Some Studies on Selected Stream Ciphers. Analysis, Fault Attack & Related Results

by

Subhadeep Banik

under the supervision of
Professor Subhamoy Maitra

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

in the

Applied Statistics Unit
Indian Statistical Institute
203, B.T. Road, Kolkata-700 108, India.

Submitted: May 2014, Minor Revision: February 2015

"There are no shortcuts to any place worth going."

Beverly Sills

Abstract

Stream Ciphers are important Symmetric Cryptological primitives, built for the purpose of providing secure message encryption. As no formal security proofs exist, our confidence in these algorithms is largely based on the fact that intense cryptanalysis has been carried out over several years without revealing any weakness. This thesis makes some independent contributions to the cryptanalysis of a selection of stream ciphers.

In this thesis, we take a closer look at two stream ciphers viz. RC4+ designed by Maitra et al. at Indocrypt 2008 and GGHN designed by Gong et al. at CISC 2005. Both these ciphers were designed as viable alternatives to the RC4 stream cipher. It is shown that a distinguishing attack requiring around 2^{27} keystream bytes can be mounted on RC4+. Also, a differential fault attack on RC4+ requiring 2^{16} faults is presented. Thereafter, two cryptanalytic results are presented against the GGHN stream cipher. First, it is shown that numerous short cycles occur during the keystream generation phase of the cipher. Secondly, it is shown that a randomized variant of this cipher is expected to reach the all zero state in just around 2^{147} iterations, after which the cipher only produces the zero keystream byte at every iteration.

The Grain family of stream ciphers (Grain v1, Grain-128 and Grain-128a) designed by Ågren, Hell, Johansson, Maximov and Meier are a prominent family of stream ciphers especially since Grain v1 is included in the hardware portfolio of eStream. We first outline probabilistic methods that compute Key-IV pairs in the Grain family that can generate key-streams which are either almost similar in the initial segment, or exact shifts (the value of the shift being 2^{l_p}, where l_p is the length of the pad in bits used in the design of Grain) of each other throughout the generation of the stream. We then investigate the possibility of obtaining related Key-IV pairs that produce shifted keystream bits with smaller shifts. In a work by De Cannière et. al. at Africacrypt 2008, a method for finding related Key-IV pairs that produced i-bit shifted keystream (for Grain v1 and Grain-128) was proposed that required 4^i random trials. The method mainly took advantage of the fact that in both Grain v1 and Grain 128, the symmetric all 1 constant was used as the pad. We propose a new algorithm that improves the complexity to 2^i random trials. Furthermore, in the above work, it was observed that devising such a method for Grain-128a was not possible as the pad used in this cipher was asymmetric. However, we present a different technique to find related Key-IV pairs that produce 32-bit shifted keystream bits for Grain-128a in around 2^{32} random trials. We also present another method that finds related Key-IV pairs that produces shifted

keystream bits for shifts lesser than 32. The second method produces ϵ-bit shifted keystreams (for $0 < \epsilon < 32$) using $\frac{2^{32}}{1-2^{-\epsilon}}$ random trials.

Thereafter, we describe a set of three differential fault attacks on the Grain family, each of which is mounted under different experimental setups in which the attacker is granted varying degrees of freedom. The first attack assumes that the attacker can synchronize the timing of fault injection with a given stage of the cipher operation. Also, it is assumed that a fault causes a change in the logical value of precisely one of the flip-flops of the registers storing the internal state of the cipher (i.e a single bit-flip). Although he cannot choose the register location to be faulted, a flip-flop once faulted can be faulted multiple times. The second attack obviates the requirement of multiple faults on the same register location, but assumes that the attacker can still inject single bit-flipping time synchronized faults. The third attack requires the attacker to exercise minimal control over fault injections, i.e., the attack will be carried out under the assumption that the fault is neither time-synchronized, nor is there any guarantee that it causes a single bit flip at a random register location. Instead, the attacker is certain that the fault he injects toggles the logic value at a maximum of three contiguous flip-flops. This attack enlists the use of SAT solvers to reduce the number of faults required to complete the attack.

As far as the Differential Cryptanalysis of reduced round Grain v1 is concerned, the best results were those published by Knellwolf et al. in Asiacrypt 2011. In an extended version of the paper, it was shown that it was possible to retrieve five expressions in the Secret Key bits for a variant of Grain v1 that employs 97 rounds (in place of 160) in its Key Scheduling process using 2^{27} chosen IVs. The authors had arrived at the values of these Secret Key expressions by observing certain biases in the keystream bits generated by the chosen IVs. These biases were observed purely experimentally and no theoretical justification was provided for the same. In this work, we revisit Knellwolf's attacks on Grain v1 and try to provide a theoretical framework that will serve to prove the correctness of the attack. We describe a tool that serves to track the differential trails introduced in the cipher via the IV during the Key scheduling phase. By tracking these differentials it is possible to prove the bias in the keystream bits generated by the chosen IVs.

The stream cipher MICKEY 2.0 designed by Babbage and Dodd is also a part of eStream's hardware portfolio and hence a fairly important stream cipher. We describe a Differential Fault Attack under the assumption that the attacker can inject time-synchronized, single bit-flipping faults. Thereafter the attack is carried out if the injected fault toggles a maximum of three neighboring flip-flops. Thereafter, SAT solvers

are used to reduce the fault requirement, both in the single bit-flip and multiple bit-flip models.

A Scan-Chain is a popular DFT (Design for Testability) technique, which is used to check whether a chip is functioning properly or not. It provides the designer an easy way to ascertain whether the device has any structural defects or not. We analyze Scan-Chain based hardware design and the related vulnerabilities that may creep into a cryptosystem implemented with Scan Chains. We follow up on a work by Agrawal et. al. at Indocrypt 2008, in which, a scan based attack on the stream cipher Trivium was presented. We show why the same attack can not be extended to MICKEY 2.0, and suggest an alternative strategy to attack MICKEY 2.0 via Scan-Chains. Further, in the in the above work, an XOR gate based countermeasure was suggested to protect Scan-Chains from cryptanalytic attacks. We show that this countermeasure may fail to protect the underlying cryptosystem under certain classes of cryptanalytic attacks. It goes on to suggest a novel Double Feedback XOR-CHAIN countermeasure that is shown to be secure against the given class of cryptanalytic attacks. It is also shown than that such a Double Feedback XOR-CHAIN structure, like an ordinary Scan-Chain, may also be used for DFT purposes.

Acknowledgements

I had once held a belief, that I entertained almost religiously, that human beings are by nature selfish, that every thing we do is motivated purely by self-interest and nothing more. That period, is thankfully, long in the past. Now, I believe that although the propensity to be motivated by personal interests shall always persist in us, the instinct to come to the assistance of a fellow sufferer is also inherent in each of us. A human being is essentially this bundle of contradictions, and the lives we live are evidence of the fact that the desire to help has decidedly outshone the desire to harm.

To the people who made me the person I am, my family, I shall always be thankful. Although my father and grandmother could not be here to witness my graduation, I hope that they are proud of my achievements. To my mother and sister, who have stood by me come whatever may, no words are sufficient to repay the gratitude. To them I can only make a solemn promise, that I would make a better human being of myself.

My supervisor Prof Subhamoy Maitra had taken the risk of putting his faith in me, at a time when I was at the lowest point of my professional career. What exactly influenced him to take responsibility of overseeing my dissertation, I shall never know, but I am indeed grateful that he did eventually make the decision, for without his support I would be nowhere.

Whatever knowledge I have acquired in the domain of cryptography has been courtesy the technical inputs given by Palash da, Kishan da, and Mridul da. I must thank them for the tremendous work they have done over the past few years. I must also thank Prof Mandar Mitra and the members of the Research Fellow Advisory Committee of the Computer and Communication Sciences Division of ISI Kolkata for their role in facilitating my transfer to the Applied Statistics Unit.

Over the years I have been blessed with great friends, and the rapport and camaraderie I have shared with them have diminished the perpetual gloom that seems to hang over me. With Sonu, Purnendu, Shashank, Abhishek, Ranju and others I have shared every moment of pleasure and pain over the past few years. It has been truly remarkable such friends around. I would also take a moment to thank Goutam da, Santanu da, Srimanta da, Sibu da, Sanjay, Somindu, Sumit, Sourav, Toshan, Indranil, Subhabrata, Avik, Nilanjan, Amarjit and all other members of the ASU family for the constant encouragement and support. I would also extend my gratitude towards my co-authors Meltem, Anusha and Raghu for the support given by them.

As the shadows lengthen, and the general sense of darkness shrouds my existence, I understand it is time to bid adieu and I am but left to ruminate over the glorious

memories of the past. Over the years, I have benefited from the actions of so many people, so many known and unknown forces, that I shudder to think of living in a world where people would stop helping each other. To all those people, I shall, for ever and always, remain indebted.

Contents

List of Figures

List of Tables

List of Publications

This dissertation is a culmination of my research work at the Applied Statistics Unit at the Indian Statistical Institute Kolkata from the period 2011–2013. I hope that all the experience that I have gained during this period is adequately reflected into the thesis. Following are the list of publications that have been used in the thesis. Chapter 3 is based on the papers [21, 27]. Chapter 4 is based on [25, 26]. Chapter 5 is based on [22–24, 120]. Chapter 6 is based on [18]. Chapter 7 is based on [20, 28]. Chapter 8 is based on [19].

[18] Subhadeep Banik. Some Insights into Differential Cryptanalysis of Grain v1. In *ACISP*, volume 8544 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2014.

[19] Subhadeep Banik and Anusha Chowdhury. Improved Scan-Chain Based Attacks and Related Countermeasures. In *INDOCRYPT*, volume 8250 of *Lecture Notes in Computer Science*, pages 78–97. Springer, 2013.

[20] Subhadeep Banik and Subhamoy Maitra. A Differential Fault Attack on MICKEY 2.0. In *CHES*, volume 8086 of *Lecture Notes in Computer Science*, pages 215–232. Springer, 2013.

[21] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. On the Evolution of GGHN Cipher. In *INDOCRYPT*, volume 7107 of *Lecture Notes in Computer Science*, pages 181–195. Springer, 2011.

[22] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. A Differential Fault Attack on the Grain Family of Stream Ciphers. In *CHES*, volume 7428 of *Lecture Notes in Computer Science*, pages 122–139. Springer, 2012.

[23] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. A Differential Fault Attack on the Grain Family under Reasonable Assumptions. In *INDOCRYPT*, volume 7668 of *Lecture Notes in Computer Science*, pages 191–208. Springer, 2012.

[24] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. A Differential Fault Attack on Grain-128a Using MACs. In *SPACE*, volume 7644 of *Lecture Notes in Computer Science*, pages 111–125. Springer, 2012.

[25] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. Some Results on Related Key-IV Pairs of Grain. In *SPACE*, volume 7644 of *Lecture Notes in Computer Science*, pages 94–110. Springer, 2012.

[26] Subhadeep Banik, Subhamoy Maitra, Santanu Sarkar, and Meltem Sönmez Turan. A Chosen IV Related Key Attack on Grain-128a. In *ACISP*, volume 7959 of *Lecture Notes in Computer Science*, pages 13–26. Springer, 2013.

[27] Subhadeep Banik, Santanu Sarkar, and Raghu Kacker. Security Analysis of the RC4+ Stream Cipher. In *INDOCRYPT*, volume 8250 of *Lecture Notes in Computer Science*, pages 297–307. Springer, 2013.

[28] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. Improved Differential Fault Attack on MICKEY 2.0. *To appear in Journal of Cryptographic Engineering*, 2014. DOI : 10.1007/s13389-014-0083-9.

[120] Santanu Sarkar, Subhadeep Banik, and Subhamoy Maitra. Differential Fault Attack against Grain family with very few faults and minimal assumptions. *To appear in IEEE Transactions on Computers*, 2014. DOI: 10.1109/TC.2014.2339854.

Dedicated to my Father and Grandmother, who would have been the proudest people on Earth to see me graduate.

Chapter 1

Introduction

The word cryptography evokes varied sentiments in a wide and diverse range of people. While to an outsider, its applications are limited to certain inscrutable aspects of encryption and network security, to a researcher dabbling in this field, the possibilities are endless. Over the last few decades, the cryptological research fraternity has made some remarkable progress by formally redefining security notions and thus helping establish cryptology as a full-fledged discipline. Today, cryptology is heavily ingrained with the social fabric; it directly or indirectly affects in ways that most of us do not even realize. In the coming days, as the Internet phenomenon unravels and the whole world slowly becomes a giant network of computers interconnected by optical cables, an increasing number of people would be connected to one one another. It would not be unrealistic to project that almost half the world's population would become internet users in the near future. In such a scenario, the question of trust and safety assumes paramount importance. A lot of network users nowadays are worried about the safety of the data they store on devices as simple as a PC or sophisticated as a network database or a Cloud. Security of entities like passwords, financial transactions over the internet, state secrets etc. are also some of the common security issues staring at us in the face today. As the computer processors get faster and human beings are able to call upon larger computational resources, the so called proverbial adversary, out to compromise our security on the internet, only gets stronger and more powerful. Cryptology, to some extent, helps to allay the general sense of paranoia and skepticism that may develop as a result of this. It helps build confidence in users about the underlying network protocols already in place to ensure protection against threats. It tells us that it is OK to use our credit cards to make any purchase on the web, or store any kind of sensitive data over a network. It prepares us for the 21^{st} century, where people are more likely to spend over half their lives glued to a computer.

1.1 Introductory Notions

Until not very long ago, cryptography referred almost exclusively to encryption, which is the process of converting ordinary information (called plaintext) into unintelligible text (called ciphertext), rendering it unreadable by interceptors or eavesdroppers who have no knowledge of the encryption process. Decryption is the reverse, in other words, moving from the unintelligible ciphertext back to plaintext. A cipher is a pair of algorithms that govern the encryption and the reversing decryption processes. The detailed operation of a cipher is controlled both by the algorithm and in each instance by a key. The key is typically a small string over some alphabet used to initiate the encryption and decryption processes. The knowledge of the key is usually sufficient for any entity to both encrypt any plaintext and decrypt the corresponding ciphertext. Hence for message confidentiality purposes, the key is ideally a secret, known only to the communicants. A cryptosystem is the ordered list of elements of finite possible plaintexts, finite possible ciphertexts, finite possible keys, and the encryption and decryption algorithms which correspond to each key. Keys are important, as ciphers without variable keys can be trivially broken with only the knowledge of the cipher used and are therefore useless (or even counter-productive) for most purposes. Historically, ciphers were often used directly for encryption or decryption without additional procedures such as authentication or integrity checks.

