One Computes linear relations to eliminate the terms of degree greater than e in the equations. This needs $2 \cdot D$ bits of stream bits with complexity $O(D \cdot (\log_2(D))^2)$.

3. Substitution step: One eliminates the monomials of degree greater than e. The time complexity is $O(E^2 \cdot D)$ [6] but by DFT [68] it can be further reduced to $O(E \cdot D \cdot \log_2(D))$.

4. Solving step: One solves the system with E linear equations in $O(E^{\log_2 7})$.

3

Cryptanalysis of WG-7 stream cipher

WG-7 [96] is a fast lightweight stream cipher whose design was inspired by the family of WG stream ciphers [114]. The original WG is a synchronous stream cipher submitted to the ECRYPT call. Both WG-7 and WG are hardware-oriented stream ciphers that use a word-oriented linear feedback shift register and a filter function based on the Welch-Gong (WG) transformation [64]. The structure of WG-7 is similar to the WG stream cipher. Both ciphers use LFSRs and filtering functions, however, WG works in $GF(2^{29})$ but WG-7 in $GF(2^7)$. WG-7 uses an 80-bit secret key and a 81-bit initial vector (IV). WG-7 works as follows. First the secret key and IV are used to initialise the internal state of the cipher LFSR. Next, the LFSR with its nonlinear function is clocked 46 times. After this initialisation procedure the cipher generates an appropriate string of keystreams that is used for encryption.

We assume that the initialisation procedure of WG-7 is performed as prescribed. Consequently, the internal state consists of 161 bits. Note that the security level claimed by the designers is 80 bits. The cipher has been designed for encryption in resource restricted environments, such as RFID applications, mobile phones and smart cards. The authors of the cipher analysed the design and concluded that WG-7 [96] is secure against time-memory-data trade-off attacks, differential attacks, algebraic attacks and correlation attacks.

This section is organised as follows. Section 3.1 gives a brief description of the keystream generator of the WG-7 stream cipher. Section 3.2 deals with cryptographic weaknesses of the algorithm, which lead to our distinguishing and key-recovery attacks. Section 3.3 concludes the chapter.

3.1 Description of WG-7

The structure of the WG-7 stream cipher is illustrated in Figure 3.1. It consists of a 23-word LFSR, where a single word is 7 bits long. The filter function WG is a nonlinear function defined for 7 boolean variables (a word). The word is an element of $\mathbb{F}_{2^7}$, where the finite field $\mathbb{F}_{2^7}$ is defined by the primitive polynomial $g(x) = x^7 + x + 1$ over $GF(2)$. The characteristic polynomial of the LFSR is primitive over $\mathbb{F}_{2^7}$ and is given by:

$$f(x) = x^{23} + x^{11} + \beta, \tag{3.1}$$

where β is a root of $g(x)$. The nonlinear filter function WG(x) denoted in Figure 3.1 as WG is a transformation $\mathbb{F}_{2^7} \to \mathbb{F}_2$ as defined below:

$$WG7(x) = f(x) = Tr(x^3 + x^9 + x^{21} + x^{57} + x^{87}), \quad x \in \mathbb{F}_{2^7}. \tag{3.2}$$

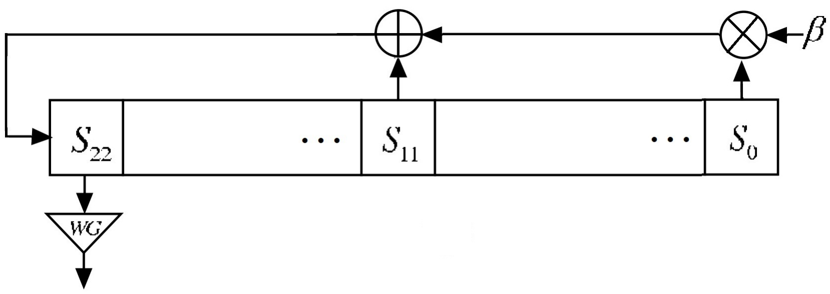

FIGURE 3.1: The WG-7 stream cipher scheme

where $Tr(x) = x + x^2 + \cdots + x^{2^{n-1}}$ is the trace function from $\mathbb{F}_{2^n} \to \mathbb{F}_2$.

3.2 Cryptanalysis of WG-7

In this section, we describe our two attacks for WG-7. The first attack distinguishes the WG-7 stream cipher from the random one. The distinguishing attack, discussed in Section 1.3.2, exploits a bias in a linear approximation of the nonlinear filter function. The second attack is a variant of the fast algebraic attack. It permits us to recover not only the internal state of the cipher, but also the secret key.

3.2.1 Distinguishing Attack for WG-7

The WG-7 stream cipher has a relatively simple structure. The main component is an LFSR that generates words that are later transformed in a nonlinear fashion by the filter function. The filter function is the only nonlinear component in the cipher. It seems to be quite a reasonable idea to check how well the filter function can be approximated by an affine function. In other words, we are looking for an affine function that approximates the filter function as closely as possible (the best linear approximation). If we apply the well-known Walsh-Hadamard transform to the filter function, then we can obtain such a linear approximation. Denote it by $\Gamma \cdot (x_0, ..., x_6) + \alpha$, where x_i is the i-th bit of the word x, the sign "$\cdot$" is the inner product, Γ ($\Gamma \in \mathbb{F}_{2^7}$) is a constant (a vector of 7 binary constants) and α is a binary constant. In the case of WG-7, we have found that there are seven affine functions, which are the best linear approximations (one such function is $1 + x_0 + x_1 + x_4$). As the nonlinearity of the filter function is 52, we can find the following probability:

$$Pr(WG(x) = (\Gamma \cdot x + \alpha)) = \frac{2^7 - 52}{2^7} = 0.59375 \tag{3.3}$$

From Equation (3.1), the following recursive relation can be derived:

$$S_{i+23} = S_{i+11} \oplus \beta \cdot S_i. \tag{3.4}$$

Consequently, we need to find the best linear approximation of the relation given below:

$$WG(S_{i+23}) \oplus WG(S_{i+11}) \oplus WG(S_i) = 0. \tag{3.5}$$

Remark 1: The piling up lemma cannot be used to compute the bias of Equation (3.5) because the input variables $(S_{i+23}, S_{i+11}, S_i)$ are not independent. In particular, S_{i+23} is correlated with other variables by Equation (3.1). In addition, $\beta \cdot S_i$ in Equation (3.4) is a linear transformation of S_i. The precise linear relations are given below:

$$\beta \cdot S_i = \beta \cdot (s_0^i, s_1^i, s_2^i, s_3^i, s_4^i, s_5^i, s_6^i) = \begin{vmatrix} s_1^i \oplus s_3^i \oplus s_4^i \\ s_2^i \\ s_2^i \oplus s_5^i \\ s_4^i \\ s_1^i \oplus s_2^i \\ s_6^i \\ s_0^i \oplus s_1^i \oplus s_2^i \oplus s_3^i \oplus s_4^i \oplus s_5^i \oplus s_6^i \end{vmatrix}^T \tag{3.6}$$

We need to determine the exact value of the bias ε in the following probability:

$$Pr(WG(S_{i+23}) \oplus WG(S_{i+11}) \oplus WG(S_i) = 0) = 0.5 + \varepsilon \tag{3.7}$$

One method to compute the bias in Equation (3.7) is as follows. We consider the bias between three output bits at clocks i, $i+11$ and $i+23$. So we get

$$\begin{aligned} z_{i+23} \oplus z_{i+11} \oplus z_i = & \\ & = WG(S_{i+23}) \oplus WG(S_{i+11}) \oplus WG(S_i) \\ \overset{From\ Eq.\ 3.4}{\Longrightarrow} & = WG(S_{i+11} \oplus \beta \cdot S_i) \oplus WG(S_{i+11}) \oplus WG(S_i) \end{aligned} \tag{3.8}$$

Observe that Equation (3.8) is a boolean function with 14 input variables (instead of 21 variables) and a single bit output. In other words, S_{i+23} depends on S_i and S_{i+11} based on Equations (3.4) and (3.6). Let $F \cdot GF(2^{14}) \rightarrow GF(2)$ be a nonlinear boolean function in the form:

$$F(S_i, S_{i+11}) = WG(S_{i+11} \oplus \beta.S_i) \oplus WG(S_{i+11}) \oplus WG(S_i) \tag{3.9}$$

Now we focus on $F(s_0^i, s_1^i, ..., s_6^i, s_0^{i+11}, s_1^{i+11}, ..., s_6^{i+11})$ that is an unbalanced boolean function, where

$$Pr(F(s_0^i, s_1^i, ..., s_6^i, s_0^{i+11}, s_1^{i+11}, ..., s_6^{i+11}) = 0) = \frac{1}{2} - 2^{-7.145}. \tag{3.10}$$

The relation given by Equation (3.9) defines a distinguisher, which is able to tell apart the output of the stream cipher from a truly random cipher with the probability expressed by Equation (3.10). The interesting question is whether there are better biases to mount a distinguishing attack. We will discuss possible answers in the remaining part of this section.

Better biases:

In the previous section, we have found a linear approximation leading us to a distinguishing attack. One wonders whether it is possible to find a better linear approximation so that the bias is closer to the maximal value of 0.5.

Let us explore this issue in more detail. Repeated squaring of the characteristic polynomial of the LFSR (see Equation 3.1) gives other linear recurrence polynomials. If we use the exponent 2^7, we get

$$x^{23 \cdot 2^7} + x^{11 \cdot 2^7} + \beta^{2^7} = 0 \tag{3.11}$$

Since $\beta = \beta^{2^7}$, $\beta \in \mathbb{F}_{2^7}$, the summation of Equations (3.1) and (3.11) gives:

$$x^{23 \cdot 2^7} + x^{11 \cdot 2^7} + x^{23} + x^{11} = 0 \tag{3.12}$$

$$\overset{divided\ by\ x^{11}}{\Longrightarrow} x^{23 \cdot 2^7 - 11} + x^{11 \cdot 2^7 - 11} + x^{12} + 1 = 0. \tag{3.13}$$

This means that the attacker can derive a bitwise linear equation, which is valid for the internal state of the LFSR. Similar to the previous subsection, the function F can be built as follows:

$$\begin{aligned} & z_{i+23\cdot 2^7-11} \oplus z_{i+11\cdot 2^7-11} \oplus z_{i+12} \oplus z_i \\ &= WG(S_{i+23\cdot 2^7-11}) \oplus WG(S_{i+11\cdot 2^7-11}) \oplus WG(S_{i+12}) \oplus WG(S_i) \\ &= WG(S_{i+11\cdot 2^7-11} \oplus S_{i+12} \oplus S_i) \oplus WG(S_{i+11\cdot 2^7-11}) \oplus WG(S_{i+12}) \oplus WG(S_i) \end{aligned} \quad (3.14)$$

Equation (3.14) can be considered as a boolean function with 21 input variables (instead of 28 variables) and a single bit output. The boolean function $F : GF(2^{21}) \rightarrow GF(2)$ is an unbalanced boolean function, where

$$Pr(F(S_{i+11\cdot 2^7-11}, S_{i+12}, S_i) = 0) = \frac{1}{2} + 2^{-6.78} \quad (3.15)$$

The required data:

Now, we explain the number of output sequences required to distinguish WG-7 from a truly random cipher. The following theorem determines the required length of keystream needed to distinguish between two random sequences, where one is uniform (both binary values occur with probability $\frac{1}{2}$) and the other is biased (one value occurs with probability $\frac{1}{2}(1+\varepsilon)$) [98].

Theorem 2 *Given two binary random sequences, where the first is uniform and the other is biased, i.e. one binary value occurs with the probability $\frac{1}{2}(1+\varepsilon)$ and the other with the probability $\frac{1}{2}(1-\varepsilon)$, then we need to observe $O(\frac{1}{\varepsilon^2})$ bits in order to distinguish the two distributions with a non-negligible probability of success.*

Proof 2 *The proof can be found in [109].*

In this case, the amount of data required for the proposed distinguishing attack is $2^{13.56}$ bits. This amount of data can be collected from consecutive (or non-consecutive) keystreams and even from one session key or from different session keys at various times.

The results of the implemented distinguishing attack on the WG-7 stream cipher are shown in Table 3.1. We have repeated the experiment 1000 times to compute the success rate of the distinguishing attack with different lengths of output sequence.

TABLE 3.1: Experimental results for applying the distinguishing attack on WG-7

	Used Data (bits)	Success Rate
1	2^9	68%
2	$2^{9.8}$	75%
3	$2^{10.3}$	85%
4	$2^{11.5}$	90%
5	$2^{13.5}$	99.99%

3.2.2 Key-Recovery Attack on WG-7

In this section, we apply an algebraic analysis to recover the initial state of the cipher, and consequently the secret key. Our attack can recover the internal states of WG-7 and then the attacker is able to clock the LFSR backward and find the secret key correctly. The designers of the WG-7 stream cipher have claimed that there is no algebraic attack with complexity smaller than the exhaustive search and with data complexity smaller than 2^{24} of consecutive keystream bits. The idea of our attack is as follows. Let $L : GF(2^{161}) \rightarrow GF(2^{161})$ be a multivariate linear transformation that corresponds to the linear transformation defined by a single clock. This transformation is done on the whole state of 23 registers each holding 7 bits $(23 \cdot 7 = 161)$.

Let z_t, $t = 0, 1, 2, \ldots$ be the keystream generated by the cipher after running the state initialisation algorithm of WG-7. Assume also that f is the nonlinear filter function WG illustrated in Figure 3.1. We consider f as a nonlinear map defined from $GF(2^7) \rightarrow GF(2)$. As the output bit is calculated on the contents of the last register or bits from 154 to 160, we denote this by

$$f(T(s_0, \ldots, s_{160})),$$

where $T(s_0, \ldots, s_{160})$ extracts the 7-bit content of the last register. So, we can establish the following system of relations for the cipher:

$$\begin{cases} z_0 = f(T(s_0, \ldots, s_{160})) \\ z_1 = f(T(L(s_0, \ldots, s_{160}))) \\ \ldots \\ z_t = f(T(L^t(s_0, \ldots, s_{160}))) \end{cases} \tag{3.16}$$

where $f(T(L^t(s_0, \ldots, s_{160})))$ indicates the output keystream at the clock t, generated by the stream cipher. Now, the cryptanalytic problem can be converted into the problem of solving a system of nonlinear equations (refer to [5, 36–38, 40]).

Algebraic attack on WG-7

Based on the algebraic attack scenario explained in 2.2.4, attacker can check possibility of the attack. The function f is of degree 5. The number N of monomials of degree smaller than or equal to 5 is

$$N = \sum_{i=1}^{5} \binom{161}{i} \approx \binom{161}{5} = 2^{29.65}.$$

Each of these monomials can be considered as a new variable, and then the attacker can solve the nonlinear system with $\approx 2^{29.65}$ equations and time complexity $\approx 2^{29.65\times\log_2 7}$ by the Gaussian elimination method. Consequently, the complexity of the attack is larger than the exhaustive key search.

An important idea to improve the efficiency of the above attack is to reduce the degree of the equations. To this end, the attacker tries to find an annihilator function so that $f \cdot g = 0$ and $\deg g < \deg f$ based on the steps explained in Section 2.2.4. The algebraic normal form (ANF) of f is as follows:

$$\begin{aligned}
&f(x_1, ..., x_7) = x_1 + x_1x_3 + x_2x_3 + x_4 + x_1x_4 + x_2x_4 + \\
&x_1x_2x_4 + x_3x_4 + x_1x_3x_4 + x_1x_2x_3x_4 + x_1x_3x_5 + x_4x_5 + x_1x_2x_4x_5 + \\
&x_1x_2x_3x_4x_5 + x_6 + x_2x_6 + x_1x_2x_6 + x_1x_2x_3x_6 + x_1x_2x_4x_6 + x_1x_2x_3x_4x_6 + \\
&x_1x_5x_6 + x_3x_5x_6 + x_1x_4x_5x_6 + x_3x_4x_5x_6 + x_7 + x_2x_7 + x_1x_2x_7 + x_2x_3x_7 + \\
&x_1x_4x_7 + x_1x_2x_4x_7 + x_1x_2x_3x_4x_7 + x_5x_7 + x_1x_5x_7 + x_1x_3x_5x_7 + x_1x_2x_3x_5x_7 + \\
&x_2x_4x_5x_7 + x_2x_3x_4x_5x_7 + x_6x_7 + x_1x_2x_6x_7 + x_1x_3x_6x_7 + x_1x_2x_3x_6x_7 + \\
&x_2x_4x_6x_7 + x_1x_3x_4x_6x_7 + x_2x_3x_4x_6x_7 + x_5x_6x_7 + x_2x_5x_6x_7 + x_1x_2x_5x_6x_7 + \\
&x_2x_3x_5x_6x_7 + x_1x_4x_5x_6x_7 + x_3x_4x_5x_6x_7.
\end{aligned}$$

The best annihilator is of the form:

$$\begin{aligned}
g(x_1, ..., x_7) \quad = \quad & 1 + x_1 + x_3 + x_1x_2x_3 + x_4 + x_1x_4 + x_2x_4 + x_1x_2x_4 + x_3x_4 + x_1x_3x_4 + \\
& x_1x_3x_5 + x_4x_5 + x_1x_4x_5 + x_3x_4x_5 + x_6 + x_1x_6 + x_2x_6 + x_1x_2x_6 + x_3x_6 + \\
& x_2x_3x_6 + x_7 + x_3x_7 + x_1x_3x_7 + x_2x_3x_7 + x_4x_7 + x_2x_4x_7 + x_2x_3x_4 + \\
& x_3x_4x_7 + x_3x_5x_7 + x_4x_5x_7 + x_6x_7 + x_1x_6x_7 + x_2x_6x_7 + x_3x_6x_7.
\end{aligned}$$

This means that the attacker can reduce the degree of the relations to 3 and solve them with time complexity $\approx \binom{161}{3}^{\log_2 7} = 2^{54.36}$ and memory complexity $\binom{161}{3} = 2^{19.38}$. It is obvious that the designers of WG-7 have ignored this attack, which breaks the cipher with memory complexity smaller than 2^{24}.

Improved Attack on WG-7

To apply the fast algebraic attack described in 2.2.4 to the WG-7 stream cipher, we found the boolean functions g and h, which are demonstrated as follows:

$$g(x_1, ..., x_7) = 1 + x_1 + x_3 + x_7.$$
$$h(x_1, ..., x_7) = x_1x_2x_3 + x_4 + x_1x_4 + x_2x_4 + x_1x_2x_4 + x_3x_4 +$$
$$x_1x_3x_4 + x_2x_3x_4 + x_1x_3x_5 + x_4x_5 + x_1x_4x_5 +$$
$$x_3x_4x_5 + x_6 + x_1x_6 + x_2x_6 + x_1x_2x_6 + x_3x_6 + x_2x_3x_6 +$$
$$x_3x_7 + x_1x_3x_7 + x_2x_3x_7 + x_4x_7 + x_2x_4x_7 + x_3x_4x_7 +$$
$$x_3x_5x_7 + x_4x_5x_7 + x_6x_7 + x_1x_6x_7 + x_2x_6x_7 + x_3x_6x_7.$$

The data complexity of the fast algebraic attack on WG-7 is $\binom{161}{d} = \binom{161}{3}$ and the time complexity is approximately $\binom{161}{e}^{\log_2 7} \approx \binom{161}{1}^{2.807}$. Table 3.2 summarises the results of our attacks. The trivial attack in Table 3.2 shows the time complexity when attacker has not used any algebraic technique to improve the attack. The attack finds the internal state of cipher, then attacker can clock the register back to recover the secret key.

TABLE 3.2: Comparison of different algebraic attacks against WG-7

	Attack type	n	d	e	Time Complexity	Data Complexity	Memory	Pre-Computation
1	Trivial Attack	161	5	-	$2^{83.02}$	$2^{29.65}$	-	-
2	Algebraic Attack	161	3	-	$2^{54.36}$	$2^{19.38}$	-	-
3	Fast Algebraic Attack	161	1	3	$2^{26.73}$	$2^{19.38}$	$2^{14.66}$	$2^{26.87}$

3.3 Conclusions

In this chapter, the security of the WG-7 stream cipher has been investigated. We have shown that a distinguishing attack works with a high probability of success after observing $2^{13.5}$ keystream bits. Additionally, the key-recovery attack which has been described can recover the secret key with time complexity of about 2^{27} and data complexity of $2^{19.38}$. The presented results have shown that the WG-7 stream cipher is not secure and, therefore, it is not recommended for use.

4

Security evaluation of Rakaposhi stream cipher

The Rakaposhi stream cipher was designed by Cid, Kiyomoto, and Kurihara in 2009 [35]. The cipher is based on a nonlinear feedback shift register and a dynamic linear shift register (DLFSR). The design was crafted to be suitable for lightweight implementations, where computing, power and time resources are in short supply. The cipher claims 128-bit security and has been designed to complement the eStream portfolio for hardware-oriented stream ciphers. The designers of the cipher claim that the Rakaposhi is an efficient synchronous stream cipher that resists all known attacks, and they conjecture that it is also secure against other, yet unknown, attacks.

This chapter analyses the Rakaposhi cipher and shows its weaknesses. We particularly:

- examine the resistance of the cipher against a related-key attack, where the adversary can access related pairs (IV, K),
- study the security implications when the NLFSR enters a short cycle,
- investigate the security level when the DLFSR enters a short cycle.

Related Work

The related-key attack is studied in the context of the Rakaposhi cipher and its initialisation procedure. A similar analysis can be found in [50, 53, 78] but in a different context. The related-key attack can be seen as a member of the differential cryptanalysis toolbox. We use the slide attack published by Cannière et al. in [43] to launch the related-key attack. The second part of the chapter is influenced by the paper of Zhang and Wang [143], in which the authors study the security of the Grain stream cipher [70, 71]. While working on this topic, we have become aware of a paper [74] that shows a similar analysis of the initialisation procedure of Rakaposhi, but in our study we have improved dramatically the efficiency of the key-recovery attack and we have also identified two classes of weak states.

This chapter is structured as follows. Section 4.1 describes briefly the Rakaposhi stream cipher. Section 4.2 presents the weaknesses of the cipher and investigates the security of the initialisation procedure under the related-key attack. Section 4.3 discusses the security implications when one of the registers (either the NLFSR or the DLFSR) enters a short cycle. Section 4.4 summarises the results.

4.1 Description of Rakaposhi Stream Cipher

The Rakaposhi stream cipher consists of the following three building blocks (see Figure 4.1) :

- a 128-bit NLFSR also called register A,
- a 192-bit DLFSR also called register B,
- a nonlinear function NLF

The NLSFR register A is defined by its feedback function:

$$\begin{aligned} g(x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9) &= x_1x_3x_9 \oplus x_1x_7x_9 \oplus x_5x_8 \oplus x_2x_5 \oplus \\ &\quad x_3x_8 \oplus x_2x_7 \oplus x_9 \oplus x_8 \oplus x_7 \oplus x_6 \oplus \\ &\quad x_5 \oplus x_4 \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0 \oplus 1, \end{aligned}$$

where $a_{t+128} = g(a_t, a_{t+6}, a_{t+7}, a_{t+11}, a_{t+16}, a_{t+28}, a_{t+36}, a_{t+45}, a_{t+55}, a_{t+62})$ and a_{t+i} is the i^{th} bit of register A at clock t.

The DLFSR register B is controlled by two bits (c_0, c_1) taken from the state of the NLFSR. The bits select one of four possible characteristic polynomials of the DLFSR.

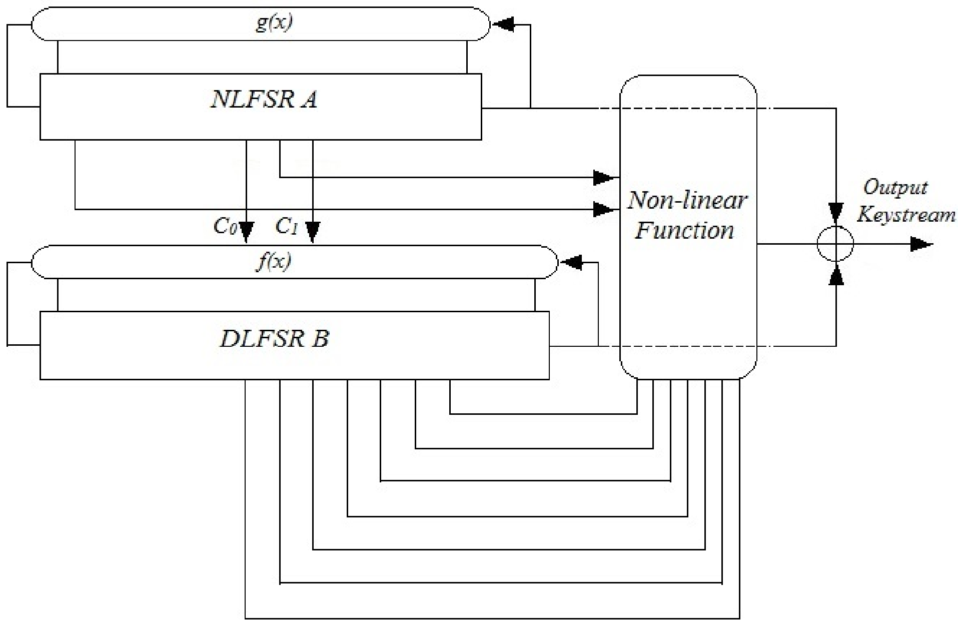

FIGURE 4.1: Rakaposhi stream cipher

The form of the polynomials is as follows:

$$\begin{aligned} f(x) &= x_{192} \oplus x_{176} \oplus c_0 x_{158} \oplus (1 \oplus c_0) x_{155} \oplus c_0 c_1 x_{136} \oplus \\ & c_0(1 \oplus c_1) x_{134} \oplus c_1(1 \oplus c_0) x_{120} \oplus (1 \oplus c_0)(1 \oplus c_1) x_{107} \oplus \\ & x_{93} \oplus x_{51} \oplus x_{49} \oplus x_{41} \oplus x_{37} \oplus x_{14} \oplus 1, \end{aligned} \tag{4.1}$$

where the bits (c_0, c_1) are the 42^{th} and 90^{th} bits of register A at clock t, respectively. The recursive relation for the DLFSR is as follows:

$$\begin{aligned} b_{t+192} &= b_t \oplus b_{t+14} \oplus b_{t+37} \oplus b_{t+41} \oplus b_{t+49} \oplus b_{t+51} \oplus b_{t+93} \oplus \\ & \overline{c_0} \cdot \overline{c_1} \cdot b_{t+107} \oplus \overline{c_0} \cdot c_1 \cdot b_{t+120} \oplus c_0 \cdot \overline{c_1} \cdot b_{t+134} \oplus c_0 \cdot c_1 \cdot b_{t+136} \oplus \\ & \overline{c_0} \cdot b_{t+155} \oplus c_0 \cdot b_{t+158} \oplus b_{t+176} \end{aligned} \tag{4.2}$$

where $\overline{c_i} = 1 \oplus c_i$ denotes the inversion of c_i and b_{t+i} is the i^{th} bit of B at clock t.