Cryptanalysis is the term used to denote the study of methods for obtaining the meaning of encrypted plaintext by observing the ciphertext, without access to the key normally required to do so; i.e., it is the study of how to crack encryption algorithms or their implementations.

Although the 21^{st} century has seen the application and analysis of cryptological protocols assuming critical importance to individuals, organizations and governments alike, the earliest use of crypto can be dated back to 1900 BC when the Egyptians used secret code to carve text on stone tablets. However, the earliest instances of encryption were limited to use of the main classical cipher types as follows.

- Transposition ciphers, which rearranges the order of letters in a message (e.g., 'good morning' is encrypted to 'dogo ngrinom' in a simple rearrangement scheme). Scytale is an example of a transposition cipher used by the Spartan Military [7].
- Substitution ciphers, which systematically replaces letters or groups of letters with other letters or groups of letters (e.g., 'fly at once' becomes 'gmz bu podf' by replacing each letter with the one following it in the Latin alphabet). An early substitution cipher was the Caesar cipher, in which each letter in the plaintext was

replaced by a letter some fixed number of positions further down the alphabet. It is known that Julius Caesar used it with a shift of three letters to communicate with his generals [7].

The discipline of Steganography which deals with hiding the existence of a message so as to keep it confidential, was also first developed in parallel. An early example, from Herodotus, concealed a message-a tattoo on a slave's shaved head-under the regrown hair [7]. Another Greek method was developed by Polybius (now called the "Polybius Square"). More modern examples of steganography include the use of invisible ink, microdots, and digital watermarks to conceal information. This discipline is considered to widely different from cryptology as it aims to conceal the existence of a message, whereas classical cryptology would only aim to alter the intelligibility of a message [7].

The systematic evolution of cryptology over the ages makes for fascinating reading. The most complete non-technical account of the subject is Kahn's "The Codebreakers" [84]. This book traces cryptography from its initial and limited use by the Egyptians some 4000 years ago, to the 20^{th} century where it played a crucial role in the outcome of both world wars.

1.2 Cryptology in the Modern Era

Over the years, the method of storing and transmitting data has changed dramatically. As we move into the era of computers, almost all data is stored electronically in one format or the other. Thus today, the ability to do any kind of data-processing is much greater than in the past and all indications point to astronomical increases in computational resources in the foreseeable future. In this scenario, any kind of communication would involve transmission over an untrusted medium, which includes just about any network, particularly the Internet. Furthermore, the underlying network used for communication may not only be untrusted but also seek to actively tamper with with any data traveling through it. In such a scenario, cryptology outlines the specific security requirements that must be fulfilled:

- **Confidentiality/Secrecy** is the security notion used to imply that the content of information is available to all but those authorized to have it.
- **Data integrity** is the notion which addresses the issue of unauthorized alteration of data. To ensure data integrity, the underlying protocol must have the ability to detect data manipulation by any unauthorized entity. Data manipulation includes

all operations that result in insertion, deletion, or substitution of characters in a message.

- **Authentication** is the notion dealing with the process of proving one's identification. It is imperative that both the sender and receiver identify each other before communication. Furthermore, information delivered over a channel should be authenticated as to origin, date of origin, data content, time sent, etc. For these reasons this aspect of cryptography is usually subdivided into two major classes: entity authentication and data origin authentication.
- **Non-repudiation** deals with preventing an entity from denying previous commitments or actions. When disputes arise due to an entity denying that certain actions were taken, a means to resolve the situation is necessary.

There are, in general, three types of cryptographic schemes typically used to accomplish these goals: secret key (or symmetric) cryptography, public-key (or asymmetric) cryptography, and hash functions, each of which is described in the following Section.

1.3 Types of Cryptographic Schemes

Broadly speaking, there are in general three classes of Cryptosystems: Symmetric Key, Public Key and Hash Functions [107]. We will take a brief look at these generic classes:

1.3.1 Symmetric Key Cryptosystems

In Symmetric Key cryptography, the same key is used for both encryption and decryption. The algorithm, used to encrypt the plaintext, is assumed to be publicly known. The sender uses the key (or some set of rules) to encrypt the plaintext and sends the ciphertext to the receiver. The receiver applies the same key (or rule-set) to decrypt the message and recover the plaintext. Because a single key is used for both functions, secret key cryptography is also called symmetric encryption. With this form of cryptography, it is obvious that the key must be known to both the sender and the receiver; that, in fact, is the secret. The biggest difficulty with this approach, of course, is the distribution of the key.

Symmetric Key Cryptosystems are generally categorized as being either stream ciphers or block ciphers.

- **A Stream cipher** is an algorithm that takes a Secret Key K as input, which is usually a small binary string of around $80-256$ bits, and applies a set a of rules to

produce a long sequence of **pseudorandom** bits/bytes/words commonly known as the **keystream**. The sequence is called pseudorandom, because it should not be distinguished from a truly random sequence in practical time. We will discuss the notion of pseudorandomness in detail in the next chapter.

This sequence of bits/bytes/words is usually xored with each bit/byte/word of the plaintext to produce the encrypted ciphertext. So, if $P = p_0, p_1, p_2, \ldots$ represents the bits/bytes/words of the plaintext, and $\kappa = k_0, k_1, k_2, \ldots$ represents the keystream bits/bytes/words produced by the Stream Cipher using the Secret Key K, then the encryption rule is given by

$$c_i \ = \ p_i \oplus k_i \ , \ \forall i,$$

where $C = c_1, c_2, \ldots$ represents the ciphertext bits/bytes/words. Since the Secret Key is already known to the receiver, he can compute the keystream bits $k_0, k_1, \ldots$ at his end, which are then used to decrypt the ciphertext as follows:

$$p_i \ = \ c_i \oplus k_i \ , \ \forall i.$$

Stream ciphers can be broadly classified into two categories. **Self-synchronizing** stream ciphers calculate the i^{th} keystream bit/byte/word k_i as a function of the previous n bits in the keystream, i.e. $k_{i-1}, k_{i-2}, \ldots, k_{i-n}$. An example of such a stream cipher is Moustique [49]. An obvious problem while using such a cipher is error propagation; a garbled bit/byte/word in transmission will result in n garbled bits/bytes/words at the receiving side. On the other hand, **Synchronous** stream ciphers generate the keystream bits/bytes/words in a fashion independent of any of the previous keystream bits/bytes/words. The same keystream generation function is used at sender and receiver ends. While synchronous stream ciphers do not propagate transmission errors, they are, by their nature, periodic so that the keystream will eventually repeat. All the stream ciphers analyzed in this thesis (Grain Family, MICKEY 2.0, RC4+, GGHN) are synchronous stream ciphers.

- **A Block Cipher,** in contrast, is a deterministic algorithm that encrypts fixed-length groups of bits, called **blocks**, with an unvarying transformation that is specified by a symmetric key. Unlike in Stream cipher encryption, where each plaintext bit/byte/word is xored with a keystream element, Block ciphers encrypt/decrypt each block of data separately using a fixed set of encryption/decryption rules. Block ciphers are important elementary components in the design of many cryptographic protocols, and are widely used to implement encryption of bulk data. Mathematically, if P_b denotes a block of b plaintext bits, and K is the Secret Key,

then the Block Cipher is completely defined by the Encryption and Decryption functions E_K and D_K respectively. The functions E_K, D_K are usually permutations on the space $\{0,1\}^b$ and are completely defined by the Secret Key K, with D_K being the inverse permutation of E_K. The encryption of the plaintext block P_b is given by $C_b = E_K\ (P_b)$, and since D_K is the inverse of E_K, the decryption rule is given by $D_K\ (C_b) = D_K\ (E_K\ (P_b)) = P_b$.

Lucifer [127] is generally considered to be the first civilian block cipher, developed at IBM in the 1970s based on work done by Horst Feistel. A revised version of the algorithm was adopted as a U.S. government Federal Information Processing Standard: FIPS PUB 46 Data Encryption Standard (DES) [2]. It was chosen by the U.S. National Bureau of Standards (NBS) after a public invitation for submissions and some internal changes by NBS (and, potentially, the NSA). DES was publicly released in 1976 and has been widely used.

DES has been superseded as a United States Federal Standard by the AES (Advanced Encryption Standard), adopted by NIST in 2001 after a 5-year public competition [1]. The cipher was developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, and submitted under the name Rijndael [50].

AES has a fixed block size of 128 bits and a key size of 128, 192, or 256 bits, whereas Rijndael can be specified with block and key sizes in any multiple of 32 bits, with a minimum of 128 bits. The blocksize has a maximum of 256 bits, but there are no specific bounds imposed on the keysize. AES operates on a 4×4 column-major order matrix of bytes, termed the state (versions of Rijndael with a larger block size have additional columns in the state).

1.3.2 Public Key Cryptosystems

Public-key cryptography, also known as asymmetric cryptography, refers to a cryptographic algorithm which requires two separate keys, one of which is secret (or private) and the other of which is public. Although different, the two parts of this key pair are mathematically linked. The public key is used to encrypt plaintext or to verify a digital signature; whereas the private key is used to decrypt ciphertext or to create a digital signature. The term "asymmetric" stems from the use of different keys to perform these opposite functions, each the inverse of the other - as contrasted with conventional ("symmetric") cryptography which relies on the same key to perform both.

Public-key algorithms are based on mathematical problems which currently admit no efficient solution that are inherent in certain integer factorization, discrete logarithm, and elliptic curve relationships. It is computationally easy for a user to generate his/her

own public and private key-pair and to use them for encryption and decryption. The strength lies in the fact that it is "impossible" (computationally unfeasible) for a properly generated private key to be determined from its corresponding public key. Thus the public key may be published without compromising security, whereas the private key must not be revealed to anyone not authorized to read messages or perform digital signatures. Public key algorithms, unlike symmetric key algorithms, do not require a secure initial exchange of one (or more) secret keys between the parties.

There are three main uses for public-key cryptography:

- Key Exchange Protocols, in which two parties communicate among themselves to obtain a shared Secret Key in such a manner that any third person eavesdropping on their conversation has does not become privy to this shared Secret Key. This protocol is used widely to establish Secret Keys between two parties for subsequent use in Symmetric Key Algorithms.
- Public-key encryption, in which a message is encrypted with a recipient's public key. The message cannot be decrypted by anyone who does not possess the matching private key, who is thus presumed to be the owner of that key and the person associated with the public key. This is used in an attempt to ensure confidentiality.
- Digital signatures, in which a message is signed with the sender's private key and can be verified by anyone who has access to the sender's public key. This verification proves that the sender had access to the private key, and therefore is likely to be the person associated with the public key. This also ensures that the message has not been tampered, as any manipulation of the message will result in changes to the encoded message digest, which otherwise remains unchanged between the sender and receiver.

An analogy to public-key encryption is that of a locked mail box with a mail slot. The mail slot is exposed and accessible to the public - its location (the street address) is, in essence, the public key. Anyone knowing the street address can go to the door and drop a written message through the slot. However, only the person who possesses the key can open the mailbox and read the message.

An analogy for digital signatures is the sealing of an envelope with a personal wax seal. The message can be opened by anyone, but the presence of the unique seal authenticates the sender.

A central problem with the use of public-key cryptography is confidence/proof that a particular public key is authentic, in that it is correct and belongs to the person or entity

claimed, and has not been tampered with or replaced by a malicious third party. The usual approach to this problem is to use a public-key infrastructure (PKI) [10], in which one or more third parties, known as certificate authorities, certify ownership of key pairs. Pretty Good Privacy (PGP) [65], is a data encryption program, which in addition to being a certificate authority structure, has used a scheme generally called the "web of trust", which decentralizes such authentication of public keys by a central mechanism, and substitutes individual endorsements of the link between user and public key.

Examples of well-regarded asymmetric key techniques for varied purposes include:

- Diffie-Hellman key exchange protocol [53]
- RSA encryption algorithm (PKCS#1) [5]
- Cramer-Shoup cryptosystem [48]
- DSS (Digital Signature Standard), which incorporates the Digital Signature Algorithm [3]
- McEliece cryptosystem [106]
- NTRUEncrypt cryptosystem [77]

1.3.3 Hash Functions

A hash function is any algorithm that maps data of arbitrary length to data of a fixed length, and are primarily used to construct message digests. Thus, it can be represented by a map $h : \{0,1\}^* \to \{0,1\}^n$. A hash function should be deterministic: when it is invoked twice on identical data (e.g. two strings containing exactly the same characters), the function should produce the same value. This is crucial to the correctness of virtually all algorithms based on hashing. Hash functions are typically not invertible, meaning that it is not possible to reconstruct the input data x from its hash value $h(x)$ alone. This is known as the First Preimage Resistance property of Hash Functions.

Since, hash functions map a relatively large domain to a fixed size range, there are bound to be distinct input strings x_1, x_2 so that $h(x_1) = h(x_2)$. This event is known as a collision of the hash function. However, such collisions are generally difficult to find in practice. This is known as the Collision Resistance property of Hash Functions. Also, given some hash value $h(x)$, it is difficult to find some y such that $h(x) = h(y)$. This is known as the Second Preimage Resistance property of Hash Functions.

Hash algorithms are typically used to provide a digital fingerprint of a file's contents, often used to ensure that the file has not been altered by an intruder or virus. Hash

functions are also commonly employed by many operating systems to encrypt passwords. Hash functions, then, provide a measure of the integrity of a file.

The NIST hash function competition [6] was an open competition held by the US National Institute of Standards and Technology (NIST) to develop a new hash function called SHA-3 to complement the older SHA-1 and SHA-2. The competition was formally announced on November 2, 2007. After 5 years of competition, the design proposal entitled Keccak [34], submitted by Guido Bertoni, Joan Daemen, Gilles Van Assche and Michaël Peeters, was declared the winner of the competition.