Rakaposhi uses a nonlinear filtering function $NLF : GF(2^8) \rightarrow GF(2)$, which is based on the AES S-Box. The NLF function is a balanced Boolean function and its algebraic degree is 7. NLF takes 8-bit inputs (2 bits from A and 6 bits from B) and outputs

$$s_t = NLF(a_{t+67}, a_{t+127}, b_{t+23}, b_{t+53}, b_{t+77}, b_{t+81}, b_{t+103}, b_{t+128}),$$

where the two bits a_{t+67}, a_{t+127} are taken from A and the other bits from B. Finally, the keystream output is generated by a linear combination of the outputs of both registers

A and B with the output of the NLF function. The reader interested in more detail is referred to the original paper [35].

4.1.1 Initialisation Procedure

The goal of the initialisation procedure is to mix IV and the secret key K. Assume that $IV = [iv_0, \cdots, iv_{191}]$ and $K = [k_0, \cdots, k_{127}]$. K and IV are loaded to the NLFSR and DLFSR respectively, so

$$\begin{aligned} a_i &= k_i \quad \text{for} \quad 0 \leq i \leq 127 \\ b_j &= iv_j \quad \text{for} \quad 0 \leq j \leq 191, \end{aligned}$$

where the bits of registers A and B are a_i and b_j, respectively. Registers A and B are then clocked 448 times without producing any output keystream bits. This stage is divided into two phases:

Phase 1: The output of the NLF is linearly combined with the feedback of register B for the first 320 clocks.

Phase 2: The output of the NLF is linearly combined with the feedback of register A for the next 128 clocks.

After finishing Phase 2, the cipher starts producing keystream outputs.

4.2 Cryptanalysis of Rakaposhi Stream Cipher

Now we show how we can launch the distinguishing and key-recovery attacks on the Rakaposhi cipher. The attacks use a sliding property of the cipher. An interesting property of the proposed attacks is that their complexities are not affected by the number of clocks, which the cipher performs during the initialisation process. This means that the attacks works even if the number of clocks is increased.

4.2.1 Properties of Rakaposhi Cipher

We present some cryptographic properties of the Rakaposhi stream cipher that corroborate the proposed attacks.

1. The secret key and IV are loaded in two registers A and B, respectively. Consequently, at clock $t = 0$, A contains K and B contains IV.

2. The initialisation procedure applies the same primitives that are used during the keystream generation stage. This implies that the initialisation for the key and IV is similar to the initialisation for the key and IV when they are shifted by one position. We refer to this characteristic as the sliding property.

3. Register A (NLFSR) has a short cycle of length '1'; when the state of A becomes all ones, then A stays in this state forever.

4. Register B (DLFSR) has a short cycle of the length '1'; when the state of B becomes all zeros, then B stays in this state forever.

The first two properties mean that the adversary may find related (K, IV) pairs, which produce keystream outputs that are shifted. These properties lead the adversary to a distinguishing attack that needs only four related (K, IV) pairs, and a key-recovery attack which recovers all bits of the secret key K after observing 2^9 related (K, IV) pairs.

The third and fourth properties can be exploited by the adversary to distinguish the cipher from a truly random binary source and recover the internal state of the cipher and finally the corresponding secret key. The proposed attacks recover the secret key with time complexity of $2^{63.87}$ and 2^{54}.

4.2.2 Related-Key Attack on Rakaposhi

In our sliding attack we assume that we have two related pairs (K, iv) and $(\widehat{K}, \widehat{iv})$. Consider the initialisation procedure for the two pairs. Let $K = (k_0, \cdots, k_{127})$ and $iv = (iv_0, ..., iv_{191})$ be loaded into the registers A and B, respectively. Denote the states of registers A and B at the clock t by A^t and B^t, respectively. The evolution of states over time is described below.

$$
\begin{array}{lll}
 & A^0 = [k_0, \cdots, k_{127}] & B^0 = [iv_0, \cdots, iv_{191}] \\
 & A^1 = [k_1, \cdots, k_{127}, a_{128}] & B^1 = [iv_1, \cdots, iv_{191}, b_{192}] \\
 & \vdots & \vdots \\
\xrightarrow{\text{Phase 2 of Initialisation}} & A^{320} = [a_{320}, \cdots, a_{448}] & B^{320} = [b_{320}, \cdots, b_{512}] \\
 & \vdots & \vdots \\
\xrightarrow{\text{Initialisation finished}} & A^{448} = [a_{448}, \cdots, a_{576}] & B^{448} = [b_{448}, \cdots, b_{640}] \\
\xrightarrow{\text{Key Generation started}} & A^{449} = [a_{449}, \cdots, a_{577}] & B^{449} = [b_{449}, \cdots, b_{641}] \\
 & A^{450} = [a_{450}, \cdots, a_{578}] & B^{450} = [b_{450}, \cdots, b_{642}]
\end{array}
$$

The keystream output bits z_i, where $i \geq 0$, are computed as follows:

$$\begin{aligned}
z_0 &= a_{449} \oplus b_{449} \oplus NLF(a_{448+67}, a_{448+127}, b_{448+23}, b_{448+53}, b_{448+77}, b_{448+81}, b_{448+103}, b_{448+128}) \\
z_1 &= a_{450} \oplus b_{450} \oplus NLF(a_{449+67}, a_{449+127}, b_{449+23}, b_{449+53}, b_{449+77}, b_{449+81}, b_{449+103}, b_{449+128}) \\
&\vdots
\end{aligned}$$

The relation between keystreams generated by the cipher when initialised by the related pairs is described by the following theorem.

Theorem 3 *Given two pairs (K, iv) and $(\widehat{K}, \widehat{iv})$, where $K = (k_0, \cdots, k_{127})$ and $iv = (iv_0, ..., iv_{191})$. then, if the pair $(\widehat{K}, \widehat{iv})$ satisfies the following equations,*

$$\begin{cases} \widehat{k_i} &= k_{i+1} \quad 0 \leq i \leq 126, \quad \hat{k}_{127} = a_{128} \\ \widehat{iv_i} &= iv_{i+1} \quad 0 \leq j \leq 190, \quad \widehat{iv}_{191} = b_{192} \end{cases} \tag{4.3}$$

the keystream output bits $\widehat{z_i} = z_{i+1}$ for $i \geq 0$ with probability 2^{-2}.

Proof 3 *By satisfying Equation (4.3), the internal states of $[A^{320}, B^{320}]$ are equal to $[\widehat{A}^{319}, \widehat{B}^{319}]$. But, at the next clock, the states may not be identical because the state $[\widehat{A}^{320}, \widehat{B}^{320}]$ is still at the first step while $[A^{321}, B^{321}]$ is running at the second step. If $\hat{b}^{512} = b^{511}$, which occurs with probability $1/2$, then*

$$[\widehat{A}^{319}, \widehat{B}^{319}] = [A^{320}, B^{320}].$$

The same argument is valid for the states $[A^{448}, B^{448}]$ and $[\widehat{A}^{447}, \widehat{B}^{447}]$. The states are identical when $\hat{a}^{446} = b^{447}$, which also happens with probability $1/2$. Consequently, $\widehat{z_i} = z_{i+1}$ for $i \geq 0$ with probability 1/4. □

Table 4.1 presents some (K, IV) pairs which produce shifted identical keystream outputs. According to Theorem 3, the adversary can use this weakness to generate the same keystreams but $l-$bit shifted keystream outputs by defining related (K, IV) pairs with probability $2^{-2 \cdot l}$.

The discovered weakness allows the adversary to distinguish the cipher from a random bit generator. Assume that the adversary can apply related (K, IV) pairs, but they do not know the exact values of the secret key. Then, after applying m ($m \gg 4$) different (randomly generated) related (K, IV) pairs, on the average $m/4$ of the generated keystream outputs have identical sequences with just one bit shift.

4.2.3 Recovery of Secret Keys

Now we propose a key-recovery attack that exploits the sliding property of pairs (K, IV). We show an algorithm that allows us to recover the 128-bit key after about 2^9 initialisation operations with related (K,IV) pairs. The attack can find the secret key with probability close to one.

Assume that both (K, IV) and $(\widehat{K}, \widehat{IV})$ generate almost identical keystream bits, where the second keystream is a copy of the first keystream shifted by one bit. At clock $t = 1$, the first generated bit is b_{192}, which is equal to:

$$\begin{aligned} b_{192} = {} & b_0 \oplus b_{14} \oplus b_{37} \oplus b_{41} \oplus b_{49} \oplus b_{51} \oplus b_{93} \oplus (1 \oplus c_0)(1 \oplus c_1)b_{107} \\ & \oplus (1 \oplus c_0)c_1 b_{120} \oplus c_0(1 \oplus c_1)b_{134} \oplus c_0 c_1 b_{136} \oplus (1 \oplus c_0)b_{155} \oplus c_0 b_{158} \oplus b_{176} \\ & \oplus NLF(a_{67}, a_{127}, b_{23}, b_{53}, b_{77}, b_{81}, b_{103}, b_{128}), \end{aligned}$$

where $c_0 = a_{41}$ and $c_1 = a_{89}$. Since the contents of b_i $(0 \leq i \leq 191)$ are known and can be chosen by the adversary, then Equation (4.4) is a nonlinear relation based on only

4 unknown variables $a_{41}, a_{89}, a_{67}, a_{127}$. We now take a closer look at Equation (4.4):

$$
\begin{aligned}
b_{192} = {} & b_0 \oplus b_{14} \oplus b_{37} \oplus b_{41} \oplus b_{49} \oplus b_{51} \oplus b_{93} \oplus (1 \oplus a_{41})(1 \oplus a_{89})b_{107} \\
& \oplus (1 \oplus a_{41})a_{89}b_{120} \oplus a_{41}(1 \oplus a_{89})b_{134} \oplus a_{41}a_{89}b_{136} \oplus (1 \oplus a_{41})b_{155} \oplus a_{41}b_{158} \oplus b_{176} \\
& \oplus a_{67}a_{127}b_{23}b_{53}b_{77}b_{81}b_{103} \oplus a_{67}a_{127}b_{23}b_{53}b_{77}b_{81} \oplus a_{67}a_{127}b_{23}b_{53}b_{77}b_{103} \\
& \oplus a_{67}a_{127}b_{23}b_{53}b_{81}b_{103}b_{128} \oplus a_{67}a_{127}b_{23}b_{53}b_{81}b_{103} \oplus a_{67}a_{127}b_{23}b_{53}b_{81}b_{128} \\
& \oplus a_{67}a_{127}b_{23}b_{53}b_{81} \oplus a_{67}a_{127}b_{23}b_{53}b_{103}b_{128} \oplus a_{67}a_{127}b_{23}b_{77}b_{81}b_{103} \\
& \oplus a_{67}a_{127}b_{23}b_{77} \oplus a_{67}a_{127}b_{23}b_{81}b_{103} \oplus a_{67}a_{127}b_{23}b_{81}b_{128} \oplus a_{67}a_{127}b_{23}b_{128} \\
& \oplus a_{67}a_{127}b_{23} \oplus a_{67}a_{127}b_{53}b_{77}b_{81}b_{103}b_{128} \oplus a_{67}a_{127}b_{53}b_{77}b_{81}b_{128} \\
& \oplus a_{67}a_{127}b_{53}b_{77}b_{81} \oplus a_{67}a_{127}b_{53}b_{77}b_{128} \oplus a_{67}a_{127}b_{53}b_{77} \oplus a_{67}a_{127}b_{53}b_{103} \\
& \oplus a_{67}a_{127}b_{77}b_{81}b_{103}b_{128} \oplus a_{67}a_{127}b_{77}b_{81}b_{103} \oplus a_{67}a_{127}b_{77}b_{81}b_{128} \oplus a_{67}a_{127}b_{77}b_{103}b_{128} \\
& \oplus a_{67}a_{127}b_{77}b_{128} \oplus a_{67}a_{127}b_{81}b_{103}b_{128} \oplus a_{67}a_{127}b_{81}b_{103} \oplus a_{67}a_{127}b_{81} \oplus a_{67}a_{127}b_{103} \\
& \oplus a_{67}b_{23}b_{53}b_{77}b_{81}b_{103} \oplus a_{67}b_{23}b_{53}b_{77}b_{81}b_{128} \oplus a_{67}b_{23}b_{53}b_{77} \oplus a_{67}b_{23}b_{53}b_{81}b_{103}b_{128} \\
& \oplus a_{67}b_{23}b_{53}b_{81}b_{128} \oplus a_{67}b_{23}b_{53}b_{103} \oplus a_{67}b_{23}b_{77}b_{81}b_{103}b_{128} \oplus a_{67}b_{23}b_{81}b_{103} \\
& \oplus a_{67}b_{23}b_{81} \oplus a_{67}b_{23}b_{103}b_{128} \oplus a_{67}b_{23}b_{128} \oplus a_{67}b_{53}b_{77}b_{81}b_{103}b_{128} \oplus a_{67}b_{53}b_{77}b_{81}b_{103} \\
& \oplus a_{67}b_{53}b_{77}b_{81}b_{128} \oplus a_{67}b_{53}b_{77}b_{81} \oplus a_{67}b_{53}b_{77}b_{128} \oplus a_{67}b_{53}b_{81}b_{103}b_{128} \oplus a_{67}b_{53}b_{81} \\
& \oplus a_{67}b_{53}b_{103} \oplus a_{67}b_{53} \oplus a_{67}b_{77}b_{81}b_{103} \oplus a_{67}b_{77}b_{103}b_{128} \oplus a_{67}b_{81}b_{103} \oplus a_{67}b_{23}b_{53}b_{81}b_{103} \\
& \oplus a_{67}b_{103} \oplus a_{67} \oplus a_{127}b_{23}b_{53}b_{77} \oplus a_{127}b_{23}b_{53}b_{81}b_{103} \oplus a_{127}b_{23}b_{53}b_{81}b_{128} \oplus a_{127}b_{81} \\
& \oplus a_{127}b_{23}b_{53}b_{81} \oplus a_{127}b_{23}b_{53} \oplus a_{127}b_{23}b_{77}b_{81}b_{103} \oplus a_{127}b_{23}b_{77}b_{103} \oplus a_{127}b_{23}b_{77} \\
& \oplus a_{127}b_{23}b_{81} \oplus a_{127}b_{23} \oplus a_{127}b_{53}b_{77}b_{81}b_{103}b_{128} \oplus a_{127}b_{53}b_{77}b_{81}b_{128} \oplus a_{127}b_{53}b_{77}b_{103}b_{128} \\
& \oplus a_{127}b_{53}b_{77}b_{103} \oplus a_{127}b_{53}b_{77} \oplus a_{127}b_{53}b_{81}b_{103} \oplus a_{127}b_{53}b_{81} \oplus a_{127}b_{53}b_{103} \\
& \oplus a_{127}b_{53}b_{128} \oplus a_{127}b_{77}b_{81}b_{103}b_{128} \oplus a_{127}b_{77}b_{81}b_{128} \oplus a_{127}b_{81}b_{103} \oplus a_{127}b_{81}b_{128} \\
& \oplus a_{127}b_{103}b_{128} \oplus a_{127}b_{103} \oplus a_{67}a_{127} \oplus a_{127} \oplus NLF'(b_{23}, b_{53}, b_{77}, b_{81}, b_{103}, b_{128})
\end{aligned}
\tag{4.4}
$$

where NLF' is a Boolean function including all monomials of NLF in which variables a_{67}, a_{127} do not exist. Note that the adversary does not need to solve the equation. Instead, the adversary can recover four bits of the secret key by choosing appropriate bits for IVs. For example, if

$$
\begin{cases}
b_i = 0 & i \in \Phi \\
b_{158} = 1
\end{cases}
$$

where $\Phi = \{0, 14, 37, 41, 49, 51, 93, 107, 120, 134, 136, 155, 176, 23, 53, 77, 81, 103, 128\}$, then $b_{192} = a_{41}$. Consequently, $\widehat{iv}_{191} = k_{41}$. In this way, the adversary is able to retrieve the four secret key bits. The number of the required related pairs (K, IV) is 4.

On the average, to find the valid pairs, the adversary needs 16 pairs. In other words, to retrieve 4 secret key bits, the adversary should run the initialisation algorithm 16 times for the related (K, IV) pairs. Now, the adversary can keep going and continue the attack, finding consecutive 4-bit parts of the secret key. Finally, to determine the whole 128-bit secret key, the adversary needs to apply $512 = 32 \times 16$ related (K, IV) pairs on the average.

4.3 Weak (K, IV) Pairs

In this section we study the security implications of short cycles of two registers A and B. Note that the initialisation procedure takes K and IV, loads them to A and B respectively, and then the cipher is clocked 448 times. At the end of the initialisation, the cipher can be set in the following weak states:

- Register A contains all ones and the state loops forever. To identify the collection of pairs (K, IV) that leads to this state of A, it is enough to set $A = \mathbf{1}$ and to set B to an arbitrary 192-bit vector and clock backwards. This process will generate 2^{192} pairs (K, IV) and cause the initialisation to weak states.
- Register B contains all zeros and the state loops forever. Again, to identify the collection of pairs (K, IV) that leads to this state of B, it is enough to set $B = \mathbf{0}$ and to set A to an arbitrary 128-bit vector and clock backwards. This process will generate 2^{128} pairs (K, IV) and cause the initialisation to weak states.
- Both registers $A = \mathbf{1}$ and $B = \mathbf{0}$. There is a single pair of (K, IV) only. To identify it, set the registers appropriately and clock backwards. This case is not very interesting as it can be easily identified.

4.3.1 Weak (K, IV) Pairs Leading to $A = \mathbf{1}$

After the initialisation phase, it may happen that the pair (K, IV) leads to $A = \mathbf{1}$. An immediate consequence of this occurrence is that register A contains all ones and stays in this state for all clocks. The adversary is able to identify this case, and is also able to recover the weak pair (K, IV) that has led to $A = \mathbf{1}$. Clearly, if the adversary knows IV, then the task of finding K is easier.

Note that the cipher with register A in the state of all ones is equivalent to a 192-bit LFSR whose outputs are filtered by a nonlinear Boolean function h with a 6-bit input. The function h is the nonlinear function NLF with two bits set to ones (those that are coming from A). The function is a balanced function from $h : GF(2^6) \to GF(2)$ of degree 5 and nonlinearity 20 and is given below.

$$\begin{aligned} h(x_1, x_2, x_3, x_4, x_5, x_6) = {} & 1 \oplus x_1 \oplus x_1x_2 \oplus x_3 \oplus x_1x_3 \oplus x_1x_4 \oplus x_3x_4 \oplus x_2x_3x_4 \oplus x_5 \\ & \oplus x_1x_2x_5 \oplus x_2x_3x_5 \oplus x_1x_4x_5 \oplus x_3x_4x_5 \oplus x_1x_3x_4x_5 \oplus x_2x_3x_4x_5 \\ & \oplus x_1x_6 \oplus x_2x_6 \oplus x_1x_3x_6 \oplus x_1x_2x_3x_6 \oplus x_4x_6 \oplus x_1x_4x_6 \oplus x_5x_6 \\ & \oplus x_1x_2x_4x_6 \oplus x_3x_4x_6 \oplus x_1x_3x_4x_6 \oplus x_2x_3x_4x_6 \oplus x_1x_2x_3x_4x_6 \\ & \oplus x_2x_3x_5x_6 \oplus x_4x_5x_6 \oplus x_2x_4x_5x_6 \oplus x_1x_2x_4x_5x_6 \oplus x_2x_3x_4x_5x_6 \end{aligned}$$

The function can be approximated by a linear Boolean function $1 \oplus x_1 \oplus x_1 \oplus x_6$ with probability:

$$Pr(h = (1 + x_1 + x_2 + x_6)) = \frac{44}{64} = 0.6875 = 0.5 + 2^{-2.415}$$

The algebraic immunity of the function is 3 and the number of annihilators is 10. To recover the contents of register B, we may apply a basic algebraic attack, described in Section 2.2.4, that needs $2^{22.75}$ observations of the keystream bits and whose complexity is $2^{63.87}$. Once the adversary knows the contents of B at the end of the initialisation, they can clock backwards to recover the weak pair (K, IV).

4.3.2 Weak (K, IV) Pairs Leading to $B = \mathbf{0}$

The second class of weak (K, IV) pairs leads to the state with $B = \mathbf{0}$. In this case, register B stays in the zero state for all clocks. Consequently, all the outputs of the DLFSR are zeros, which is equivalent to removal of register B from the cipher. The goal of the adversary is to recover the pair (K, IV). Now we show that the adversary is able to recover the initial state (and the secret key by clocking NLFSR backwards) faster than in 2^{54} steps.

Note that the NLF function is now used with its 6 bits coming from register B set to zero. Consequently, the keystream output function is a linear combination of the least significant bit of register A with the output of the NLF function. The keystream output function is denoted by $\ell : \{0, 1\}^3 \to \{0, 1\}$ and is of the following form:

$$\ell(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_1x_2 \oplus x_3.$$

The function ℓ is a nonlinear balanced Boolean function of degree 2. One of the best approximations of ℓ is the linear function x_3. It is easy to check that

$$Pr(\ell = x_3) = \frac{6}{8} = 0.75 = 0.5 + 2^{-2}. \tag{4.5}$$

Distinguishing Attack

If $B = \mathbf{0}$, then the adversary may distinguish the generated keystream bits from a random bit generator. Consider the keystream output bits at clocks $t+0$, $t+6$, $t+7$, $t+11$, $t+16$, $t+28$, $t+36$, $t+45$, $t+55$, $t+62$. If we use the approximation (see Equation (4.5)) then we can write

$$Pr(z_{t+128} = g(z_{t+0}, z_{t+6}, z_{t+7}, z_{t+11}, z_{t+16}, z_{t+28}, z_{t+36}, z_{t+45}, z_{t+55}, z_{t+62})) \approx 0.502. \tag{4.6}$$

This means that the adversary requires around 2^{17} observations of the keystream output bits to tell apart the cipher from a random bit generator with negligible error probability.

Recovery Attack

To recover the pair (K, IV), the adversary may use the linear approximation of ℓ and try to guess the contents of A. The probability of the correct guess for the state is $(0.75)^{128} = 2^{-53.12}$, which is much smaller than the probability 2^{-128}. In other words, the cipher has at most 54 bits of security.

4.4 Summary

In this chapter, we analysed the initialisation algorithm of the Rakaposhi stream cipher. From observations about cryptographic weaknesses of the cipher, we discovered the so-called sliding property of the pairs (K, IV). This property can be exploited by launching distingushing and key-recovery attacks. We showed that there is a distinguishing attack that needs only four related (K, IV) pairs. Our key-recovery attack recovers all bits of the secret key K after observing 2^9 related (K, IV) pairs.

In the second part of this chapter, we studied the security of Rakaposhi when either register A or register B enters a short cycle at the end of the initialisation procedure. When register A loops in the all-ones state, then the adversary is able to recover the pair (K, IV). Rakaposhi in this case degenerates to a LFSR cipher with a nonlinear filter function. Thus the initial state of register B can be discovered by using an algorithm of time complexity $2^{63.87}$.

If register B enters the zero state at the end of the initialisation procedure, then we showed two efficient algorithms: one to distinguish Rakaposhi from a random bit generator and the other to recover the pair (K, IV). The distinguisher needs 2^{17} keystream bit observations. The key-recovery algorithm requires around 2^{54} operations. Note that this cryptographic weakness can be explored by the adversary when they have access to the cipher device and are allowed to play with the device by running it for different IVs.

Table 4.1: Shifted identical keystream outputs corresponding two related (K, IV) pairs

Pair	Key	IV	Output bits
1	10011011110011101010 00100000001110100110 00000001100010110001 00111101110111010000 01001001000000010011 00110010010100110101 11100111	0100011110000001010001000 1101011000000000001000000 1101110000111110011101010 1100001111100001100110011 1000101110110110000010100 1101100010001110000000100 1001001011110011010110101 01100010110010101	00000110000100011100111 11000001010001010100100 10010100001101111001100 10101010110100000011011 11001110010001000111100 0111100000011110101
	00110111100111010100 01000000011101001100 00000011000101100010 01111011101110100000 10010010000000100110 01100100101001101011 11001111	1000111100000010100010001 1010110000000000010000001 1011100001111100111010101 1000011111000011001100111 0001011101101100000101001 1011000100011100000001001 0010010111100110101101010 11000101100101011	00001100001000111001111 10000010100010101001010 00101000011011110011010 01010101101000000110111 10011100100010001111000 1111000000111101010
2	00000000010111111011 00111011001110010001 01110111011011001111 00101011011101100001 11001000110110000011 10010110101001111001 10110000	0011100010111000111011100 0101100001000100011110000 1100101010010111010110010 0010100000011010011000110 1001001101101110001110011 0101011111110010100001100 1110011101110000000011110 01000110010110101	01110100100110000001111 00110010110000101110101 11111001100001111011101 10011110000100100110101 11100100000111001010001 0001011011111011111
	00000000101111110110 01110110011100100010 11101110110110011110 01010110111011000011 10010001101100000111 00101101010011110011 01100001	0111000101110001110111000 1011000010001000111100001 1001010100101110101100100 0101000000110100110001101 0010011011011100011100110 1010111111100101000011001 1100111011100000000111100 10001100101101011	11101001001100000011100 11001011000010111010111 11110011000011110111011 00111100001001001101010 11001000001110010100010 00010110111101111111

5

Security analysis of linearly filtered NLFSRs

The one-time pad (OTP) is the only cipher that is unbreakable even for an adversary who has unlimited computational power. Stream ciphers try to mimic the OTP but, instead of a truly random sequence, they produce a pseudorandom sequence derived by a relatively short random sequence (also called the seed). This, however, has a profound impact on their security. Stream ciphers do not inherit the OTP unconditional security. Their security is conditional and depends on the difficulty of recovery of the seed from an observed keystream.

The main advantage of stream ciphers is that they can be implemented very efficiently both in software and hardware making them very popular in the telecommunication industry. They are extensively used in mobile communications, providing the basic security tool to ensure confidentiality and integrity of communication. Historically, the first stream ciphers were built using shift registers with a linear feedback. Linear feedback shift registers (LFSRs) modify their internal state by using a linear recursion. Stream ciphers based on LFSR are insecure, as the recovery of the internal state from an observed keystream is equivalent to solving a system of linear equations.

To increase security, stream ciphers are built using LFSRs combined with nonlinear components. The designs are tested and analysed thoroughly. Consequently, a collection of design criteria has been identified. The collection can be used by designers to create new stream ciphers, the security of which can be tested using a collection of cryptographic attacks. The most effective tests for stream ciphers include the correlation attacks [39, 60, 108, 121] and the algebraic attacks [5, 36, 38, 68].

A natural evolution in the design of stream ciphers is the introduction of nonlinear feedback shift registers (NLFSRs). NLFSRs can be seen as a generalisation of LFSRs, where the modification of the internal state is carried out using a nonlinear relation [63]. While the mathematics behind LFSRs is well understood, the theory of NLFSRs is in its infancy. There are many basic problems related to NLFSRs that are still open. For instance, we do not know how to determine efficiently the period, identify different sub-cycles, or find out the linear complexity of NLFSRs.

One could argue that the lack of understanding of the mathematics behind NLFSRs has led to proliferation of NLFSR-based stream ciphers, as they are perceived to be more secure than other designs. The finalists of the e-Stream project include the Trivium [30] and Grain [70] ciphers that are exploiting one or several NLFSRs combined with LFSRs. The security of a NLFSR filtered by a linear boolean function is investigated using algebraic and correlation attacks in [10, 57]. In particular, the authors of [10] show that a linearly filtered nonlinear feedback shift register (LF-NLFSR) can be translated to the well-known *filter generator* that uses a LFSR and a nonlinear filter function - see Figure 5.1.

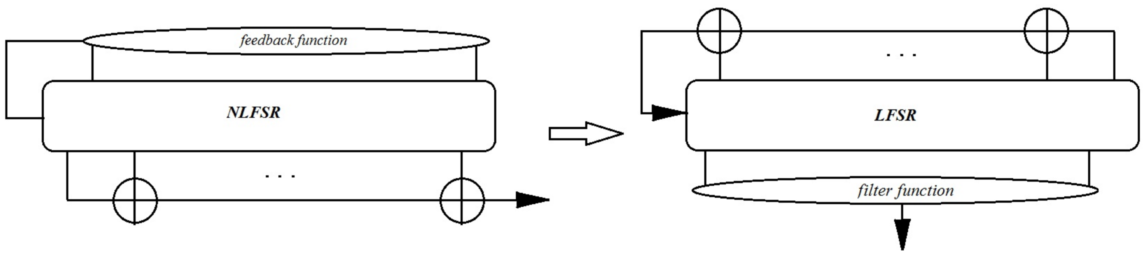

FIGURE 5.1: Translation of LF-NLFSR into LFSR with nonlinear filter

Our Contributions

This chapter investigates the design principles and security of steam ciphers built from LF-NLFSRs. First, we introduce a taxonomy of sequences generated by LF-NLFSR stream ciphers. Next, we examine the security of the LF-NLFSR stream ciphers against distinguishing attacks. Then, we identify criteria that need to be satisfied for a secure LF-NLFSR cipher. Finally, based on the proposed criteria, we show how to improve the time and data complexity of algebraic attacks on the LF-NLFSR ciphers presented in [10].

The chapter is organised as follows. Section 5.1 describes the LF-NLFSR cipher and introduces the main idea behind our distinguishing attack. Section 5.2 investigates security properties of stream ciphers whose LF-NLFSRs are chosen at random. The security properties of LF-NLFSRs associated with NLFSRs are studied in Section 5.3. In Section 5.4, we study the security of a stream cipher which is based on a linear combination of LF-NLFSRs. We show that this type of cipher may be vulnerable to distinguishing attacks. In Section 5.5, we suggest the design criteria for stream ciphers based on LF-NLFSRs. Finally, Section 5.6 concludes the chapter.

5.1 Description of LF-NLFSR

Pseudorandom sequences generated by an LFSR have been exhaustively studied and there is a good understanding of their statistical and cryptographic properties. A method to make the sequences immune against algebraic attacks is that the (linear) sequence generated by an LFSR be filtered by a nonlinear boolean function. The stream ciphers based on LFSRs with nonlinear filters have been analysed by many researchers. For instance, the works [25, 96, 114] present three recent designs of LFSR ciphers with nonlinear filters, and their security is analysed in [62, 118, 125].

The duality between LFSR stream ciphers with nonlinear filters and LF-NLFSR stream ciphers is investigated in [10, 57]. Given a LFSR stream cipher with a nonlinear filter, to determine the equivalent LF-NLFSR cipher one needs to find a nonlinear update function for the NLFSR and the linear filter function so that the ciphers generate the same keystreams. Formally, assume that a LF-NLFSR cipher consists of a n-bit NLFSR and a linear function L_f. Its operation can be described as follows:

$$\begin{aligned} s^t[i] &= s^{t-1}[i+1] \quad \text{for} \quad 0 \le i < n-1 \\ s^t[n-1] &= f(s^{t-1}[0], s^{t-1}[1], \cdots, s^{t-1}[n-1]), \end{aligned}$$

where $s^t[i]$ is i-th bit of the internal state of the NLFSR at clock t and f is a nonlinear feedback (state update) function. The output keystream is generated as follows:

$$z^t = L_f(s^{t-1}[0], s^{t-1}[1], \cdots, s^{t-1}[n-1])$$

In [10], this structure is investigated in terms of the algebraic and correlation attacks.

5.1.1 Attacks on LF-NLFSR

LF-NLFSR ciphers can be vulnerable to *distinguishing* and *state-recovery attacks*. The attacks can be more efficient if the linear filter function is chosen randomly. In this section, we propose a distinguishing attack against LF-NLFSR ciphers. In the attack, we exploit linear relations between output bits and the NLFSR internal state. We approximate the nonlinear feedback function by the nearest affine function and thus we establish probabilistic linear relations. After solving the relations, we are able to recover the internal state of the LF-NLFSR cipher. The attack works even when the NLFSR uses a highly nonlinear feedback function. The difference between our attack and the attack by Berbain et al. [10] is that our attack needs to approximate a small number of bits of the nonlinear feedback function only. In other words, our distinguisher works with a higher probability.

5.1.2 Distinguishing Attack on LF-NLFSR

In this section, we show how to apply distinguishing attacks on LF-NLFSR ciphers (see Figure 5.2). To make the presentation clearer, we start from a simple example.

Example 5.1.1 *Given a 7-bit NLFSR that generates keystream by using the linear boolean function $L_f(s_1, s_3, s_4, s_7) = s_1 \oplus s_3 \oplus s_4 \oplus s_7$, where $s_i \quad (i = 1, \cdots, 7)$ is the i-th bit of the NLFSR state. The feedback function f is the balanced nonlinear boolean function of the following form:*

$$f(s_1, s_2, s_3, s_5, s_6, s_7) = s_1 \oplus s_2 \oplus s_6 \oplus (s_3 \cdot s_5 \cdot s_7).$$

NLFSR generates nonlinear sequences of the period $T_7 = 2^7 - 1$ (see Figure 5.2) [48]. The output bits are generated as follows:

$$O_{i+1} = s_{i+1} \oplus s_{i+3} \oplus s_{i+4} \oplus s_{i+7} \tag{5.1}$$

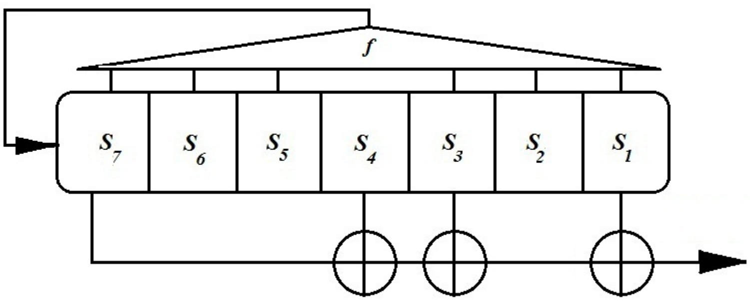

FIGURE 5.2: 7−bit LF-NLFSR cipher

Now, the adversary can replace bits in the internal state by a linear combination of the initial state and output bits. In Example 5.1.1, we can rewrite s_{i+7} $(i \geq 0)$ and get the following relations

$$
\begin{cases}
s_7 & = s_1 \oplus s_3 \oplus s_4 \oplus O_1 \\
s_8 & = s_5 \oplus s_4 \oplus s_2 \oplus O_2 \\
s_9 & = s_6 \oplus s_5 \oplus s_3 \oplus O_3 \\
s_{10} & = s_1 \oplus s_3 \oplus s_6 \oplus O_1 \oplus O_4 \\
s_{11} & = s_2 \oplus s_1 \oplus s_3 \oplus O_1 \oplus O_2 \oplus O_5 \\
s_{12} & = s_3 \oplus O_3 \oplus s_4 \oplus s_2 \oplus O_2 \oplus O_6 \\
s_{13} & = s_3 \oplus O_4 \oplus s_5 \oplus O_3 \oplus s_4 \oplus O_7 \\
s_{14} & = O_5 \oplus s_6 \oplus O_4 \oplus s_5 \oplus s_4 \oplus O_8 \\
s_{15} & = s_3 \oplus s_4 \oplus O_6 \oplus s_1 \oplus O_1 \oplus O_5 \oplus s_6 \oplus s_5 \oplus O_9 \\
s_{16} & = s_3 \oplus s_5 \oplus O_7 \oplus s_2 \oplus O_2 \oplus O_6 \oplus s_1 \oplus O_1 \oplus s_6 \oplus O_{10} \\
s_{17} & = s_6 \oplus O_8 \oplus O_3 \oplus O_7 \oplus s_2 \oplus O_2 \oplus s_1 \oplus O_1 \oplus O_{11} \\
s_{18} & = s_4 \oplus s_1 \oplus O_1 \oplus O_9 \oplus O_4 \oplus O_8 \oplus O_3 \oplus s_2 \oplus O_2 \oplus O_{12} \\
s_{19} & = s_2 \oplus s_3 \oplus s_5 \oplus O_2 \oplus O_3 \oplus O_4 \oplus O_5 \oplus O_9 \oplus O_{10} \oplus O_{13} \\
s_{20} & = s_3 \oplus s_4 \oplus s_6 \oplus O_3 \oplus O_4 \oplus O_5 \oplus O_6 \oplus O_{10} \oplus O_{11} \oplus O_{14} \\
s_{21} & = s_1 \oplus s_3 \oplus s_5 \oplus O_1 \oplus O_4 \oplus O_5 \oplus O_6 \oplus O_7 \oplus O_{11} \oplus O_{12} \oplus O_{15}
\end{cases} \tag{5.2}
$$

In addition to Equations (5.2), each generated internal state bit can be expressed by a linear approximation of the NLFSR feedback function. The approximation holds with the probability

$$
Pr(f(s_1, s_2, s_3, s_5, s_6, s_7) = s_1 \oplus s_2 \oplus s_6) = 1 - 2^{-3} = \frac{1}{2} + \frac{3}{8} \tag{5.3}
$$

By applying the linear approximations for the bits in the NLFSR internal state, the adversary can derive probabilistic linear relations, which are biased. For instance, the adversary can find a biased relation by combining O_2, O_3 and O_{15} as shown below:

$$
\begin{cases}
O_2 & = s_5 \oplus s_4 \oplus s_1 \oplus s_6 \\
O_3 & = s_6 \oplus s_5 \oplus s_3 \oplus s_2 \oplus s_1 \oplus s_4 \oplus O_1 \\
O_{15} & = s_2 \oplus s_3 \oplus O_2 \oplus O_3 \oplus O_4 \oplus O_7 \oplus O_8 \oplus O_{10} \oplus O_{11} \oplus O_{12} \oplus O_{13}.
\end{cases} \tag{5.4}
$$

Note that after linearly combining the relations, the unknown state bits are cancelled leaving the observable keystream bits that satisfy the following probabilistic linear relation:

$$
O_1 \oplus O_4 \oplus O_7 \oplus O_8 \oplus O_{10} \oplus O_{11} \oplus O_{12} \oplus O_{13} \oplus O_{15} = 0 \tag{5.5}
$$

We know that each relation of Equation (5.4) independently holds with probability $1-2^{-3}$. Therefore, after applying the Matsui piling up lemma, we obtain

$$Pr(O_1 \oplus O_4 \oplus O_7 \oplus O_8 \oplus O_{10} \oplus O_{11} \oplus O_{12} \oplus O_{13} \oplus O_{15} = 0) = \frac{1}{2} + (2^2 \cdot (\frac{3}{8})^3) = \frac{1}{2} + 2^{-2.245}. \tag{5.6}$$

Example 5.1.1 uses three linear approximations and establishes a distinguisher that tests the bias of the keystream bits. One might ask what an upper bound on the number of linear approximations for a given nonlinear function would be. Theorem 4 gives an answer.

Theorem 4 *Given a LF-NLFSR cipher built from an n-bit NLFSR with a feedback function f and a linear filter function L_f, if the best linear approximation of f is ℓ such that*

$$Pr(f = \ell) = \frac{1}{2} + \epsilon_f$$

then, having $n+1$ consecutive bits of the keystream outputs, there is at least one biased linear function.

Proof 4 *The proof can be found in [59].* □

The smallest number of output bits required to find a biased linear function (ℓ_p) depends on the linear filter function L_f and the feedback function f. In general, if all $n+1$ output bits are involved in ℓ_p (*e.g.* $n+1$ linear approximations), then the probability to find at least one ℓ_p biased function is

$$Pr(\ell_p) = \frac{1}{2} + 2^n \cdot \epsilon_f^{(n+1)}.$$

Note that Theorem 4 shows that the security of the cipher cannot be better than $\epsilon_f^{-2\cdot(n+1)}$. For each relation, we need to use at least one linear approximation with probability $P_L = 1/2 + \epsilon$. Assume that, with m linear equations, the adversary could find a biased relation for the output keystream bits with probability $P = 1/2 + (2^{m-1} \cdot \epsilon^m)$, then the attack is successful if

$$P < 2^{\Bbbk/2},$$

where $\Bbbk$ is the secret key space of the cipher. In other words, the bias $\epsilon' = 2^{m-1} \cdot \epsilon^m$ and hence the attack is faster than the exhaustive search $O(2^{\Bbbk})$ if $(\epsilon')^{-2} < 2^{\Bbbk/2}$.

There is a trend in the design of cryptographic components and systems, in which they are chosen at random. The main justification for this is the belief that random choice can protect the cryptographic system against new yet unknown attacks. In the next section, we analyse LF-NLFSR stream ciphers when both the linear filter function L_f and the nonlinear feedback function f are chosen at random.

5.2 Random LF-NLFSR Ciphers

A random LF-NLFSR cipher is a cipher whose linear filter function L_f and feedback function f are generated at random. More precisely, the nonlinear feedback function f is chosen at random from all balanced nonlinear functions. The linear filter function L_f is chosen randomly and uniformly from the set of all linear functions (excluding the constants).

5.2.1 Cryptanalysis of Random LF-NLFSR Ciphers

To analyse the security of random LF-NLFSR ciphers, we need two theorems. The first theorem evaluates the probability of choosing a set of p linearly independent q-tuples over $\mathbb{F}_2$ if the elements are drawn at random. We take advantage of the results from [87].

Theorem 5 ([87]) *Let $M_{q,q+p}$ be a $q \times (q+p)$ random matrix, over the finite field $\mathbb{F}_2$ where $-q \leq p \leq 0$. If $\rho(M)$ is the rank of matrix M, then we have,*

$$P(\rho(M_{q,q+p}) = q+p)) = \prod_{j=0}^{q+p-1} (1 - \frac{1}{2^{q-j}}), \quad -q \leq p \leq 0.$$

Proof 5 *Proof can be found in [87].* □

In general, the probability that a random $q \times (q+p)$ binary matrix $M_{q,q+p}$ is of the full rank q for $p \geq 0$ and a large q is:

$$P(\rho(M_{q,q+p}) = q) = \prod_{i=p+1}^{\infty} (1 - \frac{1}{2^i}), \quad p = 0, 1, \cdots .$$

An interesting observation shown in [26] is that, for a matrix defined as in Theorem 5, on the average, one would need two extra columns only to achieve the full rank. This result does not depend on q. For 7 or 8 extra columns, the probability of achieving the full rank is very close to 1.

Theorem 6 *Given a random binary matrix $M_{q,q+p}$ whose entries are chosen independently and uniformly, where $-q \leq p \leq 0$, then the probability that the rank of matrix M is less than $q+p$ is:*

$$P(\rho(M_{q,q+p}) < q+p) = 1 - P(\rho(M_{q,q+p}) = q+p)) = 1 - \prod_{j=0}^{q+p-1} (1 - \frac{1}{2^{q-j}}), \quad -q \leq p \leq 0.$$

Proof 6 *The rank of matrix M is at most to* $\min(q, p+q) = p+q$. *Therefore, the probability that the rank of matrix M is less than $q+p$ is* $1 - P(\rho(M_{q,q+p}) = q+p))$. *According to Theorem 5, the probability is* $1 - \prod_{j=0}^{q+p-1}(1 - \frac{1}{2^{q-j}})$, *where* $-q \leq p \leq 0$. □

Using Theorems 5 and 6, one can find the lower bound on the bias of linear approximations for random LF-NLFSR ciphers.

Theorem 7 *Given m linear approximations, then to find at least one linear biased relation with high probability, the number N_m of observed keystream bits should satisfy*

$$\pi(n,m)^{-1} = \binom{N_m}{m},$$

where $\pi(n,m)$ is the probability of finding at least one linear dependency for the corresponding matrix of an n-bit random LF-NFLSR cipher.

Proof 7 *Using Theorem 6, the probability of finding at least one linear dependency for the corresponding matrix of a n-bit random LF-NLFSR cipher can be computed as*

$$\pi(n,m) = 1 - \prod_{j=0}^{n-m-1}(1 - \frac{1}{2^{n-j}}),$$

where m is the number of rows. So, the number of $m \times n$ matrices which should be checked to find at least one linear dependency with probability near to one is $\frac{1}{\pi(n,m)}$. *The adversary needs to check all combinations of m linear equations from the required keystream bits (N_m), e.g.*

$$\pi(n,m)^{-1} = \binom{N_m}{m}$$

□

For a 64-bit random LF-NLFSR cipher, Theorem 7 states that the probability of finding a linear biased relation by applying linear approximation for two and four output bits is 2^{-64} and $2^{-61.19}$, respectively. The required number of keystream bits in order to apply the attack is $2^{32.48}$ and $2^{21.25}$, respectively.

We can consider that the matrices might have the properties of random matrices even if the feedback/filter functions are not chosen at random. In that case, the attack works even for schemes with non-random feedback/filter functions. Note that we consider balanced nonlinear functions and our assumptions do not limit us to a certain class of Boolean functions. If the adversary finds a linear biased relation using m linear approximations, then they simply need to approximate the feedback function m times, and the probability of finding a distinguisher is

$$Pr(distinguisher\ exists) = 1/2 + 2^{m-1} \cdot (\epsilon_f^m).$$

Therefore, the data complexity of the distinguishing attack is $O(\epsilon_f^{-2 \cdot m})$.

To apply a distinguishing attack on a random LF-NLFSR cipher, two main phases are needed: pre-processing and on-line. In the pre-processing phase, the adversary tries to find a distinguisher (or distinguishers). Theorem 7 determines the probability of finding it and the required data complexity of the pre-processing phase. The on-line phase consists of the distinguishing attack.

5.3 Ciphers based on LF-NLFSRs and LFSRs

Some stream ciphers are built from both LF-NLFSRs and LFSRs. The Grain stream cipher [70, 71] is an example. Figure 5.3 shows the overall structure of the Grain cipher, which has been extensively analysed (see [10, 47, 104] for example).

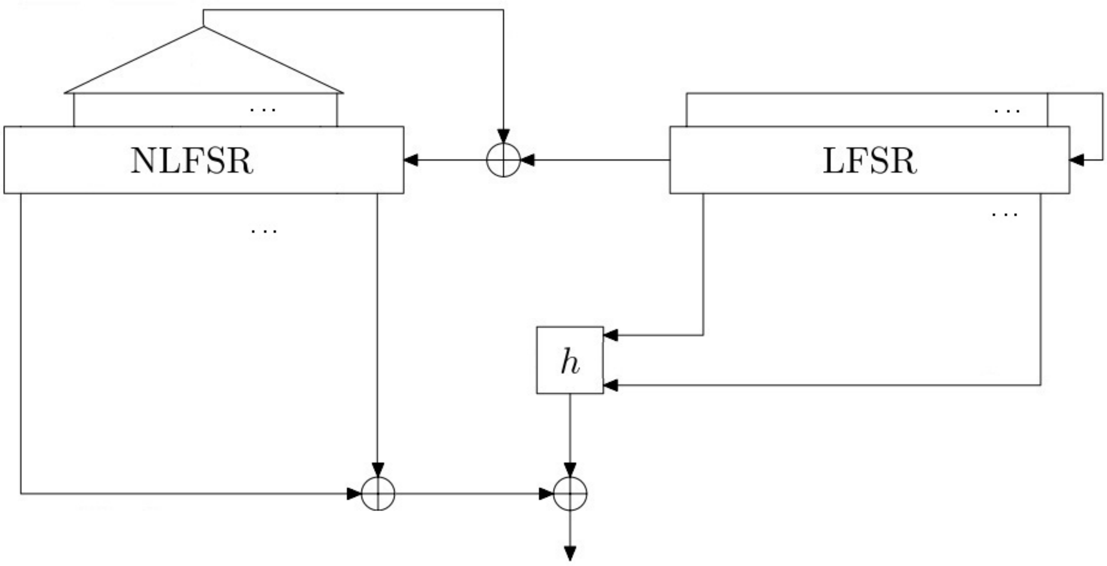

Figure 5.3: Grain cipher

5.3.1 Distinguishing Attack on Grain [10].

The structure of the Grain stream cipher gives rise to the following equations:

$$x_t = \bigoplus_{i \in \alpha} z_i \oplus \bigoplus_{j \in \beta} x_j \oplus \bigoplus_{k \in \gamma} y_k \oplus h^t(y_0, \cdots, y_m),$$

where x_i and y_i are the i^{th} bits of the internal states of the NLFSR and LFSR, respectively, and z_i are the keystream bits. The sets α, β and γ contain bit indices of the keystream bits and the NLFSR and LFSR state bits, respectively. The index sets are defined by the cipher structure. The bit $h^t(y_0, \cdots, y_m)$ is the output of filter function h at clock t. To apply a distinguishing attack on the Grain cipher, one should first replace both the nonlinear feedback function f and the function h by their best linear approximations. Next one needs to find a collection of approximations for which all the internal unobservable bits cancel themselves. In the best case, we can hope to find two such linear approximations, named z_x and z_y, such that

$$Pr(z_x \oplus z_y = 0) = \frac{1}{2} + 2^3 \cdot (\epsilon_f^{-2} \cdot \epsilon_h^{-2}),$$

where ϵ_f and ϵ_h indicate the biases of the linear approximations of the nonlinear feedback function f and nonlinear filter h, respectively. In this case, the security of the cipher against the distinguishing attack is $(2^3 \cdot (\epsilon_f^{-2} \cdot \epsilon_h^{-2}))^{-2}$.

5.4 Ciphers based on Linear Combination of LF-NLFSRs

LF-NLFSR ciphers can be extended in a natural way by allowing several LF-NLFSR structures which work in parallel, where the keystream combines bits generated by the LF-NLFSRs in some linear way. If the cipher keystream is a linear combination of several LF-NLFSRs, then we call it an LC-NLFSR. Assume that $O_1^t, \cdots, O_m^t$ are outputs of m distinct LF-NLFSRs at clock t. Then the keystream O^t of the cipher is produced as follows:

$$O^t = O_1^t \oplus O_2^t \oplus \cdots \oplus O_m^t$$

The LC-NLFSR structure is illustrated in Figure 5.4. Although, the attacks by Berbain et al. [10] cannot be applied to the LC-NLFSR cipher, we are going to show that the LC-NLFSR cipher is vulnerable to distinguishing attacks.