One may ask why there are three different types of cryptographic schemes and why only one encryption paradigm is not sufficient. The answer is that each scheme is optimized for some specific application(s). Hash functions, for example, are well-suited for ensuring data integrity because any change made to the contents of a message will result in the receiver calculating a different hash value than the one placed in the transmission by the sender. Since it is highly unlikely that two different messages will yield the same hash value, data integrity is ensured to a high degree of confidence. Symmetric key cryptography, on the other hand, is ideally suited to encrypting messages, thus providing privacy and confidentiality. The sender can generate a session key on a per-message basis to encrypt the message; the receiver, of course, needs the same session key to decrypt the message. Key exchange, of course, is a key application of public-key cryptography. Asymmetric schemes can also be used for non-repudiation and user authentication; if the receiver can obtain the session key encrypted with the sender's private key, then only this sender could have sent the message. Public-key cryptography could, theoretically, also be used to encrypt messages although this is rarely done because secret-key cryptography operates many times faster than public-key cryptography.

1.4 Stream Ciphers and the eStream Project

As defined earlier, a stream cipher is a symmetric key primitive where plaintext bits/bytes are combined with a pseudorandom bit/byte/word stream called keystream. In a stream cipher each plaintext bit/byte/word is encrypted one at a time with the corresponding bit/byte of the keystream, to give a bit/byte/word of the ciphertext stream. In practice, the combining operation is usually a bitwise exclusive-or (xor). Mathematically, a stream cipher is a set of two functions F, G and a finite state vector σ (known simply as internal state) whose value is updated continuously by the function F. If σ_t is the value of the state vector at some time instant t, K denotes the Secret Key used

by the cipher, and p_t denotes the t^{th} plaintext bit/byte then:

$$\begin{aligned} \sigma_{t+1} &= F\ (\sigma_t, p_t, K) \\ k_t &= G\ (\sigma_t, p_t) \end{aligned}$$

where k_t denotes the t^{th} keystream bit/byte. Note that for synchronous stream ciphers, the output of the functions $F,\ G$ do not depend on p_t.

1.4.1 One-Time Pad and Perfect Secrecy

Stream ciphers can be viewed as approximating the action of a proven unbreakable cipher, the one-time pad (OTP). A one-time pad uses a keystream of completely random digits, i.e., the keystream must be chosen uniformly randomly from the set of keystream digits. This means that each element of the set of keystream digits must have equal probability of being chosen. The keystream is combined with the plaintext digits one at a time to form the ciphertext. The digits may be elements of any arbitrary set, e.g. the English Alphabet (for which keystream digit and plaintext digit are added modulo 26 to produce the cipher text), but for most practical purposes, as in the case of stream ciphers, the digit is a bit/byte and the encryption operation is simply a bitwise xor of the plaintext and keystream.

One-time pads are "information-theoretically secure" in that the encrypted message (i.e., the ciphertext) provides no information about the original message to a cryptanalyst (except the maximum possible length of the message). This is a very strong notion of security first developed during the second World War by Claude Shannon and proved, mathematically, to be true for the one-time pad by Shannon about the same time. Shannon proved, using information theory considerations, that the one-time pad has a property he termed **perfect secrecy**; that is, the ciphertext gives absolutely no additional information about the plaintext. We will give a modified although equivalent proof of Shannon's perfect secrecy of the OTP. Before we do that, let us state a few notations: We will let $P, C, \mathcal{K}$ denote the ℓ-bit plaintext, ciphertext and the keystream respectively, and let $\mathcal{P}, \mathcal{C}, \mathcal{K}$ denote the set of all ℓ-bit plaintexts, ciphertexts and keystream respectively.

Definition 1.1. An encryption scheme Enc satisfies perfect secrecy if for all plaintexts $P_1, P_2 \in \mathcal{P}$ and all ciphertexts $C \in \mathcal{C}$, we have

$$\mathsf{Pr}_{\mathcal{K}\in\mathcal{K}}\ [Enc(\mathcal{K}, P_1) = C]\ =\ \mathsf{Pr}_{\mathcal{K}\in\mathcal{K}}\ [Enc(\mathcal{K}, P_2) = C]$$

where both probabilities are taken over the choice of $\mathcal{K} \in \mathcal{K}$ and over any other randomness introduced, if any, by the algorithm *Enc*.

In other words, the attacker cannot even get any partial information about the message from the ciphertext since all messages give identical distributions on the ciphertext.

Theorem 1.2. *OTP encryption satisfies the perfect secrecy requirement.*

Proof. Take any $P \in \mathcal{P}$ and any $C \in \mathcal{C}$, and let $\mathcal{K}^* = P \oplus C$. Note that:

$$\begin{aligned} \mathsf{Pr}_{\mathcal{K}\in\mathcal{K}}\left[Enc(\mathcal{K}, P) = C\right] &= \mathsf{Pr}_{\mathcal{K}\in\mathcal{K}}\left[\mathcal{K} \oplus P = C\right] \\ &= \mathsf{Pr}_{\mathcal{K}\in\mathcal{K}}\left[\mathcal{K} = C \oplus P\right] \\ &= \mathsf{Pr}_{\mathcal{K}\in\mathcal{K}}\left[\mathcal{K} = \mathcal{K}^*\right] = \frac{1}{2^\ell} \end{aligned}$$

Since the equation holds for every $P \in \mathcal{P}$, it follows that for every $P_1, P_2 \in \mathcal{P}$ we have

$$\mathsf{Pr}_{\mathcal{K}\in\mathcal{K}}\left[Enc(\mathcal{K}, P_1) = C\right] = \mathsf{Pr}_{\mathcal{K}\in\mathcal{K}}\left[Enc(\mathcal{K}, P_2) = C\right] = \frac{1}{2^\ell}$$

□

Despite Shannon's proof of its security, the one-time pad has serious drawbacks in practice:

- It requires perfectly random one-time pads which must have the same length as the message to be encrypted, which is a non-trivial software requirement. The only practical solution, in this respect, is a stream cipher, which takes as input a small random binary string as input and produces a sequence of pseudorandom bits.
- Any segment of the keystream digits once used can not be reused, hence the name One Time Pad. Indeed, because if the same $\mathcal{K}$ is used to encrypt two plaintexts P_1, P_2 to give $C_1 = P_1 \oplus \mathcal{K}$ and $c_2 = P_2 \oplus \mathcal{K}$ then the attacker can simply xor C_1, C_2 to get

  $$C_1 \oplus C_2 = P_1 \oplus \mathcal{K} \oplus P_2 \oplus \mathcal{K} = P_1 \oplus P_2$$

  Now, if the attacker somehow knows one of the plaintexts P_1 then he can easily get to know the value of P_2 by computing $P_2 = C_1 \oplus C_2 \oplus P_1$.

1.4.2 Using an Initialization Vector

An Initialization Vector (IV) is a fixed-size input to a cryptographic primitive that is typically required to be random or pseudorandom. Randomization is crucial for encryption schemes to achieve semantic security, a property whereby repeated usage of the scheme under the same key does not allow an attacker to infer relationships between segments of the encrypted message.

In the context of stream ciphers, we know that the same key will always produce the same keystream. This means that repeatedly using the same key is just as bad as reusing a One Time Pad. To solve this problem one must use a different Secret Key for every message that is to be encrypted, which is not always feasible. To overcome this, the concept of initialization vectors is useful. The IV is a typically a random binary string that changes with every instance of the cipher that is used to add some randomness to the output of the cipher. In this case the design of the stream cipher somehow allows the IV and the Secret Key information to be loaded on to the state vector of the stream cipher. Since this value of the IV is random and unique, it makes the output of the stream cipher different than other outputs, even if the same key is used. This is useful when key exchange is expensive. Initialization Vectors are generally made public, and all attack models assume that the adversary has knowledge of the value of the IV.

1.4.3 Attack models

Kerkchoffs's Principle: In 1883, Auguste Kerckhoffs wrote two journal articles on La Cryptographie Militaire [87], in which he stated that a cryptosystem should be secure even if everything about the system, except the Secret Key, is public knowledge. This statement is the fundamental basis of all cryptological schemes and cryptosystems are specifically designed so that any attacker who knows every detail of the cryptosystem, except the Secret Key would be unable to break the system. Keeping the design secret in commercial domain has no scientific justification. It may be leaked easily. The design should be such that the designer himself cannot break the system without knowing the key.

Thus any attack we will discuss, in the context of a stream cipher, assumes the fact that the attacker is aware of the exact design details. The attack model we will discuss in this context is a **chosen plaintext attack**. In this model it is assumed that the attacker is able to choose one or more than one plaintexts P_i obtain the corresponding ciphertexts C_i produced due to the encryption. Note that, for a stream cipher, the bitwise xor of the plaintext and ciphertext bits reveals the keystream. Thus the basic aim of the attacker,

when attacking a stream cipher, is to gain some meaningful knowledge of the Secret Key by simply observing one or more keystream sequences.

Goals of the Attacker: The ultimate goal of any attacker is to determine the Secret Key that is used to produce the given keystream sequence. However the attacker may also wish to do one of the following:

State Recovery Attack: The attacker may wish to compute one of the values of the state vector σ_t at some time instant t. This allows forward generation of the keystream, i.e., the attacker is able to compute the keystream sequence starting from $k_t, k_{t+1}, \ldots$ If the state update function F is invertible and independent of the Secret Key, then this allows to move backwards and compute $\sigma_{t-1}, \sigma_{t-2}, \ldots, \sigma_0$, and also the Secret Key.

Distinguishing Attack: The attacker may wish to distinguish the output keystream sequence produced by a stream cipher from an ideal random source. For that he defines a test statistic on a bit string such that the values it takes for uniform random strings and for the keystream are sequence significantly different. He then computes the value of the test statistic for the given keystream sequence and according to the value of this statistic determines if the sequence was produced by the given stream cipher or not. Sometimes distinguishing attacks can be converted to key recovery attacks [94].

Attack Paradigms: We will now list some of the popular attack techniques employed against stream ciphers.

Time-Memory-Data Tradeoff Attacks: First introduced by Hellman in [75], Time-Memory-Data (TMD) Tradeoff attacks are a generic method to perform state recovery of stream ciphers. The attacker precomputes a set of state vectors σ and a segment of keystream bits produced by this each of the states and stores them in an easily accessible table. In the online stage of the attack, the attacker examines several keystream segments and checks if the corresponding keystream segment is present in the table.

Correlation Attacks: In this attack paradigm, the attacker tries to establish some probabilistic relation between the Secret Key bits or bits of the state vector with the keystream. For example, if the attacker ascertains that $\mathsf{Pr}\ [K[0] = k_0] = \frac{1}{2} + \frac{1}{4} = \frac{3}{4}$, i.e. the first bit of the Secret Key equals the first keystream bit with high probability (the value $\frac{1}{4}$ is often referred to as the **bias** of $K[0]$ towards k_0). In such an event, the attacker simply collects keystream segments by running the

stream cipher with the same Secret Key and sufficient number of IVs. If he runs the experiment N number of times, the value of the first keystream bit will equal the value of the first Secret Key bit in about $\frac{3N}{4}$ of these cases, thus revealing the actual value of $K[0]$. Correlation attacks were first introduced in [123] and have since led to seminal research in the field of Correlation Immune Boolean Functions.

Chosen IV Attacks: The model used in Chosen IV attacks is as follows. The adversary is given access to an Oracle which is in possession of an unknown quantity (typically the Secret Key). The adversary can choose a public parameter of his choice (typically the IV) and ask the Oracle to encrypt a message of his choice. In the context of stream ciphers, this implies that the adversary is able to obtain keystream bits by querying the Oracle possessing the Secret Key with any IV of his choice. The above process can be repeated with different IVs of the adversary's choice. The task of the adversary could be either (i) to compute the Secret Key efficiently or, (ii) to distinguish the keystream output from a random stream. The first model has been successfully employed in **cube attacks** on stream ciphers [55–57]. A cube attack is a particular instance of a chosen IV attack in which the attacker queries the Oracle for keystream segments produced due to all possible IVs over a subset of IV bits. The second model has been used in distinguishing attacks on reduced round variants of stream and block ciphers [58, 61, 94, 128].

Related Key Attacks: This attack model relaxes the requirements of the chosen IV attack slightly. It is assumed that the adversary can somehow obtain keystream bits corresponding to the Key-IV pair $[f_i(K), IV_{i,j}]$, $i, j = 0, 1, 2, \ldots$, where $f_i : \mathcal{K} \rightarrow \mathcal{K}$ is a function from the Key-space $\mathcal{K}$ on to itself and K is the Secret Key. As before the adversary attempts to recover the value of K. Chosen IV related Key attacks were successfully reported against Grain v1 and Grain-128 [96].

Side Channel Attacks: In cryptography, a side channel attack is any attack based on information gained from the physical implementation of a cryptosystem, rather than brute force or theoretical weaknesses in the algorithms. For example, timing information, power consumption, electromagnetic leaks or even sound can provide an extra source of information which can be exploited to break the system. Some side-channel attacks require technical knowledge of the internal operation of the system on which the cryptography is implemented, although others such as differential power analysis are effective without such information. Typical side channel attacks include:

- **Timing Attacks**: A timing attack watches data movement into and out of the CPU, or memory, on the hardware running the cryptosystem or algorithm. Simply by observing variations in how long it takes to perform cryptographic

operations, it might be possible to determine the entire Secret Key. The reader may refer to the Cache timing attacks on AES proposed by Bernstein in [32].

- **Power Analysis Attacks**: A power analysis attack can provide even more detailed information by observing the power consumption of a hardware device such as CPU or cryptographic circuit. These attacks are roughly categorized into simple power analysis (SPA) and differential power analysis (DPA). For an introduction to Power Analysis Attacks, the reader is referred to the seminal paper by Kocher [95].

- **Fault Attacks**: The principle in these type of attacks is to induce faults or other unexpected environmental conditions into cryptographic implementations, to reveal their internal states. For example, a smartcard containing an embedded processor might be subjected to high temperature, unsupported supply voltage or current, excessively high overclocking, strong electric or magnetic fields, or even ionizing radiation to influence the operation of the processor. The processor may begin to output incorrect results due to physical data corruption, which may help a cryptanalyst deduce the instructions that the processor is running, or what its internal data state is.

  Typically, in the context of stream ciphers, optical faults using camera flashes/ laser-guns [124, 125] are induced in the registers storing the internal state vector, as a result of which the logic values stored by one or more flip-flops are toggled. As a result, the device implementing the stream cipher, starts behaving abnormally and produces keystream which it would otherwise not produce in a normal mode of operation. This is often referred to as faulty keystream sequence and may be used by the attacker to gain some non-trivial information about the Secret Key.