5.4.1 Distinguishing Attack on LC-NLFSRs

At SAC 2008, Berbain et al. presented their work [10] and mentioned a few open problems. One of them is the analysis of a linear combination of two LF-NLFSRs. In this section, we investigate the security of a linear combination of two LF-NLFSRs (LC-NLFSR). We present an analysis and criteria to design LC-NLFSR schemes.

Example 5.4.1 *Let N_1 and N_2 be two LF-NLFSRs (with nonlinear feedback functions g_1 and g_2 and linear filter functions L_1 and L_2, respectively), which are linearly combined to generate keystream bits (O_t at time $t \geq 0$). Let P_1 and P_2 be a linear*

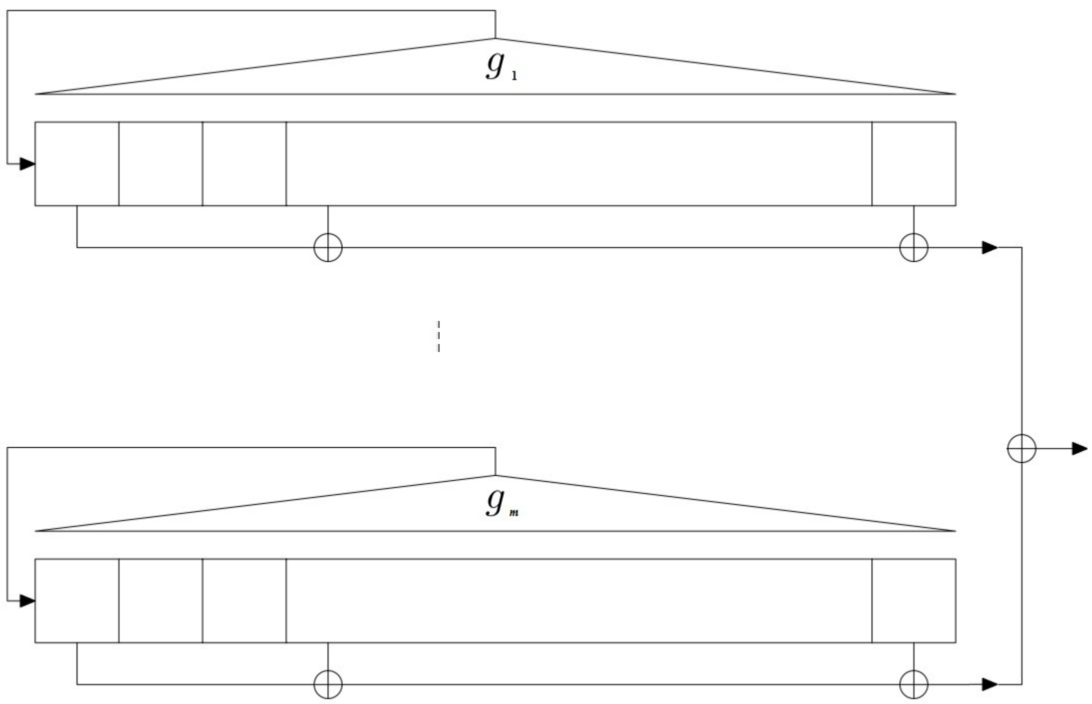

Figure 5.4: LC-NLFSR

combination of the internal states of N_1 *and* N_2*, respectively (see Figure 5.5). We know that:*

$$P_1^t \oplus P_2^t = O_t,$$

where P_i^t *is a linear filter of shift register* N_i *at clock* t *and* $i \in \{1, 2\}$*.*

Based on the method discussed in Section 5.1.1, we assume that the adversary is able to find two different biased linear relations $\lambda = \bigoplus_{i \in \{\phi_1\}} P_1^i$ and $\mu = \bigoplus_{i \in \{\phi_2\}} P_2^i$ for N_1 and N_2, respectively, where ϕ_1 and ϕ_2 represent the sets of effective coefficients. Clearly, the adversary cannot use the biased relations λ and μ to find a linear bias of the output bits, because the sets ϕ_1 and ϕ_2 are not necessarily the same. To find a linear biased relation based on the output keystream bits, we need to find linear biased relations derived from two LF-NLFSRs in the same instance. Consider linear biased relations λ and μ in the following polynomial forms:

$$\lambda(x) = c_0 + c_1 x + c_2 x^2 + \cdots + x^{l1}$$

$$\mu(x) = d_0 + d_1 x + d_2 x^2 + \cdots + x^{l2},$$

where $c_i, d_i \in \mathbb{F}_2$ are coefficients of the polynomials $\lambda(x)$ and $\mu(x)$ of degrees $l_1,\ l_2$, respectively, where $l_1 > N_1,\ l_2 > N_2$. To find a linear biased relation which is valid for

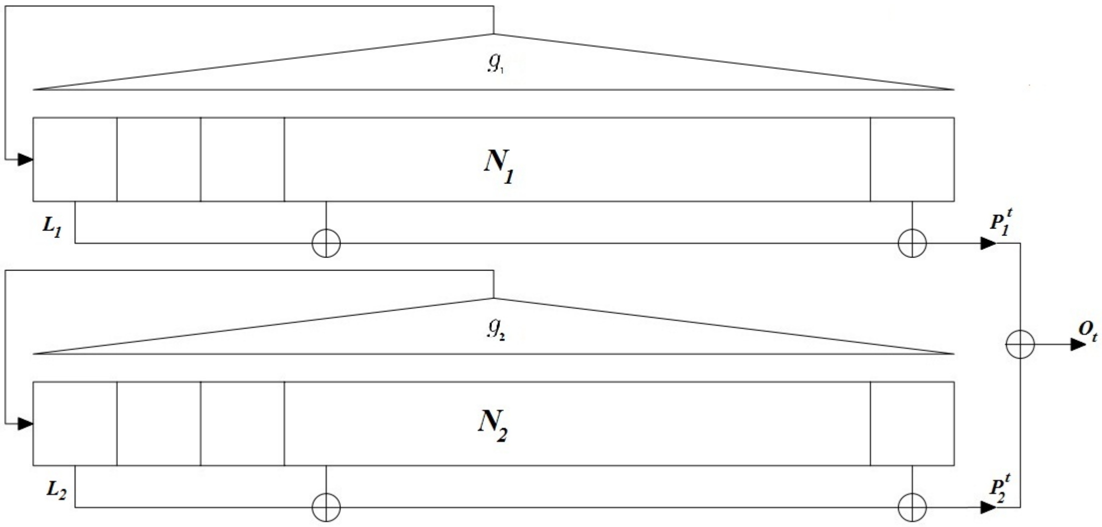

FIGURE 5.5: LC-NLFSR of Example 5.4.1

the output keystream bits, we can multiply $\lambda(x)$ and $\mu(x)$. In this case, the number of coefficients involved in the product will be higher than the number of coefficients involved in each polynomial $\lambda(x)$ and $\mu(x)$. So, it would be more efficient if we could find the polynomials with the lowest number of coefficients.

A different approach is to find the lowest degree polynomial $\Lambda(x)$ satisfying the following conditions:

1. $\lambda(x)|\Lambda(x)$
2. $\mu(x)|\Lambda(x)$

where $f(x)|g(x)$ means *g(x)* divides *f(x)*. Note that, in addition to LF-NLFSR and LC-NLFSR, the distinguishing attack can be successfully applied to m LF-NLFSRs that are linearly combined with n filter functions. For $m = 1$, $n = 1$, the authors of [10] have investigated the security of the cipher against algebraic and correlation attacks. However their attacks are not applicable for the cases when $m, n > 1$.

5.5 Linear Filter Properties

An interesting question is about the choice of a linear filter in LF-NLFSR and its impact on the cipher security. To answer this question, we need to introduce several concepts and two theorems from the work of Gammel et al [57]. We follow their notations. Let V be an infinite vector space whose elements belong to $\mathbb{F}_q$ and T be a linear operator defined on V by the relation $T\sigma = (s_{i+1})_{i=0}^{\infty}$, where $\sigma = (s_i)_{i=0}^{\infty}$ over V and $s_i \in \mathbb{F}_q$.

Further assume that g is a monic polynomial over $\mathbb{F}_q$. We call g a *characteristic polynomial* of σ if the operator $g(T)$ cancels out σ, i.e. $g(T)\sigma = 0$, where 0 stands for the zero vector of V. For any periodic sequence $\sigma \in V$,

$$J_\sigma = \{g \in \mathbb{F}_q[x] \ : \ g(T)\sigma = 0\}$$

is a non-zero ideal, known as the T-annihilator of σ, on $\mathbb{F}_q[x]$. The minimal polynomial of σ is a monic polynomial $m_\sigma \in \mathbb{F}_q[x]$ with $J_\sigma = (m_\sigma) = m_\sigma \mathbb{F}_q[x]$. Hence the characteristic polynomials of σ in $\mathbb{F}_q[x]$ are the monic polynomials, which are multiples of m_σ. Note that the degree of m_σ is defined as the linear complexity $L(\sigma)$ of σ. In [57], a method has been given to compute the minimal polynomial of a periodic sequence from a known characteristic polynomial and a suitable number of initial terms of the sequence.

Theorem 8 ([57]) *Let $A = (a_i)_{i=0}^{\infty}$ be a periodic binary sequence with minimal polynomial $p_a \in \mathbb{F}_2[x]$ and let $L_\alpha = \alpha_1 + \alpha_2 x + \cdots + \alpha_n x^{n-1}$ be a non-zero polynomial over $\mathbb{F}_2$. Then, the sequence*

$$B = (b_i)_{i=0}^{\infty} = (\alpha_1 a_{i+n} + \alpha_2 a_{i+n-1} + \cdots + \alpha_n a_i)_{i=0}^{\infty}$$

is periodic and its minimal polynomial is given by $p_b = \frac{p_a}{gcd(p_a, L_\alpha)}$.

Note that this theory allows us to derive new criteria for the design of LF-NLFSR ciphers. Let $A = (a_i)_{i=0}^{T}$ be a sequence generated by a NLFSR, with the minimal polynomial $p_a \in \mathbb{F}_2[x]$. To design a linear filter L_α achieving the maximum period of sequence A, p_a and L_α should be co-prime. This point shows the importance of designing an NLFSR with a single full period. Even if an NLFSR generates several long sequences, the linearly filtered output sequences may have a shorter period. Consequently, the best choice for a linear filter function is an irreducible polynomial. Theorem 9 describes the criterion.

Theorem 9 ([57]) *Let A be a periodic binary sequence generated by an $n-$bit NLFSR with period $2^n - 1$ (all nonzero n-bit states). The output sequences B have the same period and linear complexity if the canonical factorisation of the filter polynomial contains only irreducible factors equal to x or $x - 1$, or whose degrees do not divide n.*

5.5.1 Some Observations on the Grain LF-NLFSR

The Grain LF-NLFSR proposed in [71] is a modified version of the Grain cipher [70]. The output bits are generated by applying a linear filter function to the internal state of NLFSR. The 80-bit NLFSR has the feedback function f given as follows:

$$\begin{aligned}
s_{t+80} =& f(s_t, s_{t+1}, \cdots, s_{t+79}) \\
=& s_{t+62} \oplus s_{t+60} \oplus s_{t+52} \oplus s_{t+45} \oplus s_{t+37} \oplus s_{t+33} \oplus s_{t+28} \oplus s_{t+21} \\
& \oplus s_{t+14} \oplus s_{t+9} \oplus s_t \oplus s_{t+63}s_{t+60} \oplus s_{t+37}s_{t+33} \oplus s_{t+15}s_{t+9} \\
& \oplus s_{t+60}s_{t+52}s_{t+45} \oplus s_{t+33}s_{t+28}s_{t+21} \oplus s_{t+63}s_{t+45}s_{t+28}s_{t+9} \\
& \oplus s_{t+60}s_{t+52}s_{t+37}s_{t+33} \oplus s_{t+63}s_{t+60}s_{t+21}s_{t+15} \oplus s_{t+63}s_{t+60}s_{t+52}s_{t+45}s_{t+37} \\
& \oplus s_{t+33}s_{t+28}s_{t+21}s_{t+15}s_{t+9} \oplus s_{t+52}s_{t+45}s_{t+37}s_{t+33}s_{t+28}s_{t+21}
\end{aligned}$$

The keystream bits are generated by the following linear function:

$$O_t = s_{t+1} \oplus s_{t+2} \oplus s_{t+4} \oplus s_{t+10} \oplus s_{t+31} \oplus s_{t+43} \oplus s_{t+56} \oplus s_{t+63}.$$

Note that if the linear filter function is not designed properly, then the attacks by Berbain et al. [10] can be applied more efficiently. As mentioned in [10], the size of the blocks of equations of a constant degree is determined by the difference between the position of the highest tap index in the update function and the position updated by the feedback function. It means that $(80 - 63) = 17$ bits of the internal state can be represented as a linear combination of other internal state bits. This decreases the number of independent variables from 80 bits to 63. The algebraic technique, proposed in [10], keeps the degree of the corresponding system fixed and applies an algebraic attack to recover the internal state of the NLFSR. System 5.7 shows that every internal state bit s_i, $i \geq 80$, can be computed as a linear combination of the output bits and only 63 internal state bits ($i.e.$ s_i, $17 \geq i \geq 79$).

$$\begin{aligned}
s_{80} &= O_{17} \oplus s_{76} \oplus s_{60} \oplus s_{48} \oplus s_{27} \oplus s_{21} \oplus s_{19} \oplus s_{18} \\
s_{81} &= O_{18} \oplus s_{77} \oplus s_{61} \oplus s_{49} \oplus s_{28} \oplus s_{22} \oplus s_{20} \oplus s_{19} \\
s_{82} &= O_{19} \oplus s_{78} \oplus s_{62} \oplus s_{50} \oplus s_{29} \oplus s_{23} \oplus s_{21} \oplus s_{20} \\
s_{83} &= O_{20} \oplus s_{79} \oplus s_{63} \oplus s_{51} \oplus s_{30} \oplus s_{24} \oplus s_{22} \oplus s_{21} \\
&\vdots
\end{aligned} \tag{5.7}$$

The important point, which has not been investigated in [10], is the critical role played by the linear filter function in the security of the cipher. Now we are going to discuss an impact of the linear filter function on the security of the Grain LF-NLFSR cipher.

Lemma 10 *The number of the independent variables in System 5.7 is* 63.

Proof 8 *All new internal state bits* (s_{t+80}, $t \geq 0$) *generated by the update function can be written as* $(s_{17}, \cdots, s_{79})$ *variables. In other words, the number of the independent variables in System (5.7) is* $80 - 17 = 63$. □

Observation 1: Linear System (5.7) is generated by a specific polynomial called the generating polynomial. It is shown that the linear system inherits mathematical properties from the generating polynomial. If the polynomial is not primitive, then the linear equations are repeated with period less than $2^{80-17}-1$. Note that, because of dependency of the new generated variables on the variables $(s_{17}, \cdots, s_{79})$ and output bits $(O_t, \;\; t \geq 0)$, the new variables may not be exactly repeated, but the linear combinations of the independent variables are the same. Consequently, the linear complexity of the combination of the output bits decreases.

Assuming that the period of repetition of the linear relations of $(s_{17}, \cdots, s_{79})$ is T, then O_t and O_{t+T} satisfy the following relation:

$$O_t \oplus O_{t+T} = \bigoplus_{\tau=0}^{T} \alpha_\tau O_{t+\tau}$$

where $\alpha_\tau \in \mathbb{F}_2$ depends on the linear filter function.

Our considerations are illustrated below.

Example 5.5.1 *In Example 5.1.1, the period of the NLFSR state is $T_7 = 2^7 - 1$, but one can find a repetition of linear equations in the internal state with a period less than T_7. For instance, we have:*

$$\begin{cases}
s_7 & = s_1 \oplus s_3 \oplus s_4 \oplus O_1 \\
s_8 & = s_5 \oplus s_4 \oplus s_2 \oplus O_2 \\
s_9 & = s_6 \oplus s_5 \oplus s_3 \oplus O_3 \\
s_{10} & = s_1 \oplus s_3 \oplus s_6 \oplus O_1 \oplus O_4 \\
s_{11} & = s_2 \oplus s_1 \oplus s_3 \oplus O_1 \oplus O_2 \oplus O_5 \\
s_{12} & = s_3 \oplus O_3 \oplus s_4 \oplus s_2 \oplus O_2 \oplus O_6 \\
s_{13} & = s_3 \oplus O_4 \oplus s_5 \oplus O_3 \oplus s_4 \oplus O_7 \\
 & \ldots \\
s_{38} & = s_1 \oplus s_3 \oplus s_4 \oplus O_1 \oplus O_7 \oplus O_9 \\
 & \oplus O_{10} \oplus O_{11} \oplus O_{13} \oplus O_{14} \oplus O_{16} \oplus O_{18} \\
 & \oplus O_{21} \oplus O_{22} \oplus O_{23} \oplus O_{24} \oplus O_{28} \oplus O_{29} \oplus O_{32} \\
s_{39} & = s_5 \oplus s_4 \oplus s_2 \oplus O_2 \oplus O_8 \oplus O_{10} \\
 & \oplus O_{11} \oplus O_{12} \oplus O_{14} \oplus O_{15} \oplus O_{17} \oplus O_{19} \\
 & \oplus O_{22} \oplus O_{23} \oplus O_{24} \oplus O_{25} \oplus O_{29} \oplus O_{30} \oplus O_{33} \\
s_{40} & = s_6 \oplus s_5 \oplus s_3 \oplus O_3 \oplus O_9 \oplus O_{11} \\
 & \oplus O_{12} \oplus O_{13} \oplus O_{15} \oplus O_{16} \oplus O_{18} \oplus O_{20} \\
 & \oplus O_{23} \oplus O_{24} \oplus O_{25} \oplus O_{26} \oplus O_{30} \oplus O_{31} \oplus O_{34} \\
s_{41} & = s_1 \oplus s_3 \oplus s_6 \oplus O_1 \oplus O_4 \oplus O_{10} \\
 & \oplus O_{12} \oplus O_{13} \oplus O_{14} \oplus O_{16} \oplus O_{17} \oplus O_{19} \\
 & \oplus O_{21} \oplus O_{24} \oplus O_{25} \oplus O_{26} \oplus O_{27} \oplus O_{31} \oplus O_{32} \oplus O_{35} \\
s_{42} & = s_2 \oplus s_1 \oplus s_3 \oplus O_1 \oplus O_2 \oplus O_5 \\
 & \oplus O_{11} \oplus O_{13} \oplus O_{14} \oplus O_{15} \oplus O_{17} \oplus O_{18} \oplus O_{20} \\
 & \oplus O_{22} \oplus O_{25} \oplus O_{26} \oplus O_{27} \oplus O_{28} \oplus O_{32} \oplus O_{33} \oplus O_{36} \\
s_{43} & = s_3 \oplus O_3 \oplus s_4 \oplus s_2 \oplus O_2 \oplus O_6 \\
 & \oplus O_{12} \oplus O_{14} \oplus O_{15} \oplus O_{16} \oplus O_{18} \oplus O_{19} \oplus O_{21} \\
 & \oplus O_{23} \oplus O_{26} \oplus O_{27} \oplus O_{28} \oplus O_{29} \oplus O_{33} \oplus O_{34} \oplus O_{37}
\end{cases} \tag{5.8}$$

Relation (5.8) shows that the internal state of the NLFSR after just 31 clocks can be derived from the previous states by adding a certain linear combination of the output bits. In particular, Equation (5.9) presents the relation between s_{38} and s_7.

$$
\begin{aligned}
s_{38} =& s_7 \oplus O_7 \oplus O_9 \oplus O_{10} \oplus O_{11} \oplus O_{13} \oplus O_{14} \oplus O_{16} \\
& \oplus O_{18} \oplus O_{21} \oplus O_{22} \oplus O_{23} \oplus O_{24} \oplus O_{28} \oplus O_{29} \oplus O_{32}
\end{aligned} \tag{5.9}
$$

In the case of the Grain LF-NLFSR cipher, the polynomial describing the linear filter function is not irreducible and it can be factored as follows:

$$
\begin{aligned}
x^{80} + x^{76} + x^{60} + x^{48} + x^{27} + x^{21} + x^{19} + x^{18} =& (x+1)(x^3+x+1)(x^{18}) \\
& (x^7 + x^5 + x^4 + x^3 + 1) + \\
& (x^{14} + x^{13} + x^{11} + x^{10} + \\
& x^8 + x^6 + x^5 + x + 1) + \\
& (x^{37} + x^{35} + x^{34} + x^{32} + \\
& x^{30} + x^{25} + x^{24} + x^{23} + \\
& x^{21} + x^{17} + x^{16} + x^{10} + \\
& x^6 + x^5 + x^3 + x^2 + 1)
\end{aligned}
$$

Table 5.1 compares the results by Berbain et al. [10] with our new results for the Grain LF-NLFSR cipher.

Table 5.1: Comparison of results

	Data Complexity	Time Complexity	The number of independent variables
[10]	2^{21}	2^{49}	80
Our Results	$2^{19.28}$	$2^{44.98}$	80-17=63

5.6 Summary

This chapter investigated the security of stream ciphers based on LF-NLFSRs. First, we categorised key generations based on LF-NLFSRs. Then, we examined the security of LF-NLFSRs, random LF-NLFSRs, and a combination of LF-NLFSRs and filter generators against distinguishing attacks. We investigated a linear combination of LF-NLFSRs and observed how their structural properties impact on its security. We finally highlighted the criteria for the design of stream ciphers that employ linearly filtered nonlinear sequences. Based on the proposed criteria, we presented an improved algebraic attack on the Grain LF-NLFSR cipher. The attack has time complexity $2^{44.98}$ and data complexity $2^{19.28}$.

6

Practical attack on NLM generators

In 2000, Hoon Jae Leea and Sang Jae Moon proposed an improved summation generator with 2-bit memory (LM-type Generator) [89]. The design was intended to enhance security properties by adding an extra bit of memory to the combining function. However, there were still some cryptographic weaknesses in the cipher. Due to a high correlation between the input variables and output sequences of the combining function, the authors of [32, 112] showed that the cipher is vulnerable to correlation attacks. Also, an efficient attack recovering the internal state of the cipher in real time was published in [67].

The NLM stream cipher [90] is actually a modification of the LM-type generator, proposed by Hoon Jae Lee, Sang Min Sung, and Hyeong Rag Kim in 2009. The main idea of the NLM generator is to add a nonlinear feedback shift register to the summation generator that strengthens the cipher. In addition, the authors of [91] have checked the performance of the NLM stream cipher for low power consumption applications and have confirmed that the cipher is suitable for implementations requiring a small number of gates.

Related Work

Message authentication codes (MACs) are cryptographic tools that provide integrity and authentication of messages. A typical MAC can be designed using a symmetric cipher. There are many constructions of stream ciphers with a built-in MAC functionality (see [25, 52, 88, 137, 145] as examples). Recently, Lee et al. in [88] have proposed a lightweight secure data communication framework, based on the NLM stream cipher and a new MAC function combined with the cipher, to enhance security in wireless sensor networks.

The NLM stream cipher has also been employed in two RFID mutual authentication protocols [92, 93] to encrypt sensitive data. In addition, Lee, Kim and Lee have proposed an internet protocol to establish secure access for mobile users based on the functionality of the NLM generator [93]. The cryptographic analysis presented in this chapter shows weaknesses of the NLM generator and discusses their impact on the security of the protocols it supports.

This chapter is structured as follows. Section 6.1 describes briefly the NLM stream cipher and NLM-MAC function. Section 6.2 investigates the weaknesses of the cipher and proposes a state recovery attack on the NLM generator and a forgery attack on the MAC function. Also, the section discusses the weaknesses of the whole scheme. Section 6.3 summarises the results.

6.1 Description of NLM-MAC Scheme

In this section, we first describe the NLM-128 generator. Then we explain how the NLM-MAC algorithm works.