- **Scan Chain Attacks**: Scan-chains are one of the most commonly-used DFT (Design for Testability) techniques, which are used to check whether a chip is functioning properly or not. It provides the designer an easy way to ascertain whether the device has any structural defects or not. In this design methodology, all the flip-flops in the design are replaced with scan type flip-flops that contains a multiplexer to select either a normal mode functioning or a scan mode functioning. The design is made controllable and observable by chaining all these flip-flops together and shifting test data in and out. By suitably altering the control value to the multiplexer, the chip can be used for normal or scan test mode of operation. After selecting scan-test mode, the user is able to input test patterns of his choice into the device and thereafter scan out the contents of all the flip-flops connected to the scan-chain.

If a cryptosystem is implemented using such Scan Chains, then by observing the data shifted out of the chain, the attacker is often able to determine the internal state of the cipher.

Other Weaknesses: Apart from the aforementioned attack paradigms, an attacker may look for other weaknesses in the stream cipher design. Some of them are as follows:

- **Short Cycles**: As already stated, a Stream Cipher consists of a state vector σ that is updated by a function F. The output keystream bit/byte is a computed by function G of the internal state vector. Since the state σ consists of a finite number of bits, it stands to reason that the set of all possible values it can take on is also finite, and therefore any internal state vector must after a finite number of state updates return to its initial state. When this happens the keystream bits/bytes also start to repeat. Usually, for all well designed ciphers such cycles have immensely large periods which can not be calculated on a normal PC. Sometimes, however, stream ciphers have been known to produce cycles of small period. An example is the famous Finney Cycle [60] reported in relation to the stream cipher RC4 [4]. Short cycles are an undesirable property of stream ciphers, because if the attacker is sure that the cipher is in a state that cycles after a short time, he will be able to predict the forward generation of keystream bits. Also, Finney cycles were successfully used in [38] to mount a fault attack against RC4.
- **Slid Key-IV pairs**: It is often observed that in a stream cipher, there exist distinct Key-IV pairs (K_1, IV_1) and (K_2, IV_2) that produce keystream sequences that are finite shifts of one another. Although such Key-IV pairs do not always lead to a cryptanalytic attack, any user who has once used the Key-IV pair (K_1, IV_1) for encryption might not want to use the pair (K_2, IV_2) for subsequent encryption. There also have been instances where such slid pairs have been exploited to mount Related Key attacks on a stream cipher. For example, in [96], slid pairs were used to perform a Related Key attack against the stream ciphers Grain v1 and Grain-128.

1.4.4 The eStream Project

eSTREAM is the name given to a project to design new stream ciphers suitable for widespread adoption, organized by the EU ECRYPT network. It was set up as a result of the fact that all six stream ciphers submitted to the NESSIE project had been cryptanalyzed. The call for primitives was first issued in November 2004. The project was

completed in April 2008. The project was divided into separate phases and the project goal was to find algorithms suitable for different application profiles.

The eSTREAM portfolio ciphers fall into two profiles. Profile 1 stream ciphers are particularly suitable for hardware applications with restricted resources such as limited storage, gate count, or power consumption. Profile 2 contains stream ciphers more suitable for software applications with high throughput requirements. The portfolio [47] currently contains the following ciphers:

Profie 1 (HW)	Profile 2 (SW)
Grain v1 [73]	Salsa20 [33]
MICKEY 2.0 [16]	Sosemanuk [30]
Trivium [43]	HC128 [133]
	Rabbit [40]

TABLE 1.1: The eStream Portfolio

1.5 Motivation of this Thesis

The main motivation of this thesis is to study a few selected stream ciphers, both from the eStream portfolio and elsewhere, and analyze how the cipher designs hold up against certain typical cryptanalytic attack models. RC4 [4] is the most widely used software stream cipher and is used in popular protocols such as Transport Layer Security (TLS) (to protect Internet traffic) and WEP (to secure wireless networks). While remarkable for its simplicity and speed in software, certain weaknesses [102, 105] has been reported against the cipher. For over a decade, there has been seminal research to look for design paradigms that would somehow lead to a cipher with comparable ease of implementation and speed in software as RC4 and at the same time be free of its known weaknesses. As a result, the last decade has seen numerous design proposals like [111, 114, 138] etc. In this thesis, we take a closer look at two of such stream ciphers i) RC4+ designed by Maitra et al. [100] and ii) GGHN designed by Gong et al. [68], that were originally proposed as alternatives to RC4. We have found certain weaknesses in these designs which we present in this thesis. This leads us to conclude that eliminating the weaknesses of RC4 and at the same time preserving its simplicity is a highly complicated search and would require more painstaking research.

Thereafter we review two families the Hardware ciphers currently in the final portfolio of eStream.

- The Grain Family of Stream ciphers, i.e. Grain v1 [73], Grain-128 [74] and Grain-128a [13] designed by Ågren, Hell, Johansson, Maximov and Meier.
- MICKEY 2.0 [16] designed by Babbage and Dodd.

We investigate several avenues of cryptanalysis against these ciphers. For the Grain family, we devise methods to compute Related Key-IV pairs that produce slid keystream sequences. We also investigate the possibility of mounting a Differential Fault Attack against both the Grain family and MICKEY 2.0, under various adversarial situations. We also investigate the possibility of mounting a Scan based side channel attack on the stream cipher MICKEY 2.0. We found that an existing strategy of Scan based attack against the stream cipher Trivium [11], was ineffective against MICKEY 2.0, and hence he have provided an alternative strategy to deal with this cipher. We also make some broad observations regarding the countermeasures taken to secure Scan Chains against a class of cryptanalytic attacks.

Performing such analysis against these ciphers is important as no formal security proofs are available for them. In fact their security stems from the amount of confidence the professionals in the industry ans academia have in the strengths of these ciphers.

1.6 Organization of this Thesis

The thesis is organized in the following way:

1. Chapters 1, 2 provide a brief introduction to cryptology and outline the mathematical preliminaries necessary to read the thesis.
2. Chapter 3 is concerned with the analysis of the stream ciphers RC4+ and GGHN.
3. Chapters 4, 5, 6 is involved with the analysis of the Grain family of Stream Ciphers.
4. Chapters 7, 8 is deals with the analysis of the cipher MICKEY 2.0 and introduction to Scan Chain based attacks.

It is strongly recommended that the reader reads Chapters 1, 2 of the thesis before proceeding with the rest of the thesis. Also, it is recommended that the reader reads Chapter 4 before Chapters 5,6 and Chapter 7 before Chapter 8. For the benefit of the reader, a short summary of the Chapters in this thesis is presented below:

Chapter 1 This chapter provides a brief introduction to cryptology and the various broad classifications of cryptographic protocols. It also provides a brief introduction to Stream ciphers, the notion of Perfect Secrecy and discusses the various attack paradigms associated with Stream ciphers. Finally it introduces the reader to the eStream project which was specifically created to promote innovative design methodologies in Stream Ciphers and whose final portfolio may now be considered to consist of the state-of-the-art stream cipher designs.

Chapter 2 This chapter provides all the mathematical background necessary to read the thesis.

Chapter 3 This chapter is divided into two halves. In the first half, the stream cipher RC4+ is analyzed. It is shown that much like RC4, a distinguishing attack requiring around 2^{27} keystream bytes can be mounted on RC4+. Also, a differential fault attack on RC4+ requiring 2^{16} faults is presented. The attack presents a step by step analysis of how the internal state of the cipher may be recovered from the knowledge of the faulty keystream bytes. In the second half of the chapter, two cryptanalytic results are presented against the GGHN stream cipher. First, it is shown that numerous short cycles occur during the keystream generation phase of the cipher. Secondly, it is shown that a randomized variant of this cipher is expected to reach the all zero state in just around 2^{147} iterations, after which the cipher only produces the zero keystream byte at every iteration. Although this observation does not lead to any cryptanalytic attack on the cipher, it gives the user a limit of the amount of keystream bytes that may be safely used for encryption with such cipher designs. This chapter is based on the publications [21, 27].

Chapter 4 This chapter introduces the reader to the Grain family of stream ciphers. It provides a complete mathematical description of the ciphers in the family and enumerates all known cryptanalytic attempts reported against this family. Thereafter, we outline methods to compute Key-IV pairs in the Grain family that can generate key-streams which are either

- Almost similar in the initial segment, or
- Exact shifts of each other throughout the generation of the stream.

This chapter is based on the publications [25, 26].

Chapter 5 This chapter is involved with the Different Fault Analysis (DFA) of the Grain Family. It describes a set of three attacks on the Grain family, each of which is mounted under different setups in which the attacker is granted varying degrees of freedom.

- The first attack assumes that the attacker can exercise maximum control over the fault injection. First it is assumed that he can synchronize the timing of fault injection with a given stage of the cipher operation. Second, it is assumed that the fault that he injects, causes a change in the logical value of precisely one of the flip-flops of the registers storing the internal state (i.e a single bit-flip). Although he cannot choose the register location to be faulted, a flip-flop once faulted can be faulted multiple times.
- The second attack obviates the requirement of multiple faults on the same register location, but assumes that the attacker can still inject single bit-flipping time synchronized faults.
- The third attack requires the attacker to exercise minimal control over fault injections, i.e., the attack will be carried out under the assumption that the optical fault is not time-synchronized, nor is there any guarantee that it causes a single bit flip at a random register location. Instead, the attacker is certain that the fault he injects toggles the logic value at a maximum of three contiguous flip-flops. This attack enlists the use of SAT solvers to reduce the number of faults required to complete the attack.

This chapter is based on the publications [22–24, 120].

Chapter 6 This chapter begins with a brief description of Knellwolf's Conditional Differential Cryptanalysis [94] of reduced round Grain v1. The attack found the values of five expressions in the Secret Key bits of a variant of Grain v1 that employs only 97 out of the 160 rounds in its Key Scheduling. The values of these Secret Key expressions were deduced by observing certain non-randomness/bias in the keystream bits generated by the chosen IVs. The non-randomness were observed purely experimentally and no theoretical justification was provided for the same. In this chapter a theoretical explanation is provided of the correctness of Knellwolf's attck. First, a tool is constructed to track the the differential trails introduced in the cipher via the IV during the Key Scheduling part of the cipher operation. Using the results obtained from this tool the non-randomness in the keystream bits for the chosen IVs is proven. This chapter is based on the work presented in [18].

Chapter 7 This chapter concentrates on the stream cipher MICKEY 2.0. It starts with a mathematical description of the specifications of the cipher, and goes on to describe a Differential Fault Attack under the assumption that the attacker can inject time-synchronized, single bit-flipping optical faults. Thereafter the attack is carried out if the injected fault toggles a maximum of three neighboring flip-flops. In the second part of this chapter, SAT solvers are used to reduce the fault

requirement, both in the single bit-flip and multiple bit-flip models. This chapter is based on the work presented in [20, 28].

Chapter 8 A Scan-Chain is a popular DFT (Design for Testability) technique, which is used to check whether a chip is functioning normally or not. It provides the designer an easy way to ascertain whether the device has any structural defects or not. This chapter introduces the reader to Scan-Chain based hardware design and the related vulnerabilities that may creep into a cryptosystem implemented with Scan Chains. In [11], A Scan based attack on the stream cipher Trivium [43] was presented. The chapter outlines why the same attack can not be extended to MICKEY 2.0 [16], and suggests an alternative strategy to attack MICKEY 2.0 via Scan-Chains. Further, in [11], an XOR gate based countermeasure was suggested to protect Scan-Chains from cryptanalytic attacks. The chapter also shows that this countermeasure may fail to protect the underlying cryptosystem under certain classes of cryptanalytic attacks. It goes on to suggest a novel Double Feedback XOR-CHAIN countermeasure that is shown to be secure against the given class of cryptanalytic attacks. It is also shown than that such a Double Feedback XOR-CHAIN structure, like an ordinary Scan-Chain, may also be used for DFT purposes. This chapter is based on the work presented in [19].

Chapter 9 This chapter concludes the thesis. Here we present a comprehensive summary of our work that has been discussed throughout the thesis. We also discuss open problems which might be interesting for further investigation along this line of research.

We frequently use basic results of Boolean Functions, linear algebra and probability theory in this thesis, and expect the reader to possess a good grasp on these topics. Although we present a comprehensive overview of all necessary mathematical preliminaries in Chapter 2, a graduate level training in mathematics is recommended to read the material comfortably.

Chapter 2

Background and Preliminaries

This chapter provides the reader with a comprehensive overview of the mathematical framework that may be needed to read this thesis. The results in this chapter are mostly basic. Still we provide a few proofs for better understanding of the reader. Note that, GF(2) denotes the finite field of the two elements $\{0, 1\}$. We will use both notations interchangeably in this thesis.

2.1 Boolean Functions

An n-variable Boolean Function f is a map from $\{0,1\}^n \to \{0,1\}$. The support $\mathsf{Sup}(f)$ of the function f is defined as the set

$$\mathsf{Sup}(f) = \{x \in \{0,1\}^n \ : \ f(x) = 1\}.$$

The weight of f is denoted by $\mathsf{wt}(f)$ and is defined to be the cardinality of $\mathsf{Sup}(f)$. An n-variable Boolean Function f is said to be **balanced** if $\mathsf{wt}(f)$ is equal to 2^{n-1}. It is said to be **unbalanced** otherwise. The distance between two n-variable Boolean functions f and g is denoted by $d(f, g)$ and is defined as the cardinality of the following set D:

$$D = \{x \in \{0,1\}^n \ : \ f(x) \oplus g(x) = 1\}.$$

It is easy to see that $d(f, g) =$ Cardinality of $D = \mathsf{wt}(f \oplus g)$.

2.1.1 Representation of Boolean Functions

There are two standard ways of representing a Boolean Function:

- Truth Table representation,
- Algebraic Normal Form representation.

Truth Table A truth table is a tabulation of all possible combinations of input values and their corresponding outputs. Let $\sigma(0), \sigma(1), \ldots, \sigma(2^n - 1)$ be respectively the n-bit binary representations of the integers $0, 1, \ldots, 2^n - 1$. Then the truth table of the function f is given by the vector T_f:

$$T_f = [f(\sigma(0)), f(\sigma(1)), \ldots, f(\sigma(2^n - 1))]$$

Thus, the truth table representation of f requires 2^n bits.

Algebraic Normal Form (ANF) Apart from the truth table, another important way to represent a Boolean function is by its Algebraic Normal Form (ANF). A n-variable Boolean function f can expressed as a a multivariate polynomial $f(x_1, \ldots, x_n)$ over GF(2). This polynomial can be expressed as a sum of products representation of all distinct k-th order products ($0 \leq k \leq n$) of the variables. More precisely, $f(x_1, \ldots, x_n)$ can be written as

$$\omega_0 \oplus \bigoplus_{1 \leq i \leq n} \omega_i x_i \oplus \bigoplus_{1 \leq i < j \leq n} \omega_{ij} x_i x_j \oplus \bigoplus_{1 \leq i < j < k \leq n} \omega_{ijk} x_i x_j x_k \cdots \bigoplus \omega_{12\ldots n} x_1 x_2 \ldots x_q,$$

where the coefficients $\omega_i, \omega_{ij}, \ldots, \omega_{12\ldots n} \in \{0, 1\}$. This is the ANF representation of f. The number of variables in the highest order product term with nonzero coefficient is called the *algebraic degree*, or simply the degree of f and denoted by $\mathsf{deg}(f)$. Functions of algebraic degree at most one are called **Affine Functions**. Affine Functions which have $\omega_0 = 0$ are called **Linear Functions**.

The set of all n-variable affine (respectively linear) functions is denoted by $A(n)$ (respectively $L(n)$). Therefore, for a fixed $\omega = (\omega_1, \omega_2 \ldots, \omega_n) \in \{0, 1\}^n$, an affine function is of the form $\omega \cdot x \oplus \omega_0$, where $\omega \cdot x = \omega_1 x_1 \oplus \omega_2 x_2 \oplus \cdots \oplus \omega_n x_n$ is the inner product over GF(2).

The nonlinearity of an n-variable function Boolean Function f, denoted by $\mathsf{nl}(f)$ is defined as

$$\mathsf{nl}(f) = \min_{g \in A(n)} (d(f, g)),$$

i.e., the minimum distance from the set of all n-variable Affine Functions.

2.1.2 Walsh Spectrum

For the point $\omega = (\omega_1, \omega_2 \ldots, \omega_n) \in \{0,1\}^n$, let us define the n-variable Linear function $L_\omega = \omega \cdot x = \omega_1 x_1 \oplus \omega_2 x_2 \oplus \cdots \oplus \omega_n x_n$. The Walsh transform of the Boolean Function $f(x)$ on the point ω is defined as

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

From, the definition of $W_f(\omega)$ it is clear that whenever $L_\omega(x) \oplus f(x) = 0$, the sum is increased by 1, and when $L_\omega(x) \oplus f(x) = 1$, the sum is decreased by 1. So, we have

$$\begin{aligned} W_f(\omega) &= (2^n - \mathsf{wt}(L_\omega \oplus f)) \cdot 1 + \mathsf{wt}(L_\omega \oplus f) \cdot (-1) \\ &= 2^n - 2\ \mathsf{wt}(L_\omega \oplus f) \\ &= 2^n - 2\ d(L_\omega, f). \end{aligned}$$

The vector $[W_f(\sigma(0)), W_f(\sigma(1)), \ldots, W_f(\sigma(2^n - 1))]$ is called the Walsh Spectrum of f. If, among all the functions in $A(n)$, the minimum distance of f occurs for the linear function $L_{\omega'}$, for some $\omega' \in \{0,1\}^n$, then the above equation implies that the value of $W_f(\omega')$ is the maximum among all the other elements in the Walsh spectrum of f. On the other hand, if the minimum distance occurs for the affine function $1 \oplus L_{\omega'}$, then $-W_f(\omega')$ will have the maximum value. Therefore, in terms of the Walsh spectrum, the nonlinearity of f is given by

$$\mathsf{nl}(f) = 2^{n-1} - \frac{1}{2} \cdot \max_{\omega \in \{0,1\}^n} |W_f(\omega)|.$$

Parseval's Theorem One important identity related to the Walsh spectra of any n-variable Boolean function f is the Parseval's identity [54] which gives

$$\sum_{\omega \in \{0,1\}^n} W_f^2(\omega) = 2^{2n}.$$

It is clear that the maximum nonlinearity is achieved when the maximum absolute value of the Walsh spectrum is minimized. For n even, given the implications of the Parseval's Theorem, this happens when $W_f(\omega) = \pm 2^{n/2}$, for all $\omega \in \{0,1\}^n$. These functions, having nonlinearity $2^{n-1} - 2^{n/2-1}$, are known as **Bent Functions** [118]. For n odd, no concrete results are available. For $n \leq 7$, it is known that the maximum possible nonlinearity can be $2^{n-1} - 2^{\frac{n-1}{2}}$ [110]. It has been shown in [86, 113] that one can achieve nonlinearity strictly greater than $2^{n-1} - 2^{\frac{n-1}{2}}$ for $n \geq 9$.

Correlation Immunity An n-variable Boolean function f is called m-CI (Correlation Immune of the m^{th} order) if and only if its Walsh Spectrum satisfies

$$W_f(\omega) = 0, \quad \forall \quad 1 \leq \| \omega \| \leq m,$$

where $\| \cdot \|$ denotes the number of non-zero elements in a vector. An m-CI function f is called m-resilient if it is balanced as well. For an n-variable m-CI function f, $\mathsf{deg}(f)$ is bounded by the relation $\mathsf{deg}(f) \leq n - m$. If f is m-resilient, $\mathsf{deg}(f)$ is bounded by $\mathsf{deg}(f) \leq n - m - 1$ (see [123] for details).

2.2 Recurrences and Feedback Shift Registers

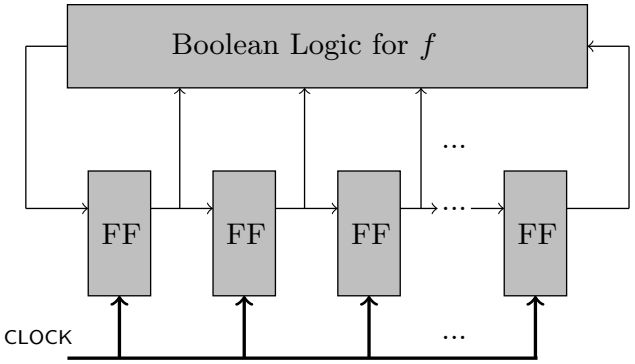
A sequence of elements $b_0, b_1, b_2, \ldots \in GF(2)$ is said to satisfy a recurrence relation of order n if b_{n+k} can be expressed as a Boolean Function of the previous n elements of the sequence, i.e.,

$$b_{n+k} = f(b_{n+k-1}, b_{n+k-2}, \ldots, b_k)$$

The recurrence is said to be Linear if the function f is a linear Boolean Function, and is said to be Non-Linear if f is nonlinear.

The standard way of constructing hardware implementations of such sequences is by employing Feedback Shift Registers (FSR) (See Fig 2.1). The design consists of a cascade of n number of D Flip-flops connected serially. The output of each flip-flop is connected to a set of gates implementing the Boolean logic for the feedback function f, the output of which is connected to the input port of the first flip-flop, thus completing the feedback circuit. The flip-flops are initialized with the values $b_0, b_1, \ldots, b_{n-1}$. At each clock edge the values in the flip-flops are shifted towards the left by 1 flip-flop, and an updated value emanating from the circuit for the Boolean logic for f, is stored in the first flip-flop. Thus, it is easy to see that after the k^{th} clock edge, the flip-flops carry the values $[b_k, b_{k+1}, \ldots, b_{n+k-1}]$. This vector is often referred to as the state of the register at time k. FSR's implementing linear recurrences are called **Linear Feedback Shift Registers (LFSR)**, and those implementing nonlinear recurrences are called **Nonlinear Feedback Shift Registers (NFSR)**.

Period of an FSR The period of the sequence $b_0, b_1, b_2, \ldots$ generated by an FSR is defined as the minimum positive integer T such that $b_{k+T} = b_k$, for all positive integers k. Since FSRs are finite state machines with the maximum number of states being 2^n, if the FSR is implemented with n flip-flops, it stands to reason that the period of any sequence generated by an FSR is less than or equal to 2^n.



$t=0$	b_{n-1}	b_{n-2}	b_{n-3}	$\cdots$	b_0	Output
$t=1$	b_n	b_{n-1}	b_{n-2}	$\cdots$	b_1	b_0
$t=2$	b_{n+1}	b_n	b_{n-1}	$\cdots$	b_2	b_1
$t=3$	b_{n+2}	b_{n+1}	b_n	$\cdots$	b_3	b_2
$t=4$	b_{n+3}	b_{n+2}	b_{n+1}	$\cdots$	b_4	b_3

FIGURE 2.1: A Feedback Shift Register

Reversibility An FSR is said to be reversible if the knowledge of any state vector $[b_k, b_{k+1}, \ldots, b_{n+k-1}]$ can lead to the unique determination of the previous state vector $[b_{k-1}, b_k, \ldots, b_{n+k-2}]$, for all $k > 1$. It was shown in [63], that an FSR is reversible if its update function $f(b_{n+k-1}, b_{n+k-2}, \ldots, b_k)$ can be written in the following manner

$$f(b_{n+k-1}, b_{n+k-2}, \ldots, b_k) = b_k \oplus f'(b_{n+k-1}, b_{n+k-2}, \ldots, b_{k+1}).$$

This can easily be verified since it can be seen that given $[b_k, b_{k+1}, \ldots, b_{n+k-1}]$, we only need to determine b_{k-1} uniquely to prove that the FSR is reversible. Since we have

$$\begin{aligned} b_{n+k-1} &= f(b_{n+k-2}, b_{n+k-3}, \ldots, b_{k-1}) \\ &= b_{k-1} \oplus f'(b_{n+k-2}, b_{n+k-3}, \ldots, b_k), \\ \text{i.e., } b_{k-1} &= b_{n+k-1} \oplus f'(b_{n+k-2}, b_{n+k-3}, \ldots, b_k). \end{aligned}$$

Thus, b_{k-1} is uniquely determined.

2.2.1 Primitive Polynomials and Maximum length LFSRs

Consider the linear recurrence given by the following update rule:

$$b_{n+k} = c_{n-1}b_{n+k-1} \oplus c_{n-2}b_{n+k-2} \oplus \cdots \oplus c_1 b_{k+1} \oplus c_0 b_k.$$

Define the feedback polynomial $p(x)$ for this recurrence as:

$$p(x) = x^n + c_{n-1} \cdot x^{n-1} + c_{n-2} \cdot x^{n-2} + \cdots + c_1 \cdot x + c_0$$

Notice that, if the LFSR is initialized with the all zero vector, i.e., $b_0 = b_1 = \cdots = b_{n-1} = 0$, then according to its update rule, it will forever remain in that state. However, an LFSR generates a sequence of period $2^n - 1$, if its feedback polynomial is a **primitive polynomial**. Primitive Polynomials are important mathematical objects used extensively in Coding theory, Finite Fields and Algebraic Geometry. A detailed discussion of such polynomials is beyond the scope of this thesis (please refer to [98] for more details), but we shall give an equivalent definition for the purpose of understanding the material covered in the thesis.

Definition 2.1. Consider the polynomial $p(x)$ with degree n and coefficients in $GF(2)$. Define the set of polynomials

$$S_m = \{a_0 + a_1 \cdot x + \cdots + a_{n-1} \cdot x^{n-1} \ : \ a_i \in GF(2)\} - \{0\}$$
$$S_p = \{x^i \bmod p(x) \ : \ 0 \leq i \leq 2^n - 2\}$$

Note that S_m is the set of all polynomials over $GF(2)$ of degree $n-1$ except the zero polynomial. S_p is the set of polynomials obtained by taking the remainder when x^i (for $i \in [0, 2^n - 2]$) is divided by $p(x)$. The polynomial $p(x)$ is said to be primitive if $S_p = S_m$. An example of a primitive polynomial over GF(2) is $x^4 + x + 1$.

An LFSR employing a primitive polynomial as its feedback polynomial is called a maximum length LFSR. As previously stated, a maximum length LFSR produces a sequence with period $2^n - 1$, also popularly called an m-sequence. Such sequences have extremely useful statistical properties and are used extensively Digital Broadcasting and Spread Spectrum Communications. However these sequences are cryptographically weak, and given $2n$ bits of such a sequence it is possible to employ the Berlekamp-Massey algorithm [103] to recover not only the initial state but also the feedback polynomial. Hence, LFSRs must be employed with some kind of nonlinearity to construct secure cryptographic primitives.

2.2.2 Nonlinear Feedback Shift Registers (NFSR)

As stated previously, an FSR with a nonlinear feedback function f is said to be a Nonlinear Feedback Shift Register (NFSR). NFSRs combined with LFSRs are the principal design components of many stream ciphers. For example, two of the three ciphers in the Hardware portfolio of eStream, i.e., Grain v1 [73] and MICKEY 2.0 [16] are constructed

in this manner, whereas the the third cipher in the Hardware portfolio, i.e., Trivium [43] is composed of 3 NFSRs.

The theory regarding NFSRs is not as developed as that of LFSRs. However, there has been extensive characterization of NFSRs that produce sequences of period 2^n also known as **Debruijn Sequences**. For more details on this topic one may refer to [46, 63].

2.3 Elementary Discrete Probability Theory

The world around us is full of phenomena we perceive as random or unpredictable. For example, when we toss a coin we can either expect to land a Head or a Tail, when we roll a die we can expect one of the possible elements from the set $\{1, 2, 3, 4, 5, 6\}$. We aim to model these phenomena as outcomes of some experiment. The set of all possible outcomes of an experiment are elements of a **sample space** Ω, and subsets of Ω are called events. The events are assigned a **probability**, a real number between 0 and 1 that expresses how likely the event is to occur.

We represent the outcome of the experiment by a capital Roman letter, such as X, called a **random variable**. In this thesis, we will consider the case where the experiment has only finitely many possible outcomes, i.e., the sample space is finite. Since the sample space is finite, the random variable is said to be discrete.

2.3.1 Probability Distribution Function

We next describe the assignment of probabilities to each event of the sample space. In this context, we will define the concept of a probability distribution function.