6.1.1 NLM-128 Stream Cipher

The NLM-128 stream cipher is based on summation generation, which uses LFSR and NLFSR sequences and two memory bits: a carry bit (c_i), and a memory bit (d_i). Figure 6.1 depicts the cipher. The LFSR has primitive polynomial $P(x)$ as follows:

$$P(x) = x_{127} \oplus x_{109} \oplus x_{91} \oplus x_{84} \oplus x_{73} \oplus x_{67} \oplus x_{66} \oplus x_{63} \oplus x_{56} \oplus x_{55} \oplus x_{48} \oplus x_{45} \oplus x_{42} \oplus x_{41} \oplus x_{37} \oplus x_{34} \oplus x_{30} \oplus x_{27} \oplus x_{23} \oplus x_{21} \oplus x_{20} \oplus x_{19} \oplus x_{16} \oplus x_{13} \oplus x_{12} \oplus x_{7} \oplus x_{6} \oplus x_{2} \oplus 1$$

NLFSR uses a nonlinear feedback function $f(x)$ of degree 129 defined as

$$
\begin{aligned}
f(x) \quad = \quad & x_5 \oplus x_9 \oplus x_{13} \oplus x_{17} \oplus x_{21} \oplus \\
& x_{25} \oplus x_{29} \oplus x_{33} \oplus x_{37} \oplus x_{41} \oplus \\
& x_{45} \oplus x_{49} \oplus x_{53} \oplus x_{57} \oplus x_{61} \oplus \\
& x_{65} \oplus x_{69} \oplus x_{73} \oplus x_{77} \oplus x_{81} \oplus \\
& x_{85} \oplus x_{89} \oplus x_{93} \oplus x_{97} \oplus x_{101} \oplus \\
& x_{105} \oplus x_{109} \oplus x_{113} \oplus x_{117} \oplus x_{121} \\
& \oplus x_{125} \oplus x_{129} \oplus (x_1 \cdot x_2 \cdots x_{128} \cdot x_{129}).
\end{aligned} \tag{6.1}
$$

The carry bit c_j and the additional memory bit d_j are updated according to the following relations:

$$c_j \quad = \quad a_j \cdot b_j \oplus (a_j \oplus b_j) \cdot c_{j-1} \tag{6.2}$$

$$d_j \quad = \quad b_j \oplus (a_j \oplus b_j) \cdot d_{j-1} \tag{6.3}$$

Finally, the keystream bit, z_j, is generated as shown below:

$$z_j = a_j \oplus b_j \oplus c_{j-1} \oplus d_{j-1} \tag{6.4}$$

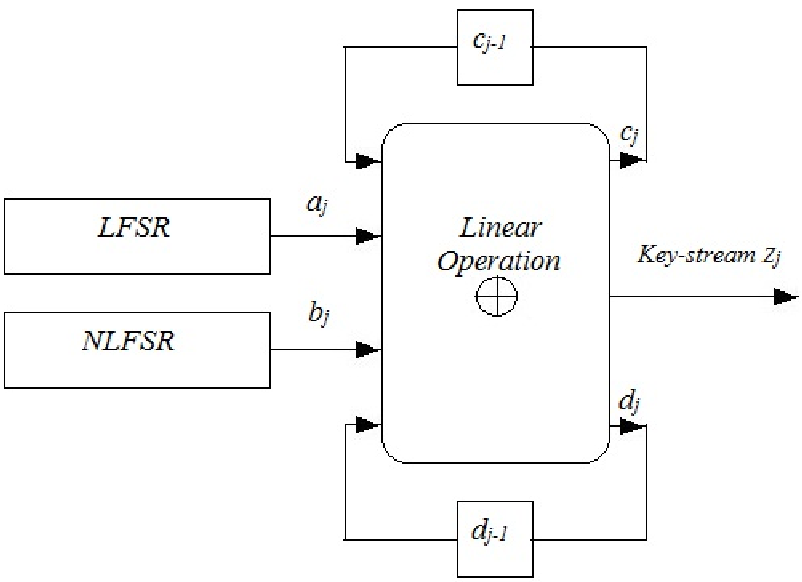

Figure 6.1: NLM family stream cipher

6.1.2 The NLM-MAC Function

The NLM message authentication code authenticates the two parties, the sender and receiver, and verifies the integrity of transmitted messages as follows:

1. The sender encrypts a plaintext with an encryption key and an initialisation vector and generates the corresponding ciphertext (C) using the NLM-128 stream cipher.

2. The sender computes a MAC value for the ciphertext with a MAC-Key (i.e., K_{mac}) according to the following steps:

   2.1 C is split into 32-bit words and then the last word is padded with zeros if required.

   2.2 K_{mac} is fed through 32-bit variables l, m, n, p and then K_{mac} is linearly combined with 32-bit C words and 32 bits of l.

   2.3 After linearly combining all 32-bit C words with l, the NLM-MAC will be generated as follows:

$$\text{NLM-MAC} = l \oplus m \oplus n \oplus p$$

3. The receiver checks the validity of the MAC tag and then decrypts the authenticated ciphertext to obtain the plaintext.

The protocol uses a time stamp to check the freshness of the messages. The time stamp has no impact on our proposed attack.

6.2 Cryptanalysis of NLM-MAC Scheme

In this section, we reveal several weaknesses of the NLM algorithm and demonstrate the attack's details. We also prove that the attacker not only can break the NLM stream cipher but can also generate valid MAC tags for fake messages in real time.

6.2.1 Cryptanalysis of NLM generator

The NLM generator is the strengthened version of the LM generator family. The generator aims to prevent the attacks published in [32, 67, 112] by using an NLFSR to make the design resistant against correlation and algebraic attacks. First, we identify the weaknesses of the cipher.

1. The algebraic degree of the keystream output bits when the cipher uses two LFSRs is only 2. In [67], Han and Lee show that the algebraic degree of LM-type generators can be kept constant (with degree 2). To show this, we take Equations (6.2) and (6.3) and get

   $$c_j \oplus d_j = a_j b_j \oplus b_j \oplus (a_j \oplus b_j)(c_{j-1} \oplus d_{j-1}).$$

   If we put $c_{j-1} \oplus d_{j-1} = z_j \oplus (a_j \oplus b_j)$ to Equation (6.4), we obtain the following equation

   $$c_j \oplus d_j = a_j b_j \oplus b_j \oplus (a_j \oplus b_j)(z_j \oplus (a_j \oplus b_j)) \tag{6.5}$$

   Substituting $(j+1)$ for j in Equation (6.4) and using Equation (6.5), we finally have

   $$z_{j+1} = a_j + 1 \oplus b_j + 1 \oplus a_j \oplus a_j b_j \oplus (a_j \oplus b_j) z_j. \tag{6.6}$$

   Equation (6.6) creates equations of degree 2 connecting two output bits and the register outputs.

2. The NLM designers believed that replacing an LFSR by an NLFSR strengthens the design and makes it resistant against algebraic analysis. To keep the desirable properties of the LM cipher, they have used an NLFSR that has the full period with feedback function (relation 6.1). Although the algebraic degree of the feedback function (6.1) is high and equal to 129, the nonlinearity is surprisingly low. The attacker can approximate the nonlinear feedback function with a linear function with the following probability:

   $$Pr(f(x) = L(x)) = 1 - 2^{-129}$$

   where $L(x) = (x_5 \oplus x_9 \oplus x_{13} \oplus x_{17} \oplus x_{21} \oplus x_{25} \oplus x_{29} \oplus x_{33} \oplus x_{37} \oplus x_{41} \oplus x_{45} \oplus x_{49} \oplus x_{53} \oplus x_{57} \oplus x_{61} \oplus x_{65} \oplus x_{69} \oplus x_{73} \oplus x_{77} \oplus x_{81} \oplus x_{85} \oplus x_{89} \oplus x_{93} \oplus x_{97} \oplus x_{101} \oplus x_{105} \oplus x_{109} \oplus x_{113} \oplus x_{117} \oplus x_{121} \oplus x_{125} \oplus x_{129}$.

The second weakness lets the attacker replace the NLFSR with the LFSR defined by the feedback function $L(x)$. The cipher can be broken in the two following steps.

1. The attacker constructs the nonlinear algebraic system based on equations of the form (6.6). The number of variables equals the total length of the shift registers and two memory bits (*e.g.* $n = 258$)). Han and Lee [67] prove that the time complexity of solving the system is $O(n^{5.6})$ and the attack needs about n^2 bits.

2. Then, the attacker checks the validity of the recovered internal state. To this end, the attacker needs to generate additional output bits by using the recovered internal state. The probability of recovering incorrect internal state equals to $2^{-129} \times n^2 = 2^{-129} \times (258)^2 = 2^{-111}$, which is still a negligible probability. In addition, one can repeat the attack on the next n^2 bits of keystream and find the internal state to verify the previous result.

6.2.2 Analysis of NLM-MAC Function

The most critical point is that the NLM-MAC function is totally linear. This means that all relations between the MAC secret key K_{mac} and the ciphertext are linearly constructed. So, one can compute the linear relation of K_{mac} words by having only one MAC tag and its corresponding ciphertext. This leakage reveals the linear relation of l, m, n, p which are enough to compute a valid MAC value for every arbitrary ciphertext.

6.2.3 Attack on NLM Scheme

Now, we show how we can launch a key-recovery attack on the NLM cipher and forge the MAC value. What the attacker needs is about 2^{16} bits of keystream, a MAC tag and its corresponding ciphertext. The attack works as follows:

1. For a ciphertext of length n^2 bits, where n is the number of internal bits, the attacker finds the internal states of the cipher with negligible error probability.

2. For the pair (ciphertext, MAC tag), the attacker applies the attack explained in Section 6.2.2.

3. The attacker sends an arbitrary ciphertext along with a valid MAC tag, or by adding new plaintext bits following the original plaintext, he computes ciphertext and updates a new MAC value. Another approach is to replace the original plaintext with an arbitrary text and compute the corresponding ciphertext and MAC tag.

6.3 Summary

In this chapter, we analysed the NLM-MAC scheme proposed for lightweight applications, such as wireless sensor networks. We discovered some weaknesses leading to two successful cryptographic attacks. The first attack allows the adversary to recover the internal state with time complexity of about $2^{44.86}$. The proposed attack requires about

2^{16} output bits. The second attack permits the adversary to forge an MAC tag for every ciphertext in real time. Finally, we proposed an attack on the protocol, which lets the attacker generate arbitrary ciphertexts along with a valid MAC tag. In conclusion, it is shown that the proposed scheme is totally insecure and is not recommended for use.

7

Cryptanalysis of RC4(n,m) stream cipher

The RC4 stream cipher was designed in 1987 by Ron Rivest. The cipher uses a large internal state that is stored in an array of words. Because of its simplicity of design and the high speed offered by software implementation, the cipher has gained popularity in many internet applications, such as TLS/SSL and WEP.

In fact, RC4 is a family of stream ciphers indexed by an integer n that indicates the size of the word in bits. The internal state is an array S of 2^n words. RC4 consists of two algorithms. The first is a key scheduling algorithm (KSA) and it initialises the internal state. The second is a pseudorandom generation algorithm (PRGA). It generates the output keystream. The KSA algorithm takes an array S and a secret key K, and produces the initial state or a secret permutation of $\{0, 1, 2, \ldots, 2^n - 1\}$. The PRGA algorithm accepts the initial state S and produces a sequence of words. A popular instantiation of RC4 is for $n = 8$. In this instantiation, words are 8 bits long and the array S contains $2^8 = 256$ entries. The security of RC4 has been extensively studied. The key schedule of RC4 is examined in [55, 97, 98, 113, 119, 133]. Distinguishing attacks are presented in [54, 61, 97, 103, 120]. The internal state recovery attacks are investigated in [86, 105].

When the parameter n is bigger, for instance $n = 32$, the implementation requires more memory and, in general, the cipher becomes slower. On the positive side, one would expect that the cipher will be stronger. This line of investigation resulted in several generalisations of RC4-like stream ciphers, see for example RC4A [119], VMPC

[144], NGG [115] and $RC4(n,m)$ [65]. We focus on the Gong et al. design, $RC4(n,m)$, given in [65]. In this cipher the state array S no longer consists of 2^n entries. Consequently the state is no longer a permutation. This cipher is called $RC4(n,m)$, where the state array consists of 2^n entries (words) and each word is m bits long ($n < m$). For a 32−bit architecture, the recommended parameter values are $n = 8$ and $m = 32$.

$RC4(n,m)$ is a fast synchronous stream cipher proposed by Guang Gong, Kishan Chand Gupta, Martin Hell and Yassir Nawaz in [65]. $RC4(n,m)$ produces m bits per clock. The main idea of design $RC4(n,m)$ is to exploit the 32-bit and 64-bit processor architectures without increasing the size of the table significantly. The internal state size of $RC4(n,m)$ is $(2^n m) + 2n + m$ bits long, since it consists of an array of 2^n entries and each entry takes m bits , one m-bit variable k and two n-bit indexes i and j. Note that the key length is proposed to be up to 8192 bits but security is provided for keys up to 256 bits in size.

Previous Work

$RC4(n,m)$ has been proposed based on a 32-bit RC4-like stream cipher [115] by Y. Nawaz, K.C. Gupta, and G. Gong called the NGG stream cipher, to improve the security of the cipher against proposed attacks. H. Wu proposed a distinguishing attack on NGG [140] which could distinguish the keystream outputs from random sequences with about 3200 output bits. Also, in [84], a new distinguisher and key-recovery attack on the cipher has been proposed. The attack can distinguish the cipher from a random stream using only the first keystream word. The attacker also can recover the secret key by exploiting leaked information from the first few kilobytes of the keystream output. But the story about the new version called $RC4(n,m)$ or GGHN is different. [1] Note that none of the above proposed attacks on NGG are applicable to $RC4(n,m)$.

To our best knowledge, there are few useful attacks on $RC4(n,m)$. Paul and Preneel have proposed a distinguishing attack that needs $2^{32.89}$ output keystream words to distinguish the cipher from a random source [120]. The second attack [132], proposed by Tsunoo, Saito, Kubo, and Suzaki, is a distinguishing attack which uses the bias along with the first two words of a keystream associated with approximately 2^{30} secret keys. In the attack, the authors explore the correlation between indices and entries of the array. The third attack, proposed by Kircanski and Youssef [83], is a fault attack which extracts the internal state of the cipher by applying induced faults. The attack also needs 2 keystream words for each of 257×255 induced faults and approximately 257 non-faulted keystream words.

[1] NGG (or NGG(n,m)) is a previous version of $RC4(n,m)$ (or GGHN, GGHN(n,m)). In this chapter, we analyse the security of $RC4(n,m)$ (GGHN).

Our Contributions

We study the security of $RC4(n,m)$. We will show several weaknesses in the initialisation and update function of the algorithm. Two distinguishing attacks are described. The first attack takes an advantage of the bias of the least significant bits of the internal state. The idea of this attack is similar to [120, 132], but we apply it to the key scheduling algorithm. The second attack is based on truncated differentials and requires 256 output words only. Finally, we will present a key-recovery attack, which is able to find 256-bit secret keys with time complexity of about 2^{13} algorithm operations for $RC4(8,32)$. The current state of analysis is summarized in Section 7.3

In application protocols like Wired Equivalent Privacy (WEP), there is a session-dependent initial value that needs to be introduced as input to the stream cipher to produce different pseudorandom streams for different sessions. In these protocols, the attacker can often manipulate the initial value, as for example in the attack [55] on RC4 used in the WEP protocol, and exploit a chosen IV attack model to investigate the security of the scheme. Consequently, for such applications, the stream cipher needs to be designed to accept initial value inputs, and its security needs to be assessed with respect to initial value inputs chosen by the attacker. $RC4(n,m)$ also uses three input parameters (Figure 7.1): secret key, initial vector, and initial value (to initialise the internal state before applying the key scheduling algorithm). From a practical point of view, the system designer may exploit all the features of the crypto-algorithm to improve efficiency. In this case, using the initial value and initial vector as variable input parameters to differentiate applications, and also to increase the security level, may seem to be reasonable. We study the extreme misuse of $RC4(n,m)$ when the initial value is assumed to be under the attacker's control. The protocol initial value could be incorporated in two ways: either as the "initial value" input to the KSA* module, or as part of the secret key input (using a hash function) as the authors of RC4(n,m) proposed. From the implementation point of view, the first option might be tempting since it may be simpler to implement. For the sake of clarity, we assume that the attacker is able to change the initial value. In this case, we will prove that the cipher is surprisingly insecure against the distinguishing and key-recovery attacks. We also note that the attacks (5) and (6) in Table 7.2 are not applicable when the attacker is not allowed to manipulate the initial value.

This chapter is organised as follows. Section 7.1 provides a description of the initialisation and key generation parts of the scheme. Section 7.2 is the main part of the study, which contains our distinguishing and key-recovery attacks. Section 7.3 summarises the results.

7.1 Description of RC4(n,m) Stream Cipher

The $RC4(n,m)$ stream cipher uses the building blocks defined for the RC4 stream cipher. These blocks, however, are modified by the authors. A general illustration of $RC4(n,m)$ is presented in Figure 7.1.

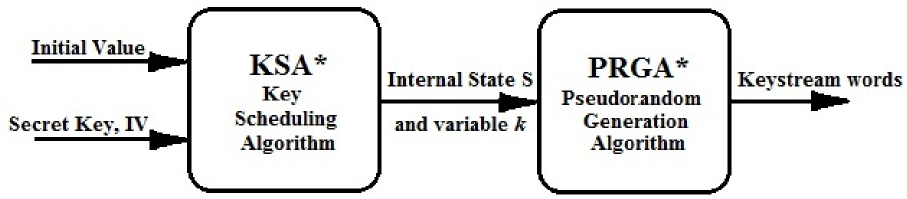

FIGURE 7.1: $RC4(n,m)$ stream cipher scheme

The algorithms of $RC4(n,m)$ are called KSA* and PRGA* to distinguish them from KSA and PRGA, respectively. To recall, the parameters of $RC4(n,m)$ are defined as follows. The size of the state array is $N = 2^n$ and each entry of the array holds m bits. Entries are going to be called words. We define a constant $M = 2^m$. For example, $RC4(8,32)$ means that the size of array S is 256 and each entry of S holds 32-bit words. The original key scheduling algorithm of RC4 is described in Figure 2.7. We give the modified algorithm of $RC4(n,m)$.

- KSA* (key scheduling algorithm of $RC4(n,m)$) – the algorithm takes a secret key[2] of a size between 40 and 256 bits and a table of initial values as the input and returns an updated internal state stored in the array S and a m-bit variable k. The original paper [65] specifies the minimum value of r for $RC4(n,m)$. The full details are given in Figure 7.2.

- PRGA* (Pseudorandom Generation Algorithm) – it takes the pair: the internal state $\langle S \rangle$ and variable k as the inputs and generates output keystream words. The pseudo-code of the algorithm is given in Figure 7.3.

[2]The designers suggest using a hash function to generate the *Key* array from the secret key and initial vector to prevent possible attacks on KSA*.

Input: initial values a_i, Secret Key $Key[j]$ $0 \le i < N,\ 0 \le j < l$

Output: Internal State $\langle S \rangle$, variable k

for $i = 0$ **to** $N - 1$

  $S[i] = a_i;$

end for

$j = 0;$

$k = 0;$

Repeat r **times**;

  for $i = 0$ **to** $N - 1$

    $j = (j + S[i] + Key[i \mod l]) \mod N;$

    $swap(S[i], S[j]);$

    $S[i] = S[i] + S[j] \mod M;$

    $k = k + S[i] \mod M;$

  end for

Figure 7.2: KSA*: The key-scheduling algorithm of $RC4(n, m)$

7.1.1 Notations

$[X]_0$ is the least significant bit of the word X.
$[X]_{i,\ldots,j}$ are $(j - i + 1)$ consecutive bits of word X starting from position i and through to j.

7.2 Cryptanalysis of RC4(n,m) Stream Cipher

In this section, we prove that $RC4(n, m)$ is not resistant against distinguishing and key-recovery attacks. First, we identify weaknesses in the KSA* algorithm. Next, we exploit these weaknesses and show how to distinguish the output stream of $RC4(n, m)$ from a random cipher. The data complexity of the attack is 256 output words. Then, we

Input: Internal State $\langle S \rangle$, variable k

Output: Output (Keystream words)

$i = 0;$

$j = 0;$

Repeat the loop

  $i = i + 1 \mod N;$

  $j = (j + S[i]) \mod N;$

  $k = (k + S[j]) \mod M;$

  $output = (S[S[i] + S[j] \mod N] + k) \mod M;$

  $S[S[i] + S[j] \mod N] = S[i] + k \mod M;$

FIGURE 7.3: PRGA*: Pseudorandom generation algorithm of $RC4(n, m)$

propose a key-recovery attack based on truncated differentials. The time complexity of the attack to recover a 256-bit secret key of $RC4(8, 32)$ is about 2^{13} algorithm operations.

In some applications, one may design a cryptosystem in which the initial values are varied in different sessions. It looks like this may increase the security level of the cipher, as the attacker cannot use the results from the analysis of previous sessions generated for different initial values. In the second distinguishing attack and key-recovery attack, we assume that the initial value can be modified and selected as in the chosen IV attack. We show that the cipher is susceptible to this kind of attack.

7.2.1 Weaknesses of RC4(n,m)

Before describing our attacks, we discuss properties of the $RC4(n, m)$ cipher that underpin our attacks.

Non-Randomness Property of Internal States:

The array S has 256 elements whose lengths are 32 bits and the pointer j takes one byte. This means that, if we choose two indices $i, j \in \{0, 1\}^8$ at random, then the probability $Pr(S[i] = S[j]) = Pr(i = j) = 2^{-8}$ assuming that $S[i] \neq S[j]$ for $i \neq j$, while for two random 32-bit words, this probability is 2^{-32}. Now, we can prove that, for every element in the array S after applying the initialisation algorithm, $Pr([S[i]]_0 = 0) = 0.5 + 2^{-8}$, where $0 < i < 256$.

Weak Keys:

There are several classes for secret keys that generate internal states with short cycles. The final internal states (after a run of KSA* but before an execution of PRGA*) can be computed using a certain relationship among the states. For example, in the following, we show that the state $S[0]$ in the array moves and all other states swap with this state only.

Example 7.2.1 *Let the internal states of the algorithm be equal to the values suggested in appendix A in [65], and the secret key be 0X0101 ... 01. According to the KSA algorithm, we can write the following relations:*

$i = 0, j = 0, k = 0$	$i = 1, j = 1, k = S[0]$	$i = 2, j = 2, k = S[0] + S[1]$
$j = 0 + S[0] + K[0] = 1$	$j = 1 + S[1] + K[1] = 2$	$j = 2 + S[2] + K[2] = 3$
$Swap(S[0], S[1])$	$Swap(S[1], S[2])$	$Swap(S[2], S[3])$
$S[0] = S[0] + S[1]$	$S[1] = S[1] + S[2]$	$S[2] = S[2] + S[3]$
$k = 0 + S[0]$	$k = S[0] + S[1]$	$k = S[0] + S[1] + S[2]$

The above relations show that, in $RC4(n, m)$, there are Finney states [66] that swap $S[i]$ with $S[i + 1]$ and both indices i and j are incremented by 1. Other weak keys can be found using probabilistic relations. Note that other weaknesses have been deeply investigated in [9] recently.

Weak States

We are going to find several initial states for which the outputs will be distinguishable from a truly random source. The main weakness, which we exploit here, is a low diffusion of bits in KSA*.

1. For an arbitrary secret key, if the least significant bits of the words stored in the initial states are equal to zero, then the least significant bits of keystreams will be zero with the probability one. We can extend this observation for the least significant 2, 3, $\cdots$, 32-bit of the initial states.

2. Assume that
$$S[i] \pmod{2^n} = 1 - K[i \pmod{l}]$$
and
$$S[0] \pmod{2^n} = -K[0]$$

then j will be equal to i in the first round. This means that, after one round, all the internal states will be even (the least significant bits of internal states will always be zero).

3. Suppose that $K[0]$ is odd and $K[1] = K[0]-2$. Also assume that $(S[0])\ mod\ 2^n = (1 - K[0])\ mod\ 2^n$, $S[1]$ is even, $(S[2])\ mod\ 2^n = (2 - K[2])\ mod\ 2^n$ and $(S[i])\ mod\ 2^n = 1 - K[i\ mod\ l]\quad 3 < i < 255$, then the internal states after one round will be even. In other words, the least significant bit of the keystreams will be always zero.

Low Diffusion Property

Clearly, the update function of $RC4(n, m)$ is like a T-function [85]. This means that the i-th bit of output depends on the i-th bit of input and all less significant bits (i.e. bits $i - 1, \ldots, 0$) of the input. This is a serious weakness for $RC4(n, m)$ because, if the cryptanalyst changes the most significant bits of initial values (a_i) in KSA*, then only the most significant bits of the keystreams will be changed (other bits will be unchanged). In the RC4 initialisation algorithm, all bytes of the initial state and secret key are involved in providing the internal state as an input for the PRGA to generate the output keystream. This means that, by complementing one bit of the initial state, all bits of the keystream will be changed with a probability close to $\frac{1}{2}$. In fact, this property, called *the avalanche criterion*, is one of the most essential properties of a secure cipher. However, this property has been confirmed only for the least significant bytes of the initial value array. In other words, if we change the $I-$th bit $(32 > I \geq 8)$, then more significant bits i may change $(i > I)$, while less significant bits (with index $i < I$) are not going to change. This property of the KSA* algorithm of $RC4(n, m)$ is illustrated in Figure 7.4. Now, we are ready to describe the proposed attacks on $RC4(n, m)$.

7.2.2 Distinguishing Attack on RC4(n,m)

The first attack

This attack is based on the non-randomness property of the internal states, which is described in the previous section. Consider line 12 of Figure 7.2 in the $r - th$ round of the algorithm.