Definition 2.2. Let X be a random variable which denotes the value of the outcome of a certain experiment, and assume that this experiment has only finitely many possible outcomes. Let Ω be the sample space of the experiment (i.e., the set of all possible values of X, or equivalently, the set of all possible outcomes of the experiment.) A distribution function for X is a real-valued function $\mathsf{Pr}(\cdot) : \Omega \to \mathbb{R}(0, 1)$ where $\mathbb{R}(0, 1)$ denotes the set of reals between and including 0 and 1, and which satisfies:

1. $\mathsf{Pr}(\omega) \geq 0$, for all $\omega \in \Omega$, and
2. $\sum_{\omega \in \Omega} \mathsf{Pr}(\omega) = 1$.

For any subset E of Ω, we define the probability of E to be the number $\mathsf{Pr}(E)$ given by

$$\mathsf{Pr}(E) = \sum_{\omega \in E} \mathsf{Pr}(\omega).$$

Note that if $E = \emptyset$, we assign $\mathsf{Pr}(E) = 0$.

Example 2.1. *A die is rolled once. We let X denote the outcome of this experiment. Then the sample space for this experiment is the 6-element set*

$$\Omega = \{1, 2, 3, 4, 5, 6\},$$

where each outcome i, for $i = 1, \ldots, 6$, corresponds to the number of dots on the face which turns up. The event

$$E = \{2, 4, 6\}$$

corresponds to the statement that the result of the roll is an even number. The event E can also be described by saying that X is even. Unless there is reason to believe the die is loaded, the natural assumption is that every outcome is equally likely. Adopting this convention means that we assign a probability of $\frac{1}{6}$ to each of the six outcomes. Mathematically we express this as $\mathsf{Pr}(i) = \frac{1}{6}$, for $1 \leq i \leq 6$. In this example, therefore $\mathsf{Pr}(E) = \mathsf{Pr}(2) + \mathsf{Pr}(4) + \mathsf{Pr}(6) = \frac{1}{2}$, as the events are mutually exclusive.

In many cases, events can be described in terms of other events through the use of the standard constructions of set theory. We will briefly look a few identities regarding probabilities when defined on sets.

Theorem 2.3. *Let X be a random variable which denotes the value of the outcome of a certain experiment, and assume that this experiment has only finitely many possible outcomes. Let Ω be the sample space of the experiment. then the following properties hold regarding the probability distribution function $\mathsf{Pr}(\cdot)$ hold:*

1. $\mathsf{Pr}(E) \geq 0$ *for every* $E \subset \Omega$
2. $\mathsf{Pr}(\Omega) = 1$ *and* $\mathsf{Pr}(\emptyset) = 0$
3. *If* $E \subset F \subset \Omega$ *then* $\mathsf{Pr}(E) \leq \mathsf{Pr}(F)$
4. $\mathsf{Pr}(A \cup B) = \mathsf{Pr}(A) + \mathsf{Pr}(B) - \mathsf{Pr}(A \cap B)$.
5. *If A and B are disjoint subsets, i.e., $A \cap B = \emptyset$ then* $\mathsf{Pr}(A \cup B) = \mathsf{Pr}(A) + \mathsf{Pr}(B)$
6. $\mathsf{Pr}(A^c) = 1 - \mathsf{Pr}(A)$

Proof. Properties 1 and 2 follow from the definition of the probability distribution function. To prove property 3 note that if $E \subset F$ then every element ω that belongs to E also belongs to F. Therefore,

$$\sum_{\omega \in E} \mathsf{Pr}(\omega) \leq \sum_{\omega \in F} \mathsf{Pr}(\omega) \Rightarrow \mathsf{Pr}(E) \leq \mathsf{Pr}(F).$$

To prove property 4, we can divide the set $A \cup B$ in three parts. The 1st part consists of elements belonging only to A and not B (therefore belonging to $A - A \cap B$), the second part consists of elements belonging only to B and not A (therefore belonging to $B - A \cap B$) , and the third part consists of elements common to both A and B, i.e., belonging to $A \cap B$. So we have,

$$\begin{aligned}\mathsf{Pr}(A \cup B) = \sum_{\omega \in A \cup B} \mathsf{Pr}(\omega) &= \sum_{\omega \in A - A \cap B} \mathsf{Pr}(\omega) + \sum_{\omega \in B - A \cap B} \mathsf{Pr}(\omega) + \sum_{\omega \in A \cap B} \mathsf{Pr}(\omega) \\ &= \mathsf{Pr}(A) - \mathsf{Pr}(A \cap B) + \mathsf{Pr}(B) - \mathsf{Pr}(A \cap B) + \mathsf{Pr}(A \cap B) \\ &= \mathsf{Pr}(A) + \mathsf{Pr}(B) - \mathsf{Pr}(A \cap B)\end{aligned}$$

Property 5 follows from property 4. To prove Property 6 we note that $\Omega = A \cup A^c$. Since A and A^c are disjoint sets we use Properties 2, 5 to write

$$1 = \mathsf{Pr}(\Omega) = \mathsf{Pr}(A \cup A^c) = \mathsf{Pr}(A) + \mathsf{Pr}(A^c) \Rightarrow \mathsf{Pr}(A^c) = 1 - \mathsf{Pr}(A).$$

□

2.3.2 Conditional Probability

In this section we ask and answer the following question. Suppose we assign a distribution function to a sample space and then learn that an event E has occurred. How should we change the probabilities of the remaining events? We shall call the new probability for an event F the conditional probability of F given E and denote it by $\mathsf{Pr}(F|E)$.

Example 2.2. *An experiment consists of rolling a die once. Let X be the outcome. Let F be the event $X = 6$, and let E be the event $X > 4$. Since $X \sim U(\{1, 2, 3, 4, 5, 6\})$, we have, $\mathsf{Pr}(F) = \frac{1}{6}$. Now suppose that the die is rolled and we are told that the event E has occurred. This leaves only two possible outcomes: 5 and 6. In the absence of any other information, we would still regard these outcomes to be equally likely, so the conditional probability of F becomes $\frac{1}{2}$, making $\mathsf{Pr}(F|E) = \frac{1}{2}$.*

Let $\Omega = \{\omega_1, \omega_2, \ldots, \omega_r\}$ be the original sample space with distribution function $\mathsf{Pr}(\omega_j)$ assigned. Suppose we learn that the event E has occurred. We want to assign a new

distribution function $\mathsf{Pr}(\omega_j|E)$ to reflect this fact. Clearly, if a sample point ω_j is not in E, we want $\mathsf{Pr}(\omega_j|E) = 0$. Moreover, in the absence of information to the contrary, it is reasonable to assume that the probabilities for $\omega_k \in E$ should have the same relative magnitudes that they had before we learned that E had occurred. For this we require that

$$\mathsf{Pr}(\omega_k|E) = c \cdot \mathsf{Pr}(\omega_k)$$

or all $\omega_k \in E$, with c being some positive constant. We also must have

$$\mathsf{Pr}(\omega_k|E) = c \cdot \sum_{\omega_k \in E} \mathsf{Pr}(\omega_k) = 1 \Rightarrow c = \frac{1}{\sum_{\omega_k \in E} \mathsf{Pr}(\omega_k)} = \frac{1}{\mathsf{Pr}(E)}$$

(Note that this requires us to assume that $\mathsf{Pr}(E) > 0$). Thus, we will define

$$\mathsf{Pr}(\omega_k|E) = \frac{\mathsf{Pr}(\omega_k)}{\mathsf{Pr}(E)}$$

for all $\omega_k \in E$. We will call this new distribution the conditional distribution given E. For a general event F, this gives

$$\mathsf{Pr}(F|E) = \sum_{\omega_k \in F \cap E} \mathsf{Pr}(\omega_k|E) = \sum_{\omega_k \in F \cap E} \frac{\mathsf{Pr}(\omega_k)}{\mathsf{Pr}(E)} = \frac{\mathsf{Pr}(F \cap E)}{\mathsf{Pr}(E)}$$

We call $\mathsf{Pr}(F|E)$ the conditional probability of F given E.

Example 2.3. *Let us return to the example of rolling a die. Recall that F is the event $X = 6$, and E is the event $X > 4$. Note that $E \cap F$ is the event F . So, the above formula gives*

$$\mathsf{Pr}(F|E) = \frac{\mathsf{Pr}(F \cap E)}{\mathsf{Pr}(E)} = \frac{\mathsf{Pr}(F)}{\mathsf{Pr}(E)} = \frac{1/6}{2/6} = \frac{1}{2}.$$

in agreement with the calculations performed earlier.

2.3.3 Independent Events

It often happens that the knowledge that a certain event E has occurred has no effect on the probability that some other event F has occurred, that is, that $\mathsf{Pr}(F|E) = \mathsf{Pr}(F)$. If this is true, we might say that F is independent of E. This idea is formalized in the following definition of independent events.

Definition 2.4. Let E and F be two events. We say that they are independent if either

- Both events have positive probability and

$$\mathsf{Pr}(E|F) = \mathsf{Pr}(E) \quad \text{and} \quad \mathsf{Pr}(F|E) = \mathsf{Pr}(F),$$

- Or, at least one of the events has probability 0.

The following theorem provides another way to check for independence.

Theorem 2.5. *Two events E and F are independent if and only if*

$$\mathsf{Pr}(E \cap F) = \mathsf{Pr}(E)\mathsf{Pr}(F).$$

Proof. If either event has probability 0, then the two events are independent and the above equation is true, so the theorem is true in this case. Thus, we may assume that both events have positive probability. Assume that E and F are independent. Then $\mathsf{Pr}(E|F) = \mathsf{Pr}(E)$, and so

$$\mathsf{Pr}(E \cap F) = \mathsf{Pr}(E|F)\mathsf{Pr}(F) = \mathsf{Pr}(E)\mathsf{Pr}(F).$$

Assume next that $\mathsf{Pr}(E \cap F) = \mathsf{Pr}(E)\mathsf{Pr}(F)$. Then

$$\mathsf{Pr}(E|F) = \frac{\mathsf{Pr}(E \cap F)}{\mathsf{Pr}(F)} = \mathsf{Pr}(E).$$

Similarly,

$$\mathsf{Pr}(F|E) = \frac{\mathsf{Pr}(E \cap F)}{\mathsf{Pr}(E)} = \mathsf{Pr}(F).$$

Therefore, E and F are independent. □

2.3.4 Joint Distribution Functions and Independence of Random Variables

It is frequently the case that when an experiment is performed, several different quantities concerning the outcomes are investigated. If we have several random variables $X_1, X_2, \ldots, X_n$ which correspond to a given experiment or a set of different experiments, then we can consider the joint random variable $X = (X_1, X_2, \ldots, X_n)$ defined by writing, as an n-tuple, the corresponding n outcomes for the random variables $X_1, X_2, \ldots, X_n$. Thus, if the random variable X_i has, as its set of possible outcomes the set R_i, then the set of possible outcomes of the joint random variable X is the Cartesian product of the R_i's, i.e., the set of all n-tuples of possible outcomes of the X_i's. Thus the sample space is written as

$$\Omega = R_1 \times R_2 \times \cdots \times R_n$$

The function which gives the probability of each of the outcomes of X is said to be the **joint distribution function** of X.

We now look at another important definition.

Definition 2.6. The random variables $X_1, X_2, \ldots, X_n$ are said to be mutually independent if

$$\mathsf{Pr}(X_1 = r_1, X_2 = r_2, \ldots, X_n = r_n) = \mathsf{Pr}(X_1 = r_1) \cdot \mathsf{Pr}(X_2 = r_2) \cdots \mathsf{Pr}(X_n = r_n)$$

for any choice of $(r_1, r_2, \ldots, r_n) \in R_1 \times R_2 \times \cdots \times R_n$. Thus, if $X_1, X_2, \ldots, X_n$ are mutually independent, then the joint distribution function of the random variable X is just the product of the individual distribution functions.

Example 2.4. *Suppose we toss a coin three times. The basic random variable X corresponding to this experiment has eight possible outcomes, which are the ordered triples consisting of* Heads *and* Tails*. We can also define the random variable X_i , for $i = 1, 2, 3$, to be the outcome of the i^{th} toss. Since, the coin is fair, the distribution functions of X_1, X_2, X_3 are identical and defined by $\mathsf{Pr}(\mathsf{H}) = p$, $\mathsf{Pr}(\mathsf{T}) = 1-p$. Since one toss does not affect the outcome of the other, X_1, X_2, X_3 can be assumed to be mutually independent. Thus, the joint distribution for the random variable X can be written as*

$$\mathsf{Pr}[X_1 = x_1, X_2 = x_2, X_3 = x_3] = \mathsf{Pr}[X_1 = x_1] \cdot \mathsf{Pr}[X_2 = x_2] \cdot [X_3 = x_3].$$

So if $x_1 =$ Heads, $x_2 =$ Tails, $x_3 =$ Tails, we have $\mathsf{Pr}[X_1 = x_1, X_2 = x_2, X_3 = x_3] = p \cdot (1-p)^2$.

2.3.5 Bayes' Formula

Suppose we have a set of events $H_1, H_2, \ldots, H_m$ that are pairwise disjoint and such that the sample space Ω satisfies the equation

$$\Omega = H_1 \cup H_2 \cup \cdots \cup H_m$$

We call these events hypotheses. We also have an event E that gives us some information about which hypothesis is correct. We call this event evidence. We have a set of prior probabilities $\mathsf{Pr}(H_1), \mathsf{Pr}(H_2), \ldots, \mathsf{Pr}(H_m)$ for the hypotheses. If we know the correct hypothesis, we know the probability for the evidence. That is, we know $\mathsf{Pr}(E|H_i)$ for all i. We want to find the probabilities for the hypotheses given the evidence. That is, we want to find the conditional probabilities $\mathsf{Pr}(H_i|E)$. These probabilities are called the posterior probabilities.