Proposition 7.2.1 *Assume that (1) The index j at line 10 of KSA^* is uniformly distributed in* $\{0, ..., N - 1\}$ *and independent of i, and (2) if* $i \neq j$ *then S[i]+S[j] mod*

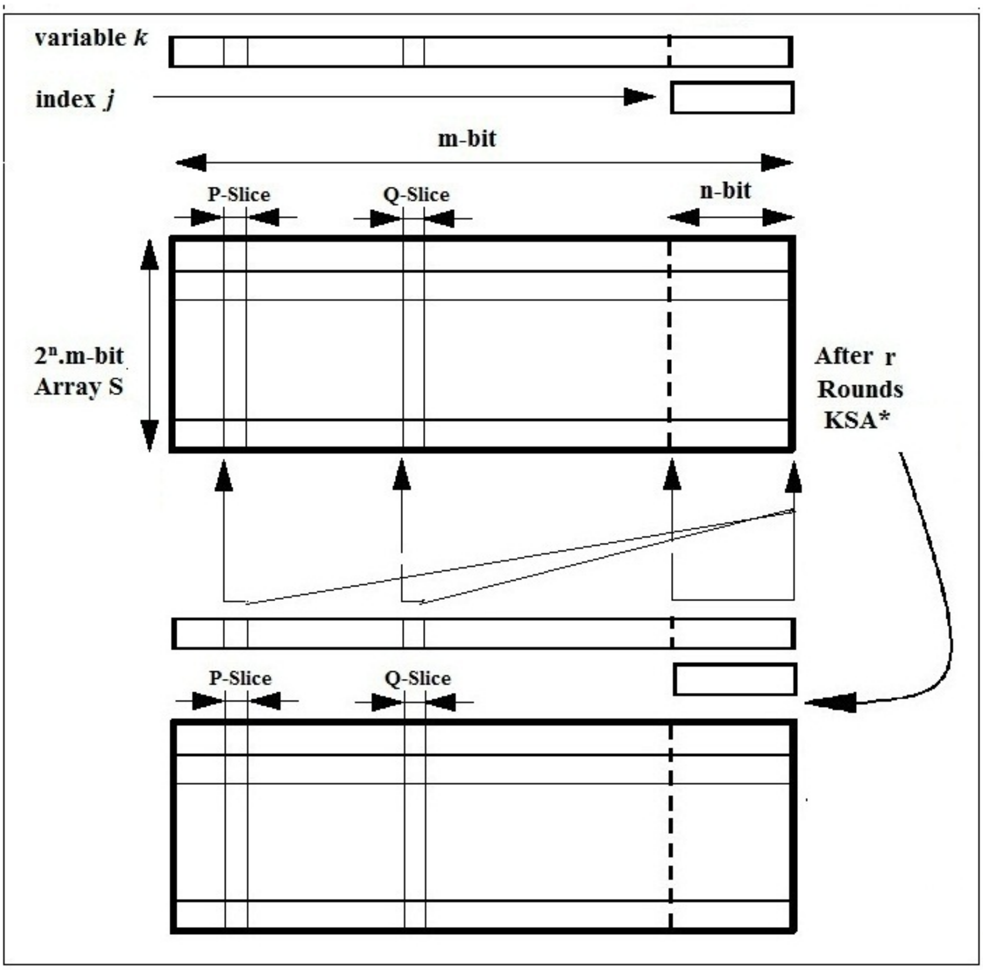

Figure 7.4: RC4(n,m): Another perspective of KSA*. According to the Low Diffusion Property, the P-th slice of the internal state in the r-th round of KSA* (r is an arbitrary round) depends on the P-th slice of the internal state and previous slices in the initial state. Also, any random difference in the P-th slice in the initial state does not change the Q-th slice in the r-th round of KSA*.

M is independent and uniform in $\{0, ..., N-1\}$. Then, for all elements of Array S, after performing KSA^* we have:

$$Pr([S[i]]_0 = 0) = \frac{1}{2}(1 + \frac{1}{2^n}) \qquad 0 \leq i < 2^n$$

Proof 9 *First, we know that, if $i = j$, then in line 12 we have that $S[i] = 2 \cdot S[i] \bmod M$ is even since M is even. And then, $Pr([S[i]]_0 = 0) = Pr([S[i]]_0 = 0 | i \neq j) \cdot Pr(i \neq j) + Pr([S[i]]_0 = 0 | i = j) \cdot Pr(i = j) = \frac{1}{2} \cdot \frac{2^n - 1}{2^n} + 1 \cdot \frac{1}{2^n} = \frac{1}{2} \cdot (1 + \frac{1}{2^n})$.* $\square$

Thus we find that all the least significant bits of the array S contents are biased. If the keystream just depended on the array S, then we could exploit the bias to mount a distinguishing attack. But the keystream output is the summation of a word from the array S and the variable k. In addition, the variable k is the sum of randomly chosen elements of the array S. It can be shown that the least significant bit of the variable k is also biased but the bias is very close to zero. So, we need to use a combination of outputs to eliminate the effect of variable k and find a biased linear relationship. For instance, the linear combination of two consecutive outputs can reveal the expected bias. To do this, let the event E denote the condition in which the relation $k_{t+1} = k_t + S[y]$ is satisfied as follows:

$$\begin{aligned} Output[t] &= S[x] + k_t \mod M, \\ Output[t+1] &= S[y] + k_{t+1} \mod M, \end{aligned}$$

where x and y are randomly chosen indices and $t = 0$. Now, by adding $Output[0]$ and $Output[1]$, we get

$$[Output[1] \oplus Output[0]]_0 = [S[x]]_0.$$

We can formulate the following proposition.

Proposition 7.2.2 *In RC(n,m), the probability of* $([Output[1] \oplus Output[0]]_0 = 0)$ *is* $\frac{1}{2} \cdot (1 + \frac{1}{2^{(2\cdot n)}})$.

Proof 10 $Pr([Output[1] \oplus Output[0]]_0 = 0) = Pr([Output[1] \oplus Output[0]]_0 = 0|E).Pr(E) + Pr([Output[1] \oplus Output[0]]_0 = 0|E^c) \cdot Pr(E^c) = (\frac{1}{2} \cdot (1 + \frac{1}{2^n})) \cdot \frac{1}{2^n} + \frac{1}{2} \cdot \frac{2^n - 1}{2^n} = \frac{1}{2} \cdot (1 + \frac{1}{2^{(2\cdot n)}})$.
□

For an ideal PRBG, the above probability would have been exactly $\frac{1}{2}$. We can extend our assumption for more than two consecutive output words in which $S[x]$ is not updated in time $t = 0$ (or $t = 0$ and $t = 1$). In other words,

$$\begin{aligned} Output[0] &= S[z] + k_t \ \ mod\ M; \\ Output[1] &= S[x] + k_{t+1} \ \ mod\ M; \\ Output[2] &= S[y] + k_{t+2} \ \ mod\ M. \end{aligned}$$

The probability of $[Output[2] \oplus Output[1]]_0 = 0$ can be simply computed by applying Bayes' theorem as follows:

$$\begin{gathered} Pr([Output[2] \oplus Output[1]]_0 = 0) = \\ Pr([Output[2] \oplus Output[1]]_0 = 0 | x \neq z) \cdot Pr(x \neq z) + \\ Pr([Output[2] \oplus Output[1]]_0 = 0 | x = z) \cdot Pr(x = z) = \frac{1}{2} \cdot (1 + \frac{1}{2^{(2\cdot n)}} - \frac{1}{2^{(3\cdot n)}}). \end{gathered}$$

The above attack is a generalisation of the attack proposed in [120, 132] with emphasis on the initialisation part of the algorithm. In the next section, we will present distinguishing and key-recovery attacks, which exploit a low diffusion property of KSA*.

Algorithm 1 Distinguishing Attack Scenario on $RC4(n,m)$

Input: The first two (four) output words corresponding to $2^{4.n}$ randomly chosen secret keys.

Output: To distinguish between $RC4(n,m)$ outputs and a truly random source.

1. Generate $Output^k[0]$ and $Output^k[1]$, $0 \leq k < 2^{4.n}$, $Output^k[i]$ is i-th output associated with k-th secret key.
2. $S = \dfrac{\sum_k (Output^k[0] \oplus Output^k[1])}{2^{4.n}}$;
3. If $S \geq \dfrac{1}{2}$ then the algorithm which is analysed in this test is $RC4(n,m)$.

Note that the required amount of data to distinguish a biased sequence Z in which $Pr(Z_i = 0) = 1/2 + 1/2^n$ from a truly random sequence is determined as the Chernoff bound by an exponential function in n is greater than $2^{2n} \cdot \ln \frac{1}{\sqrt{1-P_S}}$ where P_S is the expected success probability. In Table 7.1, the success probabilities in theory and practice are shown.

Table 7.1: Experimental Results and Comparison between theory and simulation

n	The required amount of data	Success Probability P_e in Theory	The founded Success Probability P_e in simulation
4	$2^{16.58}$	0.95	0.97
4	$2^{14.87}$	0.60	0.59
5	$2^{20.58}$	0.95	0.94
5	$2^{18.87}$	0.60	0.58

The second attack

The second distinguishing attack is based on differential cryptanalysis [21]. Differential attacks on stream ciphers use a chosen initial value or other public variables. This kind of attack can be launched if the adversary has access to the cipher and can manipulate the external (public) elements. But, of course they cannot see the secret elements that are assumed to be hidden by, for example, tamper proof hardware. We are going to use a generalisation of differential cryptanalysis called truncated differential cryptanalysis. Whereas the standard differential cryptanalysis considers the full difference between two inputs, the truncated variant takes differences that are only partially determined. So, the attacker can predict only some of the bits.

As noted in Section 7.2.1, a modification of more significant bits will not change less significant bits. This property is rephrased below.

Remark 1: Let $X + Y = Z$, $X, Y, Z \in GF(2^m)$, $\Delta = R\underbrace{0\ldots 0}_{m-k}$, $R \in GF(2^k)$ and R be an arbitrary differential input then, for

$$\begin{cases} (X \oplus \Delta_1) \boxplus Y = Z_{\Delta_1} \\ (X \oplus \Delta_2) \boxplus Y = Z_{\Delta_2} \end{cases} \tag{7.1}$$

where $\Delta_1 = R_1 0 \cdots 0$, $\Delta_2 = R_2 0 \cdots 0$, and $R_1 \neq R_2$, we have:

$$[Z_{\Delta_1}]_{0\cdots(m-k)} = [Z_{\Delta_2}]_{0\cdots(m-k)}.$$

In modular addition, the most significant bits do not affect the least significant bits. So, applying difference vectors to more significant bits changes just the corresponding output bits and the difference for less significant bits will be zero.

Remark 2: If $Y = (X \oplus \Delta)$ and $\Delta = 1\underbrace{0\cdots 000}_{m-1}$ then $X \boxplus X = Y \boxplus Y = (2 \cdot X)\ mod\ 2^m$. (*i.e.* the differential value in the output will have disappeared.)

Theorem 11 *Given the RC4(n,m) cipher. Let there be two initial values IV_1 and IV_2, where $IV_2[i] = IV_1[i] \oplus \Delta_{IV}[i]$ and $\Delta_{IV}[i] = 0XRR\ 00 \cdots\ 00$ is a truncated differential vector. The length of $\Delta_{IV}[i]$ is m bits and $0 \leq i < 2^n$. For $\Delta_{IV}[i]$, the byte RR is different from zero. Then, for all output keystream words $Output_1$ and $Output_2$ related to IV_1 and IV_2, we have:*

$$[Output_1[j]]_{0\cdots(m-8)} = [Output_2[j]]_{0\cdots(m-8)}$$

with probability one, where $[Output_k[j]]_0$ is the least significant bit of the j-th output keystream word, $j \geq 0$, and k=1,2.

Proof 11 *For two initial vectors IV_1 and IV_2, the least significant bytes are the same. According to lines 9 and 10 in Figure 7.2 and lines 6 and 7 in Figure 7.3, the indices i and j are all updated modulo 2^8. So, the index j, which depends on the secret key bytes and the least significant bytes of the internal state, will be the same. Consequently,* *updating the internal state and variable k is similar. However, updating array S and variable k is based on modular addition, then changing MSB does not change the least significant bits (see Remark 1), and the bits with less significant bits will remain the same.* □

A distinguishing attack on $RC4(8, 32)$ based on Theorem 11 is shown as Algorithm 2.

Algorithm 2 Distinguishing Attack Scenario on $RC4(8, 32)$

Input: Two initial vectors IV_1 and IV_2 which satisfy Equation 7.2.

Output: To distinguish between RC4(8,32) outputs and a truly random source.

1. Select k elements to apply differential input vectors from set $\mathcal{K}$ ($|\mathcal{K}| = k$) where $1 \leq k < 2^8$.
2. Select differential vectors $\Delta_{IV}[i] = 0xRR000000$ where $i \in \mathcal{K}$ and RR are non-zero and arbitrary bytes.
3. Generate 2^n output keystream words $Output_1[j]$ and $Output_2[j]$ corresponding to IV_1 and IV_2 where

$$\begin{cases} IV_1[i] = IV_2[i] \oplus \Delta_{IV}[i] & i \in \mathcal{K} \\ IV_1[i] = IV_2[i] & otherwise \end{cases} \tag{7.2}$$

4. Compute output differential vector as $\Delta_{Output}[j] = Output_1[j] \oplus Output_2[j]$
5. If the general form of $\Delta_{Output}[j] = 0xSS$ 00 00 00 where SS is output truncated differential bytes, then the algorithm which is analysed in this test is RC4(8,32).

7.2.3 Key-Recovery Attack on RC4(n,m)

Now we prove that the attacker is able to recover the secret key of $RC4(n, m)$ by guessing each byte of the secret key sequentially. There are three phases of our key-recovery attack.

1. Guess each byte of the secret key,
2. Generate appropriate input differential initial values by Equation 7.2,
3. Verify the validity of the guess.

Without loss of generality, we first focus on recovering the first byte of the secret key. According to Figure 7.2, this byte first affects the $S[0]$ array. Let us define $\Delta = 0X80$ 00 00 00. We consider $\mathcal{K} = \{0\}$, and $\Delta_{IV}[0] = \Delta$, and $SK[0]$ as the least

Algorithm 3 Key-Recovery Attack Scenario on $RC4(8, 32)$ (First byte of Secret Key)

Input: Two initial vectors IV_1 and IV_2 which satisfy Equation 7.3.

Output: Key Recovery of $SK[0]$ (the least significant byte of the secret key).

1. Guess SK[0]=$\widehat{SK_0}$,
2. Compute $[IV_1[0]]_{0\cdots7} = [IV_2[0]]_{0\cdots7} = (-\widehat{SK_0})\ mod\ 2^8$,
2. Select a differential vector $\Delta_{IV}[0] = \Delta$,
3. Generate 2^8 output keystream words $Output_1[j]$ and $Output_2[j]$ corresponding to IV_1 and IV_2 where

$$\begin{cases} IV_1[0]= IV_2[0] \oplus \Delta_{IV}[0] & \\ IV_1[i]= IV_2[i] & 1 \leq i < 2^8 \end{cases} \tag{7.3}$$

4. Compute output differential vector as $\Delta_{Output}[j] = Output_1[j] \oplus Output_2[j]$
5. If $\Delta_{Output}[j] = 0X00\ 00\ 00\ 00$, then $\widehat{SK_0}$ is the least significant byte of the secret key with probability close to one. Otherwise, Go to step 1.

significant byte of the secret key. Now the key-recovery attack on $RC4(8, 32)$ can be designed based on Theorem 11 and is shown as Algorithm 3.

Remark 3: When the attacker finds a $\widehat{SK_0}$ which is confirmed in step 5, then they have to repeat the scenario with the same $\widehat{SK_0}$ and different $IV_1[i]$ and $IV_2[i]$ to be sure that the guessed $\widehat{SK_0}$ is the least significant byte of the secret key with probability one.

Remark 4: The attack efficiency does not depend on the parameter r. This means that, if the designers increase r, the attack will still be applicable.

To recover all bytes of the secret key, we simply need to perform the above sequences again. For example, to recover the $k-th$ ($0 \leq k < 32$ for 256-bit secret key) byte of the secret key, the attacker has to find all bytes of $SK[i]$ where $0 \leq i < k$ according to Algorithm 4.

To verify the theoretical results, we implemented the key-recovery attack on RC4(8,32). The attack can recover a 256-bit secret key in less than one second on a standard PC. Also, the attack complexity is not different for other members of the $RC4(n, m)$ family.

7.2.4 Discussion

Thwarting the proposed attacks: The attacks proposed in this chapter were based on two critical weaknesses. The first weakness is the non-randomness property of the internal state after applying KSA*. This weakness is actually a natural attribute of

Algorithm 4 Key-Recovery Attack Scenario on $RC4(8, 32)$

Input: Two initial vectors IV_1 and IV_2 which satisfy Equation 7.4.

Output: Key Recovery of $SK[k]$ (the k-th byte of the secret key).

1. Guess $SK[k] = \widehat{SK_k}$,
2. Compute $[IV_1[k]]_{0\cdots7} = [IV_2[k]]_{0\cdots7} = (-\widehat{SK_k})\ mod\ 2^8$,
2. Select differential vector $\Delta_{IV}[k] = \Delta$,
3. Generate 2^8 output keystream words $Output_1[j]$ and $Output_2[j]$ corresponding to IV_1 and IV_2 where

$$\begin{cases} [IV_1[i]]_{0\cdots7} = [IV_2[i]]_{0\cdots7} = SK[i] & 0 \le i < k \\ IV_1[i] = IV_2[i] \oplus \Delta_{IV}[i] & i = k \\ IV_1[i] = IV_2[i] = arbitrary\ \ 32-bit\ \ values & k < i < 2^8 \end{cases} \tag{7.4}$$

4. Compute the output differential vector as $\Delta_{Output}[j] = Output_1[j] \oplus Output_2[j]$
5. If $\Delta_{Output}[j] = 0X00\ 00\ 00\ 00$, then $\widehat{SK_k}$ is the the $k-th$ byte of the secret key with a probability close to one. Otherwise, Go to step 1.

the cipher. The main difference between RC4 and $RC4(n, m)$ is that the length of elements of internal i and j are kept fixed. The second weak point is a low diffusion property. This weakness can be removed by using some simple linear operations like bit-rotation to relocate the positions of the least and most significant bits of the internal states during the running of KSA* and PRGA*.

7.3 Summary

The security of $RC4(n, m)$ was investigated, and in particular the initialisation part of the algorithm was analysed. The proposed attacks were based on the non-randomness of the internal state, which led to a statistical distinguishing attack. In addition, based on low diffusion property in KSA* and PRGA*, the attacker can apply a truncated differential technique to recover all bytes of the *key* array. These attacks are only applicable when the protocol allows manipulation of the initial value. In this scenario, we have shown that the output keystream can be distinguished from a truly random sequence with possession of just 256 output words with success probability near one. By using this weak point, a practical key-recovery attack, which recovers a 256−bit secret key with time complexity about 2^{13} algorithm operations, has been proposed. Table 7.2 gives a comparison between the previous attacks and the proposed attacks.

TABLE 7.2: Comparison between the previous attacks and our proposals

	Attack type	The result	Data Complexity	Time Complexity	Comments
1	Correlation attack [120]	Distinguishing	$2^{32.82}$ output words of a single stream	$2^{32.82}$	Applied on RC4(8,32)
2	Correlation attack [132]	Distinguishing	2^{30} first two words of keystreams	2^{30}	Applied on RC4(8,32)
3	Fault attack [83]	Internal state Recovery	2 keystream words. For each of 257×255 induced faults and approximately 257 non-faulted keystream words	$\approx 2^{16}$ + negligible additional complexity to perform attack	Applied on RC4(8,32)
4	Our correlation attack	Distinguishing	$2^{4 \cdot n}$	$2^{4 \cdot n}$	Applied on $RC4(n,m)$
5	Our differential attack	Distinguishing	2^n output words corresponding to two initial vectors	2^{n+1}	Applied on $RC4(n,m)$
6	Our differential attack	Secret Key Recovery	$2^n \times 2^n$ output words corresponding to two initial vectors to recover each key byte	$(L/n) \cdot 2^n$ where L is secret key length in bits	Applied on $RC4(n,m)$

8

Cryptanalysis of a hash function based on RC4

Hash functions are indispensable for a variety of security applications that include message authentication, integrity verification, and digital signatures. Recent developments in the analysis of hash functions have demonstrated that most members of the MD family have many weaknesses that may compromise the security of the applications in which the hash functions are used. It turns out that for hash functions, such as MD5, SHA-0 and SHA-1 [134–136], there are attacks that allow us to find random collisions faster than expected. These advances in the cryptanalysis of hashing functions are the main motivation for the NIST call for the new SHA-3 cryptographic hash standard [1]. SHA-3 was public and generated a great deal of interest in the cryptographic community.

There has been a constant flow of new design ideas and new analysis techniques. One such idea is the use of stream ciphers to construct new hash functions. The RC4 stream cipher seems to be an attractive option to build a fast, and lightweight hash function [33, 142]. It is a very simple and elegant cipher that can be implemented using relatively modest computing resources. More importantly, RC4 has been studied for many years, and its efficiency makes it a good cryptographic tool for building hash functions that can be implemented as a lightweight algorithm. In 2006 Chang, Gupta, and Nandi [33] proposed a hash function, called RC4-Hash, that uses RC4 as the building block. The compression function in RC4-Hash applies the key scheduling algorithm (KSA) that is one of the main components of RC4. Because of a specific structure of RC4-Hash, generic attacks, that are so effective against hash functions from the MD family, fail to work. However, in 2008 Idesteege and Preneel [73] showed that RC4-Hash is not collision resistant.

Recently Yu, Zhang, and Haung [142] came up with an another hash function design that is based on RC4. The function was called the RC4-based hash function and we are going to call it RC4-BHF. In addition to the KSA function, the RC4-BHF hash function also uses two other RC4 functions, namely H-KSA* and H-PRGA*. The aim of the designers was to avoid the attacks by Idesteege and Preneel [73]. The H-KSA* function is similar to KSA but without the initialisation part. The H-PRGA* is similar to the original pseudorandom generation algorithm (PRGA) of RC4 with the difference that H-PRGA* does not generate output but changes the internal state. Note that padding of messages in RC4-BHF is different from the one used in RC4-Hash. A brief description of RC4-BHF is given in the next section. Full details about RC4-BHF can be found in [142]. The authors of RC4-BHF argue that their hash function is collision resistant and very efficient. They claim that RC4-BHF is roughly 4.6 times faster than SHA-1 and 16 times faster than MD4 [142].

In this chapter, we show that their claim about the security of RC4-BHF is not valid and we describe how to find collisions. We propose two attacks including a collision attack and distinguishing attack. In the first attack, by using the periodic nature of the internal states, we construct colliding message pairs with complexity of 2^{13} compress function operations. Also, we exploit this attack to make multicollisions. In the second attack, we show that the output of RC4-BHF is distinguishable from random sequences.

This chapter is structured as follows. Section 8.1 gives details of the RC4-BHF construction. Section 8.2 contains the main results of this study. In this section, after identifying weak points of the algorithm, we present a method for finding colliding messages, and also show how to construct a distinguisher for the hash function. Section 8.3 summarises the chapter.

8.1 Description of the RC4-BHF hash function

The hash function, RC4-BHF, was designed by Yu, Zhang and Hung in 2010. This hash function uses the algorithms similar to the KSA and PRGA used in the RC4 stream cipher. These blocks, however, are modified by the authors. The algorithms in question are:

- KSA (key-scheduling algorithm of RC4) – this function takes as an input a 64-byte message $M = (M[0], \ldots, M[63])$ and outputs the internal state $\langle S, i, j\rangle$, where $S = (S[0], \ldots, S[255])$ is a 256-byte sequence and j is a 1-byte index; also a 1-byte index called i. The function is described in Figure 2.7. Note that the KSA function is called at the very beginning of RC4-BHF to initialise the internal state.

- H-KSA* – the function takes two inputs: the message M and the internal state $\langle S, i, j\rangle$, as the input and provides an updated internal state. The full details are given in Figure 8.1.

1. **Input:** Message M and Internal State $\langle S, i, j\rangle$
2. **Output:** Updated Internal State $\langle S, i, j\rangle$
3. **for** $i = 0$ **to** 255
4. $j = (j + S[i] + M[i \mod 64]) \mod 256;$
5. $swap(S[i], S[j]);$
6. **end for**

FIGURE 8.1: H-KSA* function

- H-PRGA* (pseudorandom generation algorithm) – the function takes two inputs: an integer len and the internal state $\langle S, i, j\rangle$, as the input and generates an updated internal state on its input. The pseudocode of the function is given in Figure 8.2.

The building blocks (functions) are used to create a sequence of compression functions according to the well-known Merkle-Damgård (MD) structure. Given a binary message M of arbitrary length, the hashing algorithm proceeds through the following steps:

1. **padding** – a binary representation of the padding length is appended to the message and then an appropriate number of bits (constant or random) is attached

1. **Input:** Integer *len*, Internal State $\langle S, i, j \rangle$
2. **Output:** Updated Internal State $\langle S, i, j \rangle$
3. **for** $i = 0$ **to** len
4. $i = i + 1 \mod 256;$
5. $j = (j + S[i]) \mod 256;$
6. $swap(S[i], S[j]);$
7. **end for**

FIGURE 8.2: H-PRGA* function

so the number of bits in the resulting message is a multiple of 512. Consequently, the message can be represented as a sequence of $M = (M_1, \ldots, M_n)$, where each M_i is a 512-bit long (or alternatively 64-byte) sequence,

2. **compression** – the message M_1 is used to initialise the internal state $\langle S, i, j \rangle$ as follows

$$\langle S, i, j \rangle \leftarrow KSA(M_1)$$

and then the function H-PRGA* modifies the state depending on the length len_1 of the message M_1 ($len_1 = M_1 \mod 2^5$)

$$\langle S, i, j \rangle \leftarrow PRGA^*(len_1, \langle S, i, j \rangle).$$

For k; $k = 2, \ldots, n$, the internal states are updated step by step

$$\langle S, i, j \rangle \leftarrow PRGA^*(len_k, KSA^*(M_k, \langle S, i, j \rangle)$$

where $len_k = M_k \bmod 2^5$. Figure 8.3 illustrates the compression process. Note that the number of rounds applied in H-PRGA* is controlled by the integer $len_i = (M_i \bmod 2^5)$.

3. **truncation** – the output of the compression step consists of 258 bytes (256 bytes of the state together with 2 index bytes). The final hash value includes the least significant bit of each state byte and the indices. This means that the hash value is 272 bits long.