Before we proceed further, recall that the events H_i are mutually disjoint ans $\cup_{i=1}^{m} H_i = \Omega$. This implies that the events $E \cap H_i$ are also mutually disjoint, and we have

$$E = (E \cap H_1) \cup (E \cap H_2) \cup \cdots \cup (E \cap H_m) \tag{2.1}$$

By Property 5 Theorem 2.3, this implies

$$\begin{aligned}\mathsf{Pr}(E) &= \mathsf{Pr}(E \cap H_1) + \mathsf{Pr}(E \cap H_2) + \cdots + \mathsf{Pr}(E \cap H_m) \\ &= \mathsf{Pr}(H_1)\ \mathsf{Pr}(E|H_1) + \mathsf{Pr}(H_2)\ \mathsf{Pr}(E|H_2) + \cdots + \mathsf{Pr}(H_m)\ \mathsf{Pr}(E|H_m)\end{aligned} \tag{2.2}$$

We start with the identity

$$\mathsf{Pr}(H_i|E) = \frac{\mathsf{Pr}(H_i \cap E)}{\mathsf{Pr}(E)}.$$

Since we have $\mathsf{Pr}(H_i \cap E) = \mathsf{Pr}(H_i)\ \mathsf{Pr}(E|H_i)$, we can rewrite the above equation as

$$\begin{aligned}\mathsf{Pr}(H_i|E) &= \frac{\mathsf{Pr}(H_i)\ \mathsf{Pr}(E|H_i)}{\mathsf{Pr}(E)} \\ &= \frac{\mathsf{Pr}(H_i)\ \mathsf{Pr}(E|H_i)}{\mathsf{Pr}(H_1)\ \mathsf{Pr}(E|H_1) + \mathsf{Pr}(H_2)\ \mathsf{Pr}(E|H_2) + \cdots + \mathsf{Pr}(H_m)\ \mathsf{Pr}(E|H_m)} \\ &= \frac{\mathsf{Pr}(H_i)\ \mathsf{Pr}(E|H_i)}{\sum_{i=1}^{m} \mathsf{Pr}(H_i)\ \mathsf{Pr}(E|H_i)}.\end{aligned}$$

The above identity is known as Bayes' formula.

2.3.6 Expectation of a Random variable

Consider the random variable X on the finite sample space $\Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$ having a probability distribution function $\mathsf{Pr}(\cdot)$. Then the expectation of X is denoted by the symbol $\mathbf{E}[X]$ ands defined as:

$$\mathbf{E}[X] = \sum_{i=1}^{n} \omega_i \cdot \mathsf{Pr}(\omega_i).$$

The expectation of any random variable X can be considered the average outcome of X, i.e., it is the value of X one would expect to find if one could repeat the experiment an infinite number of times and take the average of all the values obtained.

Example 2.5. *Let X represent the outcome of a roll of a six-sided die. The possible values for X are $1, 2, 3, 4, 5, 6$, all equally likely (each having the probability of $\frac{1}{6}$). The expectation of X is*

$$\mathbf{E}[X] = 1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + 3 \cdot \frac{1}{6} + 4 \cdot \frac{1}{6} + 5 \cdot \frac{1}{6} + 6 \cdot \frac{1}{6} = 3.5.$$

Note that the expectation of any function $g : \Omega \to \mathbb{R}$ of the random variable X is denoted as $\mathbf{E}[g(X)]$ and is calculated similarly

$$\mathbf{E}[g(X)] = \sum_{i=1}^{n} g(\omega_i) \cdot \mathsf{Pr}(\omega_i).$$

So in the previous example the expectation of X^2 is given as

$$\mathbf{E}[X^2] = 1 \cdot \frac{1}{6} + 4 \cdot \frac{1}{6} + 9 \cdot \frac{1}{6} + 16 \cdot \frac{1}{6} + 25 \cdot \frac{1}{6} + 36 \cdot \frac{1}{6} = 15\frac{1}{6}.$$

Linearity of Expectation The expected value operator (or expectation operator) $\mathbf{E}$ is linear in the sense that if $X,\ Y$ are any two random variables then the following hold:

$$\begin{aligned} \mathbf{E}[X + c] &= \mathbf{E}[X] + c, \\ \mathbf{E}[X + Y] &= \mathbf{E}[X] + \mathbf{E}[Y], \\ \mathbf{E}[aX] &= a\mathbf{E}[X]. \end{aligned}$$

Conditional Expectation Let X and Y be discrete random variables defined on the sample spaces $\Omega_X,\ \Omega_Y$ respectively, then the conditional expectation of X given the event $Y = y$ is a function of y over the range of Y and is defined as:

$$\mathbf{E}(X|Y = y) = \sum_{x \in \Omega_X} x\ \mathsf{Pr}(X = x|Y = y) = \sum_{x \in \Omega_X} x\ \frac{\mathsf{Pr}(X = x, Y = y)}{\mathsf{Pr}(Y = y)}.$$

2.3.7 Variance/Standard Deviation of a Random variable

Variance measures how far a set of numbers is spread out. (A variance of zero indicates that all the values are identical). A small variance indicates that the data points tend to be very close to the mean (expected value) and hence to each other, while a high variance indicates that the data points are very spread out from the mean and from each other. The square root of variance is called the standard deviation. Mathematically variance of a random variable X is defined as:

$$\begin{aligned} \mathrm{Var}(X) &= \mathbf{E}[(X - \mathbf{E}[X])^2] \\ &= \mathbf{E}\left[X^2 - 2X\mathbf{E}[X] + (\mathbf{E}[X])^2\right] \\ &= \mathbf{E}\left[X^2\right] - 2\mathbf{E}[X]\mathbf{E}[X] + (\mathbf{E}[X])^2 \\ &= \mathbf{E}\left[X^2\right] - (\mathbf{E}[X])^2. \end{aligned}$$

2.3.8 Important Probability Distribution Functions

In this part, we will look at a few probability distribution functions that we are likely to encounter later in the thesis.

Uniform Distribution The uniform distribution on a sample space Ω containing n elements $\omega_1, \omega_2, \ldots, \omega_n$ is the function Pr_u defined by

$$\mathsf{Pr}_u(\omega_i) = \frac{1}{n}, \quad \forall i \in \{1, 2 \ldots, n\}.$$

If a random variable X on the sample space Ω follows the probability distribution function Pr_u, then we say that X is uniformly distributed, and denote it by the symbol $X \sim U(\Omega)$.

Expectation/Variance: The expectation of X is given by

$$\mathbf{E}[X] = \sum_{i=1}^{n} \frac{1}{n} \cdot \omega_i = \frac{\sum_{i=1}^{n} \omega_i}{n}$$

The Variance of X is given by

$$\mathrm{Var}(X) = \mathbf{E}\left[X^2\right] - (\mathbf{E}[X])^2 = \frac{\sum_{i=1}^{n} \omega_i^2}{n} - \left(\frac{\sum_{i=1}^{n} \omega_i}{n}\right)^2$$

For example, the tossing of an unbiased coin or a rolling of a fair die invokes a uniform distribution on the sets $\{\mathsf{Head}, \mathsf{Tail}\}$ and $\{1, 2, 3, 4, 5, 6\}$ respectively. In general, whenever we assume that all outcomes of an experiment are equally likely, then the random variable X which represents the outcome of an experiment of this type, is said to be uniformly distributed.

Bernoulli Distribution The Bernoulli Distribution is limited to a sample space Ω containing the two elements $\{0, 1\}$. The probability distribution function Pr_{Ber} is given by

$$\mathsf{Pr}_{Ber}(\omega) = \begin{cases} p, & \text{if } \omega = 1, \\ 1 - p, & \text{if } \omega = 0. \end{cases}$$

If a random variable X follows the probability distribution function Pr_{Ber}, we denote it by the symbol $X \sim Ber(p)$.

Expectation/Variance: The expectation of X is given by

$$\mathbf{E}[X] = p \cdot 1 + (1 - p) \cdot 0 = p$$

The Variance of X is given by

$$\mathrm{Var}(X) = \mathbf{E}\left[X^2\right] - (\mathbf{E}[X])^2 = p(1-p)$$

The Bernoulli Distribution can be used to model the outcomes of experiments which yield only two outcomes. For example, tossing a biased coin where $\mathsf{Pr}(\mathsf{Head}) = \frac{3}{4}$. If we denote Head by the symbol 1 and Tail by the symbol 0, then the tossing of this coin invokes the random variable $X \sim Ber(\frac{3}{4})$.

Binomial Distribution The binomial distribution is the discrete probability distribution of the number of successes in a sequence of n independent yes/no experiments, each of which yields success with probability p. It can also be defined as the distribution of the random variable which counts the number of heads which occur when a coin is tossed n times, assuming that on any one toss, the probability that a Head occurs is p. The distribution function is given by the formula

$$\mathsf{Pr}_{Bin}(k) = \binom{n}{k} \cdot p^k (1-p)^{n-k}, \quad \text{for } 0 \leq k \leq n.$$

If a random variable X follows the probability distribution function Pr_{Bin}, we denote it by the symbol $X \sim Binomial(n, p)$.

Expectation/Variance: The expectation of X is given by

$$\mathbf{E}[X] = \sum_{k=0}^{n} k \cdot \binom{n}{k} \cdot p^k (1-p)^{n-k} = np.$$

The Variance of X is given by

$$\mathrm{Var}(X) = \mathbf{E}\left[X^2\right] - (\mathbf{E}[X])^2 = np(1-p)$$

Note that, if $X_1, X_2, X_3, \ldots, X_n$ are independent, identically distributed random variables, all $\sim Ber(p)$, then $Y = \sum_{k=1}^{n} X_k \sim Binomial(n, p)$.

Geometric Distribution Consider a Bernoulli process continued for an infinite number of trials; for example, tossing a biased coin (with $\mathsf{Pr}(\mathsf{Head}) = p$) an infinite sequence of times, till we get a Head. Let X denote the random variable counting the number of trials up to and including the first success. Then we can see that

$$\mathsf{Pr}_{Geo}(X = k) = (1-p)^{k-1} \cdot p.$$

Expectation/Variance: The expectation of X is given by

$$\mathbf{E}[X] = \sum_{k=1}^{\infty} k \cdot (1-p)^{k-1} \cdot p = \frac{1}{p}.$$

The Variance of X is given by

$$\mathrm{Var}(X) = \mathbf{E}\left[X^2\right] - (\mathbf{E}[X])^2 = \frac{1-p}{p^2}.$$

If a random variable X follows the probability distribution function Pr_{Geo}, we denote it by the symbol $X \sim Geo(p)$.

2.3.9 Coupon Collector's Problem

In probability theory, the coupon collector's problem describes the "collect all coupons and win" contests. It asks the following question: Suppose that there is an urn of n different coupons, from which coupons are being collected, equally likely, with replacement. How many coupons do you expect you need to draw with replacement before having drawn each coupon at least once? We will give a preliminary solution to the problem here. For a detailed analysis of this problem, one may refer to [59].

Let T be the time to collect all n coupons, and let t_i be the time to collect the i^{th} coupon given $i-1$ coupons have been collected. Think of T and t_i as random variables. Note that t_1 is always 1. Observe that the probability of collecting a new coupon given $i-1$ coupons is $p_i = \frac{(n-(i-1))}{n}$. Therefore, $t_i \sim Geo(p_i)$. Therefore the expectation of this random variable is $\mathbf{E}[t_i] = \frac{1}{p_i}$. Since $T = t_1 + t_2 + t_3 + \ldots$, by the linearity of expectations we have:

$$\begin{aligned}\mathbf{E}(T) &= \mathbf{E}(t_1) + \mathbf{E}(t_2) + \cdots + \mathbf{E}(t_n) = \frac{1}{p_1} + \frac{1}{p_2} + \cdots + \frac{1}{p_n} \\ &= \frac{n}{n} + \frac{n}{n-1} + \cdots + \frac{n}{1} = n \cdot \left(\frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n}\right) = n \cdot H_n.\end{aligned}$$

Here H_n is the n^{th} harmonic number. Using the asymptotics of the harmonic numbers, we obtain:

$$\mathbf{E}(T) \approx n \cdot H_n = n \ln n + \gamma n + \frac{1}{2}, \quad \text{as } n \to \infty,$$

where $\gamma \approx 0.5772156649$ is the Euler-Mascheroni constant.

2.4 Markov Chains

Modern probability theory studies processes for which the knowledge of previous outcomes influences predictions for future experiments. In principle, when we observe a sequence of chance experiments, all of the past outcomes could influence our predictions for the next experiment. For example, this should be the case in predicting a student's grades on a sequence of exams in a course. But to allow this much generality would make it very difficult to prove general results. In 1907, A. A. Markov began the study of an important new type of chance process. In this process, the outcome of a given experiment can affect the outcome of the next experiment. This type of process is called a Markov chain.

Definition 2.7. Markov Property A set of random variables $X_1, X_2, X_3, \ldots, X_n$ on the sample space Ω is said to satisfy the Markov property if

$$\mathsf{Pr}[X_n = x_n | X_{n-1} = x_{n-1}, \ldots, X_1 = x_1] = \mathsf{Pr}[X_n = x_n | X_{n-1} = x_{n-1}]$$

for all $x_1, x_2, \ldots, x_n \in \Omega$. Thus this random process usually characterized as **memoryless**: the next state depends only on the current state and not on the sequence of events that preceded it. The sequence of states $X_1, X_2, X_3, \ldots, X_n$ is then said to follow a Markov Chain.

We describe a Markov chain as follows: We have a set of states, $S = \{s_1, s_2, \ldots, s_r\}$. The process starts in one of these states and moves successively from one state to another. Each move is called a step. If the chain is currently in state s_i , then it moves to state s_j at the next step with a probability denoted by p_{ij}, and this probability does not depend upon which states the chain was in before the current state. The probabilities p_{ij} are called transition probabilities. Note that if at stage t the state variable is denoted by the random variable X_t, then p_{ij} is simply the conditional probability given by the formula

$$p_{ij} = \mathsf{Pr}[X_{t+1} = s_j | X_t = S_i]$$

The process can remain in the state it is in, and this occurs with probability p_{ii}. An initial probability distribution, defined on S, specifies the starting state. Usually this is done by specifying a particular state as the starting state.

2.4.1 Transition Matrix

We will explain the concept of a Transition Matrix with the help of the following example.