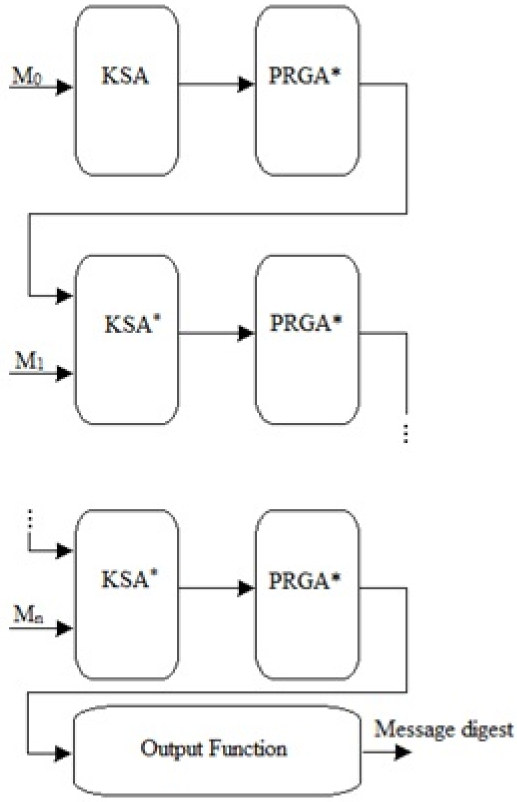

FIGURE 8.3: RC4-BHF scheme

The internal state of RC4-BHF is $\langle S, i, j\rangle$, where S indicates the internal state of RC4-BHF and (i, j) are the indices used in the KSA, H-KSA*, and H-PRGA* functions. The state can be divided into four parts S_0, S_1, S_2, S_3, where

$$\begin{aligned} S_0 &= \{s_k \mid 0 \leq k < 64\}, \\ S_1 &= \{s_k \mid 64 \leq k < 128\}, \\ S_2 &= \{s_k \mid 128 \leq k < 192\}, \\ S_3 &= \{s_k \mid 192 \leq k < 256\}, \end{aligned}$$

and s_k is the k-th byte of the internal state.

8.2 Cryptanalysis of RC4-BHF

In this section, we prove that RC4-BHF is not collision resistant. The proposed attack takes 2^{13} compression function operations and negligible memory. To apply a collision attack on the algorithm, first we describe the weaknesses of the hashing algorithm and then, by exploiting these weaknesses, we propose a collision attack and also present two distinguishers to tell apart the outputs generated by either RC4-BHF or a random number generator.

8.2.1 The weaknesses of RC4-BHF

Before describing our attack, we discuss properties of RC4-BHF that underpin our attack.

1. The internal state is controlled by the input messages and can be manipulated by an appropriate choice of message bytes. In particular, we will show that we can select messages in such a way that the internal state repeats periodically.

2. The execution of the function H-PRGA* is controlled by the integer *len*. Note that if $len = M_k \mod 2^5 = 0$, then the function H-PRGA* is not executed and can be skipped.

3. The index i is defined to be a byte or integer between 0 and 255. But, after each execution of the function H-KSA*, the index $i = 255$. Similarly, after each execution of H-PRGA*, the index i can be an integer between 0 and 31. These properties are not used in the collision attack, but they may be exploited to enhance a distinguishing attack on the scheme.

Now we can describe our collision attack on RC4-BHF.

8.2.2 Collision attack on RC4-BHF

The attack takes advantage of the periodicity of the function H-KSA* as formulated in the following theorem.

Theorem 12 *Given the function H-KSA* of RC4-BHF. Let the input internal state be* $S = \langle S_0, S_1, S_2, S_3, 63\rangle$*, the output internal state be* $S' = \langle S'_0, S'_1, S'_2, S'_3, 63\rangle$ *and the message sequence be* $M = (m_0, \ldots, m_{63})$*, where* $m_i = -(s_i - 1) \mod 256;\ 0 \leq i < 64$ *. Then*

$$KSA^*(\langle S_0, S_1, S_2, S_3, 63\rangle) = \langle S'_0 = S_0, S'_1 = S_2, S'_2 = S_3, S'_3 = S_1, 63\rangle$$

Proof 12 *It can be easily shown by applying H-KSA* to the internal state or by induction, such as a generalisation of Theorem 2 from [73]. Denote by* $\langle S^{(i)}, j^{(i)}\rangle$ *the internal state of RC4-BHF after the i-th step of the compression function H-KSA*. Note that*

$$M[i \bmod 64] = m_{i \bmod 64} = -(s_{i \bmod 64} - 1) \bmod 256.$$

First, we prove by induction that, for every $i < 256$, the following equations hold:

$$j^{(i)} = i + 63 \; mod \; 256, \; \textit{and}$$

$$S^{(i)}[i + 1 \; mod \; 256] = s_{i+1 \; mod \; 64},$$

$$S^{(i)}[i + 2 \; mod \; 256] = s_{i+2 \; mod \; 64},$$

$$\ldots$$

$$S^{(i)}[i + 64 \; mod \; 256] = s_{i+63 \; mod \; 64}.$$

It is clear that this holds before the first step, i.e., for $i = -1$, since $j^{(-1)} = 1$, $S^{(-1)}[0] = S[0] = s_0$ till $S^{(-1)}[63] = S[63] = s_{63}$. Assume that the condition holds after step i ($i < 255$). Then, the update of the pointer j in the $(i+1)$-th step is

$$\begin{aligned} j^{(i+1)} &= j^{(i)} + S^{(i)}[i+1] + M[i \; mod \; 64] \; mod \; 256 \\ &= ((i+63) + s_{i+1}) \; mod \; 256 \\ &+ (-(s_{i+1} \; mod \; 64 - 1) \; mod \; 256 \\ &= i + 64 \; mod \; 256. \end{aligned}$$

Thus, $S^{(i+1)}$ is found by swapping the $(i+1)$-th and $(i+64)$-th element of $S^{(i)}$. Hence $S^{(i+1)}[i + 64 \; mod \; 256] = S^{(i)}[i + 1 \; mod \; 256] = s_{i+1 \; mod \; 64}$. Of course, $S^{(i+1)}[i + 64 \; mod \; 256] = S^{(i)}[i + 2 \; mod \; 256] = s_{i \; mod \; 64}$. This implies that the condition also holds for step $i+1$. After 254 steps, all the elements of S have been rotated as follows:

$$\begin{gathered} S_0, S_1, S_2, S_3 \\ S_0, S_2, S_3, S_1 \end{gathered}$$

Observe that, if we apply the result of Theorem 12 in three consecutive calls to H-KSA* ($3 * 256$ steps), then the first state repeats. The situation is illustrated below:

$$\begin{aligned} & S_0, S_1, S_2, S_3 \\ \overset{H-KSA*}{\Longrightarrow} \; & S_0, S_2, S_3, S_1 \\ \overset{H-KSA*}{\Longrightarrow} \; & S_0, S_3, S_1, S_2 \\ \overset{H-KSA*}{\Longrightarrow} \; & S_0, S_1, S_2, S_3 \end{aligned}$$

This means that the application of the function H-KSA* three times to the state causes the same state to be reached. Note that, in addition to the above periodic behaviour of the internal states, one can choose other specific messages to achieve the same periodic behaviour with longer periods. In [73], this behaviour of the internal states of the RC4 stream cipher is investigated and the reader is referred to it for details. Note that the construction of colliding message pairs is easy. To apply the attack to RC4-BHF, we need to satisfy two conditions:

$$\begin{cases} \text{Condition 1: } j \text{ must be equal to 63, and} \\ \text{Condition 2: the least 5 significant bits of} \\ \qquad\qquad\qquad -(s_{63}-1) \; mod \; 256 \text{ must be zero.} \end{cases} \tag{8.1}$$

We expect that these requirements will be satisfied after testing $\approx 2^8 * 2^5$ messages.

8.2.3 Other Period Properties

In addition to cycles of length 3, other cycles can be found for the H-KSA* function. In fact, the term $M[i \; mod \; 64]$ in the functions KSA and H-KSA* can be applied to other input messages to construct internal states with periods 7, 15, 31, 63, 127.

In a similar way to Theorem 12, we can formulate appropriate conditions for the internal state and the message M. The results are summarised in Table 8.1.

Using Table 8.1, we can find other colliding messages. Finding an appropriate internal state requires the same effort (given by the time complexity column) for all cycles. Although we present two methods for the cycle equal to 3, these methods can be easily generalised for other cycles different from 3. In the next section we show how we can construct colliding messages.

8.2.4 Finding Collisions

To construct colliding messages, two methods can be used.

- **Method 1.** In this method, after applying message M_0, we obtain the suitable internal state to satisfy the conditions (8.1). Then, by applying message M_1 three times and padding the block, the hash value will be computed. Now, to generate other same hash value, we can repeat message M_1, as in blocks of 3, and finally apply a padding block, and compute the final hashing digest. The

Table 8.1: Properties and conditions to apply a collision attack on Algorithm for other cycles

	The Cycle Length	Condition 1	Condition 2	Time Complexity	Relations
1	7	$j = 31$	$-(s_{31} - 1)\ mod\ 64 = 0$	$2^8.2^5$	$m_i = -(s_i - 1)\ mod\ 256$, $m_i = m_{i+32},\ 0 \leq i < 32$
2	15	$j = 15$	$-(s_{15} - 1)\ mod\ 64 = 0$	$2^8.2^5$	$m_i = -(s_i - 1)\ mod\ 256$, $m_i = m_{i+16} = m_{i+32} = m_{i+64},\ 0 \leq i < 16$
3	31	$j = 7$	$-(s_7 - 1)\ mod\ 64 = 0$	$2^8.2^5$	$m_i = -(s_i - 1)\ mod\ 256$, $m_i = m_{i+8} = m_{i+16} = ... = m_{i+56},\ 0 \leq i < 8$
4	63	$j = 3$	$-(s_3 - 1)\ mod\ 64 = 0$	$2^8.2^5$	$m_i = -(s_i - 1)\ mod\ 256$,$m_i = m_{i+4} = m_{i+8} = ... = m_{i+60},\ 0 \leq i < 4$
5	127	$j = 1$	$-(s_1 - 1)\ mod\ 64 = 0$	$2^8.2^5$	$m_i = -(s_0 - 1)\ mod\ 256\quad i\ even$ $m_i = -(s_1 - 1)\ mod\ 256\quad i\ odd$
6	255	$j = 0$	$-(s_0 - 1)\ mod\ 64 = 0$	$2^8.2^5$	$m_i = -(s_0 - 1)\ mod\ 256$, $0 \leq i < 64$

following relations show how colliding messages can be constructed by method 1.

$$
\begin{aligned}
M^0 &= M_0\ ||\ Padding \\
M^1 &= M_0\ ||M_P||\ Padding \\
M^2 &= M_0\ ||M_P||M_P||\ Padding \\
&\ \ ... \\
M^n &= M_0\ ||M_P||...||M_P||\ Padding
\end{aligned}
$$

where $M_P = M_1||M_1||M_1$ and $M^i, 0 \leq i \leq n$, are colliding messages.

TABLE 8.2: Example for Method 1 including M_0, M_1, M_2 and generated hash value.

	M_0 (64-byte)	M_1 (64-byte)	Hash Value (272 bit)
1	03DE074C6CB1A37 A201C0C8187BA03 6E87A3CCC89C35D F742B14E0D6136F D13986858771176 85ABE130121F415 555ED9D506B5CF4 11DA3B3CF066C04 11DC5548	FF520B5101BFC98 C743E178B6521E7 A30C2E95C43FA77 B25E2E8BB5A3DD0 D9CF299EDA05B11 8CA1A57676E4FB8 041FF520BCED417 8A94D7FCD399347 AA9F5B40	0350EA16 4598FCEC 553FF9C6 9535B628 1F87F266 01D26F48 EEF72985 64265C95 007B
2	004BB7F857C5080 B47B92603AED617 99F14278CAA881C CD997991397E173 9FE27885236CD8A E0DBEF561157C71 0616EA139D1DAF7 5A5C0D9FC3CB222 0D879471	52D5AFD2DA1ACFA B46F514E32F9784 086CB228253A649 BE57835E699275A 799CC8D4F2D7F3D B95F8A21DAA37DD 94E4AC128BB6290 9E0B566560487BA 6EC3EA00	E42DD715 2E9EAB3F 4851B2A0 AFD358F2 B98DF972 0CD285FD CA314801 842ECF4B 0009

We expect that after $2^8.2^5 = 2^{13}$ executions of the compression function for random messages, a suitable M_0 can be found. Table 8.2 presents two examples of messages M_0, messages M_1 and hash values obtained using Method 1.

Note that changing the length of input message M^i has no effect on the padding content. So, we can construct an arbitrary number of colliding messages with the same hash value. This property can be used to compute multi-collisions.

- **Method 2.** The principle used is the same as in the previous method. We first find two messages M_0 and M_1 which satisfy the condition (8.1). After these two messages, messages M_1, M_3 can be made using Theorem 12. Finally, collision pairs can be made by the following relations:

$$\begin{aligned} M^0 &= M_0||M_1||M_1||M_1||M_2||Padding \\ M^1 &= M_0||M_2||M_3||M_3||M_3||Padding \\ &\ldots \end{aligned}$$

We expect that, after $(2^8.2^5)^2$ executions of the compression function for random messages, a suitable M_0 and M_2 can be found. Table 8.3 shows two examples of messages M_0, M_1 M_2, M_3, and hash values obtained using Method 2.

TABLE 8.3: Example for Method 2 including M_0, M_1, M_2, M_3 and generated hash values.

	M_0 (64-byte)	M_1 (64-byte)	M_2 (64-byte)	M_3 (64-byte)	Hash Value (272 bits)
1	273A4F51FAA4 A7CF3225E700 0A9ACDBCCABD 7CAC49991F5B B042CF9080C2 B7DCD756756F EFDBC42FE783 580CC6CC0A8D BDB335AFAC24 60F0E8B61DA7 3C953096	BAFB22B06E1F 20C50948DF65 A260D573927B 560625198784 0044523F1435 862FFC41E3CE BBDDB3D0A588 5890D759AACB 89CD72D2C1D3 BDABC7364505 F3EECF80	8FFD0B0A03E6 C6BF7714E1C0 BF9B71DE3AED 7139574F6556 57893E7155E2 7E14844B9CE8 B9DBAACC297B 352473E36D73 E1C5852DEA47 5DC6FCB75F0F 797AA7C2	4459229F9B50 A1E3F8A3A772 D464CA054F5D E62884295ADC B32609210CC0 E1A4DDBBC8AE 71E00A1243B7 7EAB017B1F48 0B4AE7958A1B 4EB8D2F902FD DAB01900	BEDE F059 71AC F6A3 AF04 5311 0417 28D5 D77E D338 5D58 4085 46A3 040B 5757 67FE 0029
2	7B45A927E089 C366BB75CB2E 06E9AD053F3A 007FBF33F060 48597B01DD73 E1F5D64A55EB 33AEF9D631B9 094C1B58562C 6306F784F1DB 3BB2BBC6E2C9 96178C36	5A154DFF7B6D 869E3DC2DF25 3F894F68D2B2 F7761C4674CA 6A8B5B944EF6 BFABB5792BC5 D89B7CA926D0 118C83698D6B A0BB9D906101 4CB68477F8A3 1D6536C0	5CFA81B5EE37 30B8FB0B01A3 5FB4C45B78E9 ECD37CD38830 1059752B16A0 D2B7C6D2B5E4 001F1C04E002 270C94C6843D 6A482A032DFE 4A1DB23882FE AEA65573	587C8C025B0B 462BB83B3C40 FFEDF472B6CC D8CF6299285A 8FCA0768E2EB 787D36EA2A6C 2E94B3019103 B1697BC3D057 00313FD496C7 521FFDC19BC8 59649580	11D3 C922 63F9 EFB1 65B6 370A A78D 6690 79B2 0706 2FE4 1228 2691 9A04 FBDF ED97 0019

8.2.5 Randomness properties of hash digest

As mentioned in Section 8.1, the hash value is generated by concatenating the least significant bits of each byte of the final internal state S and two bytes of indices i and j. Note that the first 256 bits of the hash value is the least significant bit of the numbers 0 till 255 which are swapped based on three functions KSA, H-KSA*,and H-PRGA*.

Although the positions of the integers are changed, their values are not modified, and this means that the hamming weight of the first 256 bits of hash values for every input message with arbitrary length will be exactly 128.

In addition, index i in the last round just depends on the last input message M_n, as $i = M_n \ mod \ 2^5$, and so it will be an integer between 0 and 31. The designers dedicated one byte for index i in the hash value. So first we can see that the three most significant bits for all input messages will be zero and second we can change the other five bits of the 259-th through to the 263-th bits by changing the five least significant bits of the last input message M_n with probability one. Of course, if we consider the effect of the padding block in the last round, then the index i will be fixed while the padding block does not change. These two weaknesses lead an attacker to a strong distinguisher with distinguishing advantage close to 1.

8.3 Summary

In this chapter we presented a collision attack on RC4-BHF. The attack required negligible memory and time complexity of 2^{13} compress function (H-KSA*) operations. The practicality of the attack has been demonstrated with some colliding messages for RC4-BHF. We have also shown that the hashing algorithm can be distinguishable from a truly random sequence with a probability close to one.

9

Conclusion

9.1 Thesis summary

This thesis investigates security evaluation of stream ciphers and hash functions based on stream ciphers.

Chapter 2 introduces several methods to design stream ciphers and cryptographic techniques used to analyse these ciphers.

In Chapter 3, the security of the WG-7 stream cipher has been studied. The presented distinguishing and key-recovery attacks show that the cipher is vulnerable and is not recommended for use. The distinguishing attack can detect the keystream generated by WG-7 from a random sequence with about $2^{13.5}$ keystream bits and with a negligible error probability. The proposed key-recovery attack also recovers the internal state of the cipher with a time complexity of about $O(2^{27})$. It is interesting to note whether other members of the WG family such as WG-8, WG-16, and WG-29 are resistant against the proposed cryptographic attacks. And the question is whether or not the designs can be kept secure.

Chapter 4 discovers some weak points of the Rakaposhi stream cipher. Firstly, due to the sliding property of the initialisation procedure, the distinguishing and key-recovery attacks can be applied to the cipher. The distinguisher needs only four related (key,IV) pairs. The key-recovery algorithm allows discovery of the 128-bit secret key after 2^9 initialisation operations. Secondly, the cipher is investigated when the linear and nonlinear registers enter short cycles. In this case, the internal state can be recovered with complexity rather than exhaustive search. The cipher uses a new concept known as a dynamic linear feedback shift register. As a new building block, the security properties of dynamic linear feedback shift registers would be a good place to investigate.

The next chapter, Chapter 5, identifies new security criteria of a specific design based on nonlinear feedback shift registers. The proposed idea applies a distinguishing attack to linearly filtered nonlinear feedback shift registers. The attack also extends the idea of linear combinations of linearly filtered nonlinear feedback shift registers. The proposed attacks allow the adversary to mount a linear attack to distinguish the output of the cipher and recover its internal state. This approach reveals how invulnerable the modified version of the Grain stream cipher is against distinguishing attacks. The following topics can be considered as new future work:

- Working on the mathematical background of NLFSR: NLFSRs have been attracting attention in theoretical and practical research. In the theoretical view, constructing full period NLFSRs, and determining the period and cycles of sequences generated by NLFSRs are still interesting problems to solve.

- Designing new structures with approved properties: Despite the weak mathematical background of NLFSRs, they are remarkable choices to design lightweight symmetric ciphers. One idea is to use NLFSRs along with a structure where their security properties cover their theoretical weak points. For instance, choosing cryptographic elements, such as LFSRs, and T-functions, or specific structures ,such as Grain based ciphers, which avoid short cycles or boost the period of keystream outputs, can guarantee that not only do NLFSRs not cause any structural weak point, but they also improve the security of the cipher.

- New analyses on NLFSR based stream ciphers: There is also room to investigate security of ciphers exploiting NLFSR.

Chapter 6 investigates the security of a new lightweight authenticated encryption function, known as NLM-MAC. The chapter presents critical cryptographic weak points leading to the key-recovery and forgery attacks. The internal state of the NLM-n generator can be recovered with a time complexity of about $n^{\log_2 7 \times 2}$ where the total length of the internal state is $2 \cdot n + 2$ bits. The attack needs about n^2 keystream bits. It is shown that the attacker is able to forge any MAC tag in real time by having only one pair (MAC tag, ciphertext). The proposed attacks are completely practical and break the scheme with negligible error probability. Interesting follow up questions would be how to choose the NLFSR to improve the security of the cipher and how to investigate new analyses of the cipher.

The last two chapters deal with two cryptographic functions based on the RC4 stream cipher. Chapter 7 shows some cryptographic weak points of the RC4(n,m) stream cipher. Firstly, two distinguishing attacks on the cipher have been proposed. Then, a key-recovery attack exploits a method to find the L-bit secret key with a time complexity of $(L/8) \cdot 2^n$. The implemented attack recovers the secret key of RC4(8,32) in less than one second on a standard PC.

Further, Chapter 8 proposes two cryptographic attacks, *the collision and distinguishing*, in the RC4-BHF which is based on the RC4 stream cipher. The first attack can find collisions for two different messages with time complexity of around 2^{13}, so it is very practical. In the presented distinguishing attack, it is shown that a distinguisher can detect the RC4-BHF sequence output from a random one. An interesting problem suitable for mid-term research, is how to design a secure shuffle based stream cipher using long word-oriented arrays along with simple operations, such as exclusive-or and modular addition.

9.2 Future research directions

The future research regarding the results and observations in this thesis are listed as follows.

- Design and cryptanalysis of new cryptographic primitives is necessary. Specifically, due to the variety of lightweight applications from RFID tags, smart cards, and sensor networks to 8-bit coprocessors, there is no single optimum solution to be employed to secure communications systems. To answer this challenge, the cryptographic community needs new research directions in designing suitable symmetric primitives, investigating new cryptanalytic techniques, and establishing new security criteria to estimate the security of the lightweight cryptosystems.

- The proposed attacks, mainly in chapters 4 and 7, showed that the initialisation procedures of stream ciphers play a key role in the security of the ciphers. The designer needs to check the strength of their ciphers against the threat models which give full control to adversaries to choose related keys or IV pairs. The potential weak points let the adversary apply the cryptographic attacks on the target cipher.

- Another area identified as needing future research is a security investigation of

NLFSRs against a combination of algebraic attacks and other cryptographic analyses, such as differential, linear, and guess-determine attacks. More importantly, it would be worthwhile to take a closer look at the specific structures, such as linearly filtered designs or a linear combination of two or more NLFSRs.

- Lastly, an interesting environment to research is finding low-weight linear relations derived from a linear feedback polynomial over $\mathbb{F}_{2^n}$ in a word-oriented LFSR. There are several algorithms proposed to find low-weight parity checks in bit-oriented LFSRs. The results can be directly used in the cryptanalysis of the WG family.

References

[1] SHA-3 Standardization. URL http://csrc.nist.gov/groups/ST/hash/sha-3/sha-3_standardization.html/. 3, 95

[2] The eSTREAM project. URL http://www.ecrypt.eu.org/stream/. 3

[3] European Network of Excellence in Cryptology. URL http://www.ecrypt.eu.org/. 3

[4] R. Anderson, E. Biham, and L. Knudsen. Serpent: A flexible block cipher with maximum assurance. URL http://www.cl.cam.ac.uk/~rja14/serpent.html. 3

[5] Frederik Armknecht. Improving Fast Algebraic Attacks. In *FSE*, pages 65–82, 2004. 26, 27, 34, 54

[6] Frederik Armknecht and Gwénolé Ars. Introducing a New Variant of Fast Algebraic Attacks and Minimizing Their Successive Data Complexity. In *Mycrypt*, pages 16–32, 2005. 27

[7] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. SHA-3 proposal BLAKE. URL https://131002.net/blake/. 3

[8] Steve Babbage and Matthew Dodd. The MICKEY Stream Ciphers. pages 191–209. Springer-Verlag, Berlin, Heidelberg, 2008. 3

[9] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. On the evolution of GGHN cipher. In *Proceedings of the 12th international conference on Cryptology in India*, INDOCRYPT'11, pages 181–195, Berlin, Heidelberg, 2011. Springer-Verlag. 85

[10] Côme Berbain, Henri Gilbert, and Antoine Joux. Algebraic and Correlation Attacks against Linearly Filtered Non Linear Feedback Shift Registers. pages 184–198. xii, 54, 55, 56, 61, 62, 64, 66, 69

[11] Côme Berbain, Olivier Billet, Anne Canteaut, Nicolas Courtois, Henri Gilbert, Louis Goubin, Aline Gouget, Louis Granboulan, Cédric Lauradoux, Marine Minier, Thomas Pornin, and Hervé Sibert. Sosemanuk, a Fast Software-Oriented Stream Cipher. pages 98–118. Springer-Verlag, Berlin, Heidelberg, 2008. 3, 16

[12] E. Berlekamp. Nonbinary BCH Decoding (Abstr.). *IEEE Trans. Inf. Theor.*, 14 (2):242–242, September 2006. ISSN 0018-9448. 17

[13] Daniel J. Bernstein. The Salsa20 Family of Stream Ciphers. pages 84–97. Springer-Verlag, Berlin, Heidelberg, 2008. 3

[14] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge functions. In *ECRYPT hash workshop*. 2007. 6

[15] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The KECCAK SHA-3 submission, January 2011. http://keccak.noekeon.org/. 3, 7

[16] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In *Proceedings of the theory and applications of cryptographic techniques 27th annual international conference on Advances in cryptology*, EUROCRYPT'08, pages 181–197, Berlin, Heidelberg, 2008. Springer-Verlag. 6

[17] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge-based pseudo-random number generators. In *Proceedings of the 12th international conference on Cryptographic hardware and embedded systems*, CHES'10, pages 33–47, Berlin, Heidelberg, 2010. Springer-Verlag. 6

[18] Eli Biham. New Types of Cryptanalytic Attacks Using Related Keys. In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 398–409. Springer Berlin Heidelberg, 1993. 11

[19] Eli Biham and Orr Dunkelman. Differential Cryptanalysis in Stream Ciphers. *IACR Cryptology ePrint Archive*, pages 1–18, 2007. 25

[20] Eli Biham and Jennifer Seberry. Py: A Fast and Secure Stream Cipher using Rolling Arrays. *Ecrypt submission*, 2005. 22

[21] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In *Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '90, pages 2–21, London, UK, 1991. Springer-Verlag. 89

[22] John Black, Martin Cochran, and Thomas Shrimpton. On the impossibility of highly-efficient blockcipher-based hash functions. In *Proceedings of the 24th annual international conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'05, pages 526–541, Berlin, Heidelberg, 2005. Springer-Verlag. 6

[23] Martin Boesgaard, Mette Vesterager, and Erik Zenner. The Rabbit Stream Cipher. pages 69–83. Springer-Verlag, Berlin, Heidelberg, 2008. 3

[24] An Braeken and Joseph Lano. On the (im)possibility of practical and secure nonlinear filters and combiners. In *Proceedings of the 12th international conference on Selected Areas in Cryptography*, SAC'05, pages 159–174, Berlin, Heidelberg, 2006. Springer-Verlag. 23

[25] An Braeken, Joseph Lano, Nele Mentens, Bart Preneel, and Ingrid Verbauwhede. SFINKS: A Synchronous Stream Cipher for Restricted Hardware Environments. In *SKEW - Symmetric Key Encryption Workshop*, 2005. 55, 72

[26] R. Brent, S. Gao, and A. Lauder. Random Krylov spaces over finite fields. *SIAM J. Discrete Math.*, 16:276287, 2003. 59

[27] M. Briceno, I. Goldberg, and D. Wagner. A pedagogical implementation of A5/1. 1999. URL `http://jya.com/a51-pi.htm`. 3

[28] C. Burwick, D. Coppersmith, E. DAvigno, R. Gennaro, S. Halevi, C. Jutla, S. Matyas, L. OConnor, M. Peyravia, D. Safford, and N. Zunic. MARS: a candidate cipher for AES. 1998. URL `http://domino.research.ibm.com/comm/researchprojects.nsf/pages/security.mars.html`. 3

[29] Christophe Cannière, Orr Dunkelman, and Miroslav Knežević. KATAN and KTANTAN – A Family of Small and Efficient Hardware-Oriented Block Ciphers. In *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '09, pages 272–288, Berlin, Heidelberg, 2009. Springer-Verlag. 18

[30] Christophe De Cannière and Bart Preneel. Trivium. In *The eSTREAM Finalists*, pages 244–266. 2008. 3, 18, 54

[31] Anne Canteaut and Michaël Trabbia. Improved fast correlation attacks using parity-check equations of weight 4 and 5. In *Proceedings of the 19th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'00, pages 573–588, Berlin, Heidelberg, 2000. Springer-Verlag. 24

[32] Chi-Kwong Chan and L. M. Cheng. Correlation properties of an improved summation generator with 2-bit memory. *Signal Process.*, pages 907–909, 2002. 71, 74

[33] Donghoon Chang, Kishan Chand Gupta, and Mridul Nandi. RC4-hash: a new hash function based on RC4. In *Proceedings of the 7th international conference on Cryptology in India*, INDOCRYPT'06, pages 80–94, Berlin, Heidelberg, 2006. Springer-Verlag. 95

[34] Joo Yeon Cho and Josef Pieprzyk. Distinguishing attack on SOBER-128 with linear masking. In *Proceedings of the 11th Australasian Conference on Information Security and Privacy*, ACISP'06, pages 29–39, Berlin, Heidelberg, 2006. Springer-Verlag. 23

[35] Carlos Cid, Shinsaku Kiyomoto, and Jun Kurihara. The RAKAPOSHI stream cipher. In *Proceedings of the 11th international conference on Information and Communications Security*, ICICS'09, pages 32–46, Berlin, Heidelberg, 2009. Springer-Verlag. 18, 39, 42

[36] Nicolas Courtois and Willi Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In *Advances in Cryptology - EUROCRYPT 2003, Warsaw, Poland, 2003, Proceedings*, pages 345–359. Springer, 2003. 26, 34, 54

[37] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In *ASIACRYPT*, pages 267–287, 2002.

[38] Nicolas T. Courtois. Fast Algebraic Attacks on Stream Ciphers with Linear Feedback. In *Crypto 2003, LNCS 2729*, pages 177–194. Springer. 26, 27, 34, 54

[39] Nicolas T. Courtois. Higher Order Correlation Attacks, XL algorithm and Cryptanalysis of Toyocrypt. In *ICISC 2002*, pages 182–199. Springer-Verlag, 2002. 26, 54

[40] Nicolas T. Courtois. Algebraic Attacks on Combiners with Memory and Several Outputs. In *Proc. of ICISC04*, pages 3–20, 2004. 26, 34

[41] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002. 3

[42] Ivan Damgård. A Design Principle for Hash Functions. In *Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '89, pages 416–427, London, UK, 1990. Springer-Verlag. 5

[43] Christophe De Cannière, Özgül Küçük, and Bart Preneel. Analysis of Grain's initialization algorithm. In *Proceedings of the Cryptology in Africa 1st international conference on Progress in cryptology*, AFRICACRYPT'08, pages 276–289, Berlin, Heidelberg, 2008. Springer-Verlag. 40

[44] Richard Drews Dean. *Formal aspects of mobile code security*. PhD thesis, Princeton, NJ, USA, 1999. Adviser-Appel, Andrew. 6

[45] DES. Data Encryption Standard. In *In FIPS PUB 46, Federal Information Processing Standards Publication*, 1977. 22

[46] W. Diffie and M. Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, pages 644–654, 2003. 2

[47] Itai Dinur and Adi Shamir. Breaking Grain-128 with dynamic cube attacks. In *Proceedings of the 18th international conference on Fast software encryption*, FSE'11, pages 167–187, Berlin, Heidelberg, 2011. Springer-Verlag. 61

[48] Elena Dubrova. A List of Maximum Period NLFSRs. *IACR Cryptology ePrint Archive*, 2012. 56

[49] Patrik Ekdahl and Thomas Johansson. A New Version of the Stream Cipher SNOW. In *Revised Papers from the 9th Annual International Workshop on Selected Areas in Cryptography*, SAC '02, pages 47–61, London, UK, 2003. Springer-Verlag. 16

[50] Håkan Englund, Thomas Johansson, and Meltem Sönmez Turan. A Framework for Chosen IV Statistical Analysis of Stream Ciphers. In K. Srinathan, C. Rangan, and Moti Yung, editors, *Progress in Cryptology INDOCRYPT 2007*, volume 4859 of *Lecture Notes in Computer Science*, pages 268–281. Springer Berlin / Heidelberg, 2007. 40

[51] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. The Skein Hash Function Family . URL `http://www.schneier.com/skein.html`. 3

[52] Niels Ferguson, Doug Whiting, Bruce Schneier, John Kelsey, Stefan Lucks, and Tadayoshi Kohno. Helix - Fast Encryption and Authentication in a Single Cryptographic Primitive. In *Proc. Fast Software Encryption 2003, volume 2887 of LNCS*, pages 330–346. Springer-Verlag, 2003. 72

[53] Eric Filiol. A New Statistical Testing for Symmetric Ciphers and Hash Functions. In *Proceedings of the 4th International Conference on Information and Communications Security*, ICICS '02, pages 342–353, London, UK, 2002. Springer-Verlag. 40

[54] Scott Fluhrer and David McGrew. Statistical Analysis of the Alleged RC4 Keystream Generator. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, and Bruce Schneier, editors, *Fast Software Encryption*, volume 1978 of *Lecture Notes in Computer Science*, pages 66–71. Springer Berlin / Heidelberg, 2001. 79

[55] Scott Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the Key Scheduling Algorithm of RC4 . In *RC4, Proceedings of the 4th Annual Workshop on Selected Areas of Cryptography*, pages 1–24, 2001. 79, 81

[56] Berndt Gammel, Rainer Gttfert, and Oliver Kniffler. Achterbahn-128/80: Design and analysis. In *In ECRYPT Network of Excellence - SASC Workshop Record*, pages 152–165, 2007. 18

[57] Berndt M. Gammel and Rainer Göttfert. Linear Filtering of Nonlinear Shift-Register Sequences. In *WCC*, pages 354–370, 2005. 54, 55, 64, 65

[58] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schlaffer, and Soren S. Thomsen. Grøstl a SHA-3 candidate. URL `www.groestl.info`. 3

[59] Jovan Golic. Intrinsic statistical weakness of keystream generators. In Josef Pieprzyk and Reihanah Safavi-Naini, editors, *Advances in Cryptology ASI-ACRYPT'94*, volume 917 of *Lecture Notes in Computer Science*, pages 91–103. Springer Berlin / Heidelberg, 1995. 58

[60] Jovan Dj. Golic. Correlation via Linear Sequential Circuit Approximation of Combiners with memory. In *EUROCRYPT*, pages 113–123, 1992. 54

[61] Jovan Dj. Golic. Linear statistical weakness of alleged RC4 keystream generator. In *Proceedings of the 16th annual international conference on Theory and application of cryptographic techniques*, EUROCRYPT'97, pages 226–238, Berlin, Heidelberg, 1997. Springer-Verlag. 79

[62] Jovan Dj. Golic, Andrew Clark, and Ed Dawson. Generalized Inversion Attack on Nonlinear Filter Generators. *IEEE Trans. Comput.*, pages 1100–1109, 2000. ISSN 0018-9340. 55

[63] Solomon Wolf Golomb. Shift Register Sequences. Aegean Park Press, 1982. 17, 54

[64] Guang Gong and Amr M. Youssef. Cryptographic properties of the Welch-Gong transformation sequence generators. *IEEE Transactions on Information Theory*, pages 2837–2846, 2002. 29

[65] Guang Gong, Kishan Chand Gupta, Martin Hell, and Yassir Nawaz. Towards a General RC4-Like Keystream Generator. In *FIRST SKLOIS CONFERENCE, CISC 2005*, pages 162–174. Springer-Verlag, 2005. 22, 80, 82, 85

[66] Finney H. An RC4 cycle that cannot happen. Newsgroup post in sci. crypt, September 1994. 85

[67] Daewan Han and Moonsik Lee. An algebraic attack on the improved summation generator with 2-bit memory. *Inf. Process. Lett.*, pages 43–46, 2005. ISSN 0020-0190. 71, 74, 75

[68] Philip Hawkes and Gregory G. Rose. Rewriting Variables: The Complexity of Fast Algebraic Attacks on Stream Ciphers. In *CRYPTO*, pages 390–406, 2004. 26, 27, 54

[69] M. Hell and T. Johansson. Linear Attacks on Stream Ciphers . pages 55–85. IOS Press, 2011. 23

[70] Martin Hell, Thomas Johansson, and Willi Meier. Grain - A Stream Cipher for Constrained Environments. *ECRYPT Stream Cipher Project*, 2005. 3, 18, 40, 54, 61, 65

[71] Martin Hell, Thomas Johansson, and W. Meier. Grain : A Stream Cipher for Constrained Environments. *Int. J. Wire. Mob. Comput.*, pages 86–93, 2007. 40, 61, 65

[72] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, pages 401–406, 1980. 12

[73] Sebastiaan Indesteege and Bart Preneel. Collisions for RC4-Hash. In *Proceedings of the 11th international conference on Information Security*, ISC '08, pages 355–366, Berlin, Heidelberg, 2008. Springer-Verlag. 96, 100, 102

[74] Takanori Isobe, Toshihiro Ohigashi, and Masakatu Morii. Slide Cryptanalysis of Lightweight Stream Cipher RAKAPOSHI. In Goichiro Hanaoka and Toshihiro Yamauchi, editors, *Advances in Information and Computer Security*, volume 7631

of *Lecture Notes in Computer Science*, pages 138–155. Springer Berlin Heidelberg, 2012. 40

[75] T. Johansson and A. Maximov. A linear distinguishing attack on Scream. In *Information Theory, 2003. Proceedings. IEEE International Symposium on*, pages 164–, 2003. 23

[76] Thomas Johansson and Fredrik Jönsson. Improved fast correlation attacks on stream ciphers via convolutional codes. In *Proceedings of the 17th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'99, pages 347–362, Berlin, Heidelberg, 1999. Springer-Verlag. 24

[77] Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In *Advances in Cryptology CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 99–213. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2004. 6

[78] Markku juhani O. Saarinen. Chosen-IV statistical attacks on estream stream ciphers. In *eSTREAM, ECRYPT Stream Cipher Project, Report 2006/013*, pages 5–19, 2006. 40

[79] Nathan Keller and Stephen D. Miller. Distinguishing attacks on stream ciphers based on arrays of pseudo-random words. *Inf. Process. Lett.*, pages 129–132, 2010. 23

[80] John Kelsey and Tadayoshi Kohno. Herding hash functions and the nostradamus attack. In *Proceedings of the 24th annual international conference on The Theory and Applications of Cryptographic Techniques*, EUROCRYPT'06, pages 183–200, Berlin, Heidelberg, 2006. Springer-Verlag. 6

[81] John Kelsey and Bruce Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In *Proceedings of the 24th annual international conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'05, pages 474–490, Berlin, Heidelberg, 2005. Springer-Verlag. 6

[82] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires.*, IX:161191, 1883. 9

[83] Aleksandar Kircanski and Amr Youssef. On the structural weakness of the GGHN stream cipher. pages 1–17. Springer New York, 2010. 80, 94

[84] Aleksandar Kircanski, Rabeah Al-Zaidy, and Amr Youssef. A new distinguishing and key recovery attack on NGG stream cipher. *Cryptography and Communications*, 1:269–282, 2009. 80

[85] Alexander Klimov and Adi Shamir. Cryptographic Applications of T-Functions. In Mitsuru Matsui and Robert Zuccherato, editors, *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 248–261. Springer Berlin / Heidelberg, 2004. 86

[86] Lars Knudsen, Willi Meier, Bart Preneel, Vincent Rijmen, and Sven Verdoolaege. Analysis Methods for (Alleged) RC4. In Kazuo Ohta and Dingyi Pei, editors, *Advances in Cryptology ASIACRYPT98*, volume 1514 of *Lecture Notes in Computer Science*, pages 327–341. Springer Berlin / Heidelberg, 1998. 79

[87] V. F. Kolchin. *Random graphs*. Cambridge University Press, New York, NY, USA, 1999. 59

[88] Pardeep Kumar and Hoon Jae Lee. NLM-MAC: Lightweight Secure Data Communication Framework Using Authenticated Encryption in Wireless Sensor Networks, Applied Cryptography and Network Security. *applied cryptography and network security*, pages 153–168, 2012. 72

[89] Hoon Jae Lee and Sang-Jae Moon. On an improved summation generator with 2-bit memory. *Signal Processing*, pages 211–217, 2000. 71

[90] Hoon Jae Lee, SangMin Sung, and HyeongRag Kim. NLM-128, an Improved LM-Type Summation Generator with 2-Bit memories. In *Proceedings of the 2009 Fourth International Conference on Computer Sciences and Convergence Information Technology*, ICCIT '09, pages 577–582, Washington, DC, USA, 2009. IEEE Computer Society. 18, 71

[91] Soh Yee Lee and Hoon Jae Lee. Hardware Implementation and Performance Analysis of NLM-128 Stream Cipher. In *6th International Conference Convergence and Hybrid Information Technology, ICHIT (2)*, volume 206 of *Communications in Computer and Information Science*, pages 446–453. Springer, 2011. 71

[92] Young Sil Lee, YoungMi Park, SangHan Lee, TaeYong Kim, and Hoon Jae Lee. RFID mutual authentication protocol with Unclonable RFID-tags. In *Mobile IT Convergence (ICMIC), 2011 International Conference on*, pages 74–77, 2011. 72

[93] Young Sil Lee, Tae Yong Kim, and Hoon Jae Lee. Mutual Authentication Protocol for Enhanced RFID Security and Anti-counterfeiting. In *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on*, pages 558–563, 2012. 72

[94] H. G. Liddell and R. Scott. *A Greek-English Lexicon*. Oxford University Press, 1995. 1

[95] Stefan Lucks. A failure-friendly design principle for hash functions. In *Proceedings of the 11th international conference on Theory and Application of Cryptology and Information Security*, ASIACRYPT'05, pages 474–494, Berlin, Heidelberg, 2005. Springer-Verlag. 6

[96] Yiyuan Luo, Qi Chai, Guang Gong, and Xuejia Lai. A Lightweight Stream Cipher WG-7 for RFID Encryption and Authentication. In *GLOBECOM*, pages 1–6, 2010. 29, 55

[97] Itsik Mantin. Predicting and Distinguishing Attacks on RC4 Keystream Generator. In *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, pages 491–506, 2005. 79

[98] Itsik Mantin and Adi Shamir. A Practical Attack on Broadcast RC4. In *Proc. of FSE01*, pages 152–164. Springer-Verlag, 2001. 33, 79

[99] J. Massey. Shift-register Synthesis and BCH Decoding. *IEEE Trans. Inf. Theor.*, 15(1):122–127, September 2006. ISSN 0018-9448. 17

[100] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In *Workshop on the theory and application of cryptographic techniques on Advances in cryptology*, EUROCRYPT '93, pages 386–397, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc. 22

[101] Mitsuru Matsui and Atsuhiro Yamagishi. A New Method for Known Plaintext Attack of FEAL Cipher. In RainerA. Rueppel, editor, *Advances in Cryptology EUROCRYPT 92*, volume 658 of *Lecture Notes in Computer Science*, pages 81–91. Springer Berlin Heidelberg, 1993. 22

[102] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2004. 6

[103] Alexander Maximov. Two Linear Distinguishing Attacks on VMPC and RC4A and Weakness of RC4 Family of Stream Ciphers. In *FSE*, pages 342–358, 2005. 79

[104] Alexander Maximov. Cryptanalysis of the "Grain" family of stream ciphers. In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, ASIACCS '06, pages 283–288. ACM, 2006. 61

[105] Alexander Maximov and Dmitry Khovratovich. New State Recovery Attack on RC4. In *CRYPTO*, pages 297–316, 2008. 79

[106] R. J. McEliece. A Public-Key Cryptosystem Based On Algebraic Coding Theory. *Deep Space Network Progress Report*, 44:114–116, January 1978. 2

[107] W. Meier and O. Staffelbach. Fast correlation attacks on stream ciphers. In *Lecture Notes in Computer Science on Advances in Cryptology-EUROCRYPT'88*, pages 301–314, New York, NY, USA, 1988. Springer-Verlag New York, Inc. 24

[108] Willi Meier and Othmar Staffelbach. Fast Correlation Attacks on Certain Stream Ciphers. *J. Cryptology*, pages 159–176, 1989. 54

[109] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996. 16, 17, 19, 20, 33

[110] R. Merkle and M.E. Hellman. Hiding information and signatures in trapdoor knapsacks. *Information Theory, IEEE Transactions on*, pages 525–530, 1978. 2

[111] Ralph C. Merkle. One way hash functions and DES. In *Proceedings on Advances in cryptology*, CRYPTO '89, pages 428–446, New York, NY, USA, 1989. Springer-Verlag New York, Inc. 5

[112] Jorge Carlos Mex-Perera and Simon J. Shepherd. Cryptanalysis of a summation generator with 2-bit memory. *Signal Process.*, pages 2025–2028, 2002. 71, 74

[113] Ilya Mironov. Not So Random Shuffles of RC4. In *Proc. of CRYPTO02*, pages 304–319. Springer-Verlag, 2002. 79

[114] Yassir Nawaz and Guang Gong. WG: A family of stream ciphers with designed randomness properties. *Inf. Sci.*, pages 1903–1916, 2008. 16, 29, 55

[115] Yassir Nawaz, Kishan Chand Gupta, and Guang Gong. A 32-bit RC4-like Keystream Generator. *IACR Cryptology ePrint Archive*, 2005:175, 2005. 80

[116] Kaisa Nyberg and Johan Wallén. Improved linear distinguishers for SNOW 2.0. In *Proceedings of the 13th international conference on Fast Software Encryption*, FSE'06, pages 144–162, Berlin, Heidelberg, 2006. Springer-Verlag. 23

[117] Mohammad Ali Orumiehchiha, S. Fahimeh Mohebbipoor, and Hossein Ghodosi. Cryptanalysis of MV3 Stream Cipher. In *Proceedings of the 7th International Conference on Cryptology and Network Security*, CANS '08, pages 240–251, Berlin, Heidelberg, 2008. Springer-Verlag. 23

[118] Mohammad Ali Orumiehchiha, Josef Pieprzyk, and Ron Steinfeld. Cryptanalysis of WG-7: a lightweight stream cipher. *Cryptography and Communications*, pages 277–285, 2012. 55

[119] Souradyuti Paul and Bart Preneel. A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher. In *FSE*, pages 245–259, 2004. 23, 79

[120] Souradyuti Paul and Bart Preneel. On the (In)security of Stream Ciphers Based on Arrays and Modular Addition. In *ASIACRYPT 2006*, pages 69–83. Springer, 2006. 23, 79, 80, 81, 89, 94

[121] W. T. Penzhorn. Correlation Attacks on Stream Ciphers: Computing Low-Weight Parity Checks Based on Error-Correcting Codes. In *Proceedings of the Third International Workshop on Fast Software Encryption*, pages 159–172, London, UK, 1996. Springer-Verlag. 54

[122] R. Rivest. The RC4 encryption algorithm. *RSA Data Security*, 1992. 3, 22

[123] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978. 2

[124] R. Rivest, M. Robshaw, R. Sydney, and Y. Yin. RC6 block cipher. August 1998. URL `http://people.csail.mit.edu/rivest/pubs/RRSY98.pdf`. 3

[125] Sondre Rønjom, Guang Gong, and Tor Helleseth. A survey of recent attacks on the filter generator. In *Proceedings of the 17th international conference on Applied algebra, algebraic algorithms and error-correcting codes*, AAECC'07, pages 7–17, Berlin, Heidelberg, 2007. Springer-Verlag. 55

[126] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. Twofish: A 128-bit block cipher. URL `http://www.schneier.com/papertwofishpaper.html`. 3

[127] Claude E. Shannon. Communication Theory of Secrecy Systems. *Bell Systems Technical Journal*, 28:656–715, 1949. 2

[128] Akihiro Shimizu and Shoji Miyaguchi. Fast Data Encipherment Algorithm FEAL. In *EUROCRYPT*, pages 267–278, 1987. 22

[129] T. Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications . *Information Theory, IEEE Transactions on*, pages 776–780, 1984. 23

[130] T. Siegenthaler. Decrypting a Class of Stream Ciphers Using Ciphertext Only. *Computers, IEEE Transactions on*, pages 81–85, 1985.

[131] T. Siegenthaler. Cryptanalysts Representation of Nonlinearly Filtered ML-Sequences. In Franz Pichler, editor, *Advances in Cryptology EUROCRYPT 85*, volume 219 of *Lecture Notes in Computer Science*, pages 103–110. Springer Berlin Heidelberg, 1986. 23

[132] Y. Tsunoo, T. Saito, H. Kubo, and T. Suzaki. A Distinguishing Attack on a Fast Software-Implemented RC4-Like Stream Cipher. In *IEEE Transactions on Information Theory,*, pages 3250 –3255, September 2007. 80, 81, 89, 94

[133] Serge Vaudenay and Martin Vuagnoux. Passive-Only Key Recovery Attacks on RC4. In Carlisle Adams, Ali Miri, and Michael Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *Lecture Notes in Computer Science*, pages 344–359. Springer Berlin / Heidelberg, 2007. 79

[134] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *Proceedings of the 24th annual international conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'05, pages 19–35, Berlin, Heidelberg, 2005. Springer-Verlag. 95

[135] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full SHA-1. In *Proceedings of the 25th annual international conference on Advances in Cryptology*, CRYPTO'05, pages 17–36, Berlin, Heidelberg, 2005. Springer-Verlag.

[136] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient collision search attacks on SHA-0. In *Proceedings of the 25th annual international conference on Advances in Cryptology*, CRYPTO'05, pages 1–16, Berlin, Heidelberg, 2005. Springer-Verlag. 95

[137] Doug Whiting, Bruce Schneier, Stefan Lucks, and F. Muller. Phelix: Fast Encryption and Authentication in a Single Cryptographic Primitive. In *eSTREAM, ECRYPT Stream Cipher Project Report 2005/027*, 2005. 72

[138] Hongjun Wu. The Hash Function JH. URL `http://www3.ntu.edu.sg/home/wuhj/research/jh/index.html`. 3

[139] Hongjun Wu. A New Stream Cipher HC-256. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 226–244. Springer, 2004. 3, 22

[140] Hongjun Wu. Cryptanalysis of a 32-bit RC4-like Stream Cipher. *IACR Cryptology ePrint Archive*, 2005:219, 2005. 80

[141] G. Z. Xiao and J. L. Massey. A spectral characterization of correlation-immune combining functions. *IEEE Trans. Inf. Theor.*, pages 569–571, 1988. 19

[142] Qian Yu, C.N. Zhang, and Xun Huang. An RC4-based hash function for ultra-low power devices. In *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, volume 1, pages V1–323–V1–328, 2010. 95, 96

[143] Haina Zhang and Xiaoyun Wang. Cryptanalysis of Stream Cipher Grain Family. In *Cryptology ePrint Archive, Report 2009/109*, 2009. 40

[144] Bartosz Zoltak. VMPC One-Way Function and Stream Cipher. In *LNCS*, pages 210–225. Springer-Verlag, 2004. 80

[145] Bartosz Zoltak. VMPC One-Way Function and Stream Cipher. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 210–225. Springer, 2004. 72