Example 2.6. *Consider a country in which there are three types of weather: Rainy, Nice and Snowy, denoted by the symbols R, N, S respectively. They never have two nice days in a row. If they have a nice day, they are just as likely to have snow as rain the next day. If they have snow or rain, they have an even chance of having the same the next day. If there is change from snow or rain, only half of the time is this a change to a nice day. With this information we form a Markov chain as follows. We consider the sample space as the kinds of weather, i.e., $\Omega = \{R, N, S\}$. From the above information we determine the transition probabilities. These are most conveniently represented in a matrix as*

$$\mathbf{P} = \begin{array}{c} \\ R \\ N \\ S \end{array} \begin{array}{c} \begin{array}{ccc} R & N & S \end{array} \\ \begin{pmatrix} 1/2 & 1/4 & 1/4 \\ 1/2 & 0 & 1/2 \\ 1/4 & 1/4 & 1/2 \end{pmatrix} \end{array}.$$

The entries in the first row of the matrix $\mathbf{P}$ in the above Example represent the probabilities for the various kinds of weather following a rainy day. Similarly, the entries in the second and third rows represent the probabilities for the various kinds of weather following nice and snowy days, respectively. Such a square array is called the matrix of transition probabilities, or simply the **Transition Matrix**.

We consider the question of determining the probability that, given the chain is in state i today, it will be in state j two days from now. We denote this probability by $p_{ij}^{(2)}$. Notice that if it is rainy today then the event that it is snowy two days from now is the disjoint union of the following three events:

1. It is rainy tomorrow and snowy two days from now,
2. It is nice tomorrow and snowy two days from now, and
3. It is snowy tomorrow and snowy two days from now.

Denoting $\{R, N, S\}$ by the symbols $\{1, 2, 3\}$, we have

$$\begin{aligned} p_{13}^{(2)} &= \mathsf{Pr}[X_{t+2} = 3 | X_t = 1] \\ &= \sum_{i=1}^{3} \mathsf{Pr}[X_{t+1} = i | X_t = 1] \ \mathsf{Pr}[X_{t+2} = 3 | X_{t+1} = i] \\ &= p_{11}p_{13} + p_{12}p_{23} + p_{13}p_{33}. \end{aligned}$$

This equation should remind the reader of a dot product of two vectors; we are taking the inner product of the first row of $\mathbf{P}$ with the third column of $\mathbf{P}$. Note that the inner product of two vectors $X = [x_1, x_2 \ldots, x_n]$ and $Y = [y_1, y_2, \ldots, y_n]$ is defined as $\sum_{i=1}^{n} x_i y_i$. In general, if a Markov chain has r states, then

$$p_{ij}^{(2)} = \sum_{k=1}^{r} p_{ik} p_{kj}.$$

We will now state two theorems which are easy to prove by using the above observation and induction.

Theorem 2.8. *Let $\mathbf{P}$ be the transition matrix of a Markov chain. The $(i,j)^{th}$ entry p_{ij} of the matrix $\mathbf{P}^n$ gives the probability that the Markov chain, starting in state s_i, will be in state s_j after n steps.*

Theorem 2.9. *Let $\mathbf{P}$ be the transition matrix of a Markov chain, and let $\mathbf{u}$ be the probability vector which represents the starting distribution, i.e., the distribution of the initial state variable X_1. Then the probability distribution of the state variable X_n, i.e., after n stages is given by*

$$\mathbf{u}^{(n)} = \mathbf{u} \cdot \mathbf{P}^n.$$

2.4.2 Absorbing Markov Chains

The subject of Markov chains is best studied by considering special types of Markov chains. The first type that we shall study is called an absorbing Markov chain.

Definition 2.10. A state s_i of a Markov chain is called absorbing if it is impossible to leave it (i.e., $p_{ii} = 1$). A Markov chain is absorbing if it has at least one absorbing state, and if from every state it is possible to go to an absorbing state (not necessarily in one step). In an absorbing Markov chain, a state which is not absorbing is called transient.

Canonical Form Consider an arbitrary absorbing Markov chain. Renumber the states so that the transient states come first. If there are r absorbing states and t transient states, the transition matrix will have the following canonical form

$$\mathbf{P} = \begin{pmatrix} \mathbf{Q} & \mathbf{R} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}.$$

Here $\mathbf{I}$ is the $r \times r$ identity matrix, $\mathbf{0}$ is the $r \times t$ zero matrix, $\mathbf{R}$ is a nonzero $t \times r$ matrix, and $\mathbf{Q}$ is a $t \times t$ matrix. As previously denoted, the first t states are transient and the last r states are absorbing. A standard matrix algebra argument shows that $\mathbf{P}^n$ is of the form

$$\mathbf{P}^n = \begin{pmatrix} \mathbf{Q}^n & * \\ \mathbf{0} & \mathbf{I} \end{pmatrix}.$$

The form of $\mathbf{P}^n$ shows that the entries of $\mathbf{Q}^n$ give the probabilities for being in each of the transient states after n stages for each possible transient starting state. First we prove that the probability of being in the transient states after n stages approaches zero as $n \to \infty$. Thus every entry of $\mathbf{Q}^n$ approaches zero as n approaches infinity.

Theorem 2.11. **[70, Theorem 11.3]** *In an absorbing Markov chain, the probability that the process will be absorbed is* 1 *(i.e.,* $\mathbf{Q}^n \to 0$ *as* $n \to \infty$*).*

Proof. From each transient state s_j it is possible to reach an absorbing state. Let m_j be the minimum number of steps required to reach an absorbing state, starting from s_j . Let p_j be the probability that, starting from s_j, the process will not reach an absorbing state in m_j steps. Let m be the largest of all the m_j's and let p be the largest of all p_j's. The probability of not being absorbed in m steps is less than or equal to p, in $2m$ steps less than or equal to p^2, etc. Since $p < 1$ these probabilities tend to 0. Since the probability of not being absorbed in n steps is monotone decreasing, these probabilities also tend to 0, hence $\lim_{n\to\infty} \mathbf{Q}^n = 0$. □

The Fundamental Matrix We will now define the concept of a Fundamental Matrix for an absorbing Markov Chain.

Theorem 2.12. **[70, Theorem 11.4]** *For an absorbing Markov chain the matrix* $\mathbf{I} - \mathbf{Q}$ *has an inverse* $\mathbf{N}$ *given by*

$$\mathbf{N} = \mathbf{I} + \mathbf{Q} + \mathbf{Q}^2 + \cdots .$$

The $(i, j)^{th}$ *entry* n_{ij} *of the matrix* $\mathbf{N}$ *is the expected number of times the chain is in state* s_j*, given that it starts in state* s_i*.*

Proof. Let $(\mathbf{I} - \mathbf{Q})\mathbf{x} = \mathbf{0}$, that is $\mathbf{x} = \mathbf{Q}\mathbf{x}$. Then, iterating this we see that $\mathbf{x} = \mathbf{Q}^n\mathbf{x}$. Since $\mathbf{Q}^n \to 0$, we have $\mathbf{Q}^n\mathbf{x} \to 0$, so $\mathbf{x} = \mathbf{0}$. Thus $(\mathbf{I} - \mathbf{Q})^{-1} = \mathbf{N}$ exists. Note next that

$$(\mathbf{I} - \mathbf{Q})(\mathbf{I} + \mathbf{Q} + \mathbf{Q}^2 + \cdots + \mathbf{Q}^n) = \mathbf{I} - \mathbf{Q}^{n+1}.$$

Thus multiplying both sides by $\mathbf{N}$ gives

$$\mathbf{I} + \mathbf{Q} + \mathbf{Q}^2 + \cdots + \mathbf{Q}^n = \mathbf{N}(\mathbf{I} - \mathbf{Q}^{n+1}).$$

Letting n tend to infinity we have

$$\mathbf{N} = \mathbf{I} + \mathbf{Q} + \mathbf{Q}^2 + \cdots .$$

Let s_i and s_j be two transient states, and assume that i and j are fixed. Let B_k be a random variable which equals 1 if the chain is in state s_j after k steps, and equals 0 otherwise, given that the starting state is s_i. For each k, this random variable depends upon both i and j. We have

$$\mathsf{Pr}[B_k = 1] = q_{ij}^k, \quad \mathsf{Pr}[B_k = 0] = 1 - q_{ij}^k.$$

where $q_{ij}^{(k)}$ is the ij^{th} entry of $\mathbf{Q}^k$. Therefore, $B_k \sim Ber(q_{ij}^k)$, and so $\mathbf{E}[B_k] = q_{ij}^{(k)}$. The expected number of times the chain is in state s_j in the first n steps, given that it starts in state s_i, is clearly

$$\mathbf{E}[B_0 + B_1 + \cdots B_n] = q_{ij}^{(0)} + q_{ij}^{(1)} + \cdots + q_{ij}^{(n)}.$$

Letting n tend to infinity we have

$$\mathbf{E}[B_0 + B_1 + \cdots] = q_{ij}^{(0)} + q_{ij}^{(1)} + \cdots = n_{ij}.$$

□

For an absorbing Markov chain $\mathbf{P}$, the matrix $\mathbf{N} = (\mathbf{I} - \mathbf{Q})^{-1}$ is called the **Fundamental Matrix** for $\mathbf{P}$.

Time to Absorption We now consider the question: Given that the chain starts in state s_i, what is the expected number of steps before the chain is absorbed? The answer is given in the next theorem.

Theorem 2.13. **[70, Theorem 11.5]** *Let t_i be the expected number of steps before the chain is absorbed, given that the chain starts in state s_i, and let $\mathbf{t}$ be the column vector whose i^{th} entry is t_i. Then*

$$\mathbf{t} = \mathbf{N}\mathbf{c},$$

where $\mathbf{c}$ is a column vector all of whose entries are 1.

Proof. If we add all the entries in the i^{th} row of $\mathbf{N}$, we will have the expected number of times in any of the transient states for a given starting state s_i, that is, the expected time required before being absorbed. Thus, t_i is the sum of the entries in the i^{th} row of $\mathbf{N}$. If we write this statement in matrix form, we obtain the theorem. □

Thus if the vector **u** denotes the distribution of the initial state variable, then the average absorption time, given that the chain starts from any arbitrary state, is given by the inner product of **u** and **t**.

2.5 Pseudorandomness and Distinguishing Attack

In Section 1.3.1, we had introduced the notion of pseudorandomness in the context of stream ciphers. We had pointed out that stream ciphers are generators of pseudorandom sequences. In this part we will discuss the notion of pseudorandom generators formally. Loosely speaking, these are efficient deterministic programs that expand short, randomly selected seeds (i.e., Secret Keys) into much longer "pseudorandom" bit sequences that are computationally indistinguishable from truly random sequences by efficient algorithms. Hence the notion of computational indistinguishability (i.e., indistinguishability by efficient procedures) also plays a pivotal role in our discussion.

2.5.1 Computational Indistinguishability

As stated earlier, the concept of computational indistinguishability is the basis for our definition of pseudorandomness. The concept of efficient computation leads naturally to a new kind of equivalence between objects: Objects are considered to be computationally equivalent if they cannot be differentiated by any efficient procedure.

Before we proceed we must define the word "efficient" procedure, or give some notion about the term "efficient". Consider an algorithm A that takes as input an element $x \in S$, where the cardinality of the set S is bounded by $2^{p(n)}$ for some polynomial p, i.e., x can be encoded in atmost $p(n)$ bits. Then we will call A efficient if the total number of computational steps taken by A is bounded by $\mathsf{poly}(n)$ for some polynomial function poly. Algorithms can be either deterministic or probabilistic. A deterministic algorithm A_{det}, for a given input, always returns the same output. Thus if R is the output space of the algorithm, then A_{det} is simply a map between $S \to R$. A probabilistic algorithm A_{prob}, uses some additional random strings, that it generates internally to arrive at an output. Since, at each instance the algorithm may generate different random strings, the output of probabilistic algorithms are not always the same. The additional random strings used by A_{prob} is often called **random coins** of A_{prob}. If C denotes the random coin space, then A_{prob} is essentially a map from $S \times C \to R$.

Suppose we have two infinite sequences $\{x_n\}_{n\in\mathbb{N}}$ and $\{y_n\}_{n\in\mathbb{N}}$, and we need to devise an algorithm D that will take any one of the two sequences and determine if it is $\{x_n\}_{n\in\mathbb{N}}$

or not. If yes, the algorithm outputs 1, and we say that D has accepted the sequence. Otherwise, the algorithm outputs 0, and we say that D has rejected the sequence.

The sequences $\{x_n\}_{n\in\mathbb{N}}$ and $\{y_n\}_{n\in\mathbb{N}}$ are said to be computationally indistinguishable if no efficient procedure can tell them apart. In other words, no efficient algorithm D can accept infinitely many x_n's while rejecting the y_n's (i.e., for every efficient algorithm D and all sufficiently large n, it holds that D accepts x_n if and only if D accepts y_n).

Similarly, two distributions are called computationally indistinguishable if no efficient algorithm can tell them apart. Given an efficient algorithm D, we consider the probability that D accepts (e.g., outputs 1 on input) a string taken from the first distribution. Likewise, we consider the probability that D accepts a string taken from the second distribution. If these two probabilities are close, we say that D does not distinguish the two distributions. We will introduce the following definition at this point.

Definition 2.14. Probability Ensemble [67, Chapter 3.2]: Let I be a countable[1] index set. An ensemble indexed by I is a sequence of random variables indexed by I. Namely, the family of random variables $X = \{X_i\}_{i\in I}$, where each X_i is a random variable, is an ensemble indexed by I.

Typically, we shall use $\mathbb{N}$ as the index set. An ensemble of the form $X = \{X_n\}_{n\in\mathbb{N}}$ has each X_n ranging over strings of length polynomial in n. We will now formally denote the notion of computational indistinguishability.

Definition 2.15. Polynomial-time Indistinguishability [67, Chapter 3.2] Two ensembles, $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$, are indistinguishable in polynomial time if for every probabilistic polynomial-time algorithm D, every positive polynomial $p(\cdot)$, and all sufficiently large n's,

$$|\mathsf{Pr}[D(X_n) = 1] - \mathsf{Pr}[D(Y_n) = 1]| < \frac{1}{p(n)}.$$

The LHS of the above inequality is called **Distinguishing Advantage** of D distinguishing X from Y and is denoted by the symbol $\mathbf{Adv}_D(X, Y)$.

The probabilities in the foregoing definition are taken over the corresponding random variables X_i (or Y_i) and the internal random coin of algorithm D. Implicit in the definition is the condition that the algorithm, D, must decide based on a single sample from one of the distributions. One might conceive of a situation in which the algorithm trying to distinguish between two distributions, could access as many samples as it needed. Hence two ensembles that cannot be distinguished by polynomial-time algorithms looking at multiple samples are deemed indistinguishable by **polynomial-time sampling**.

[1] A set is I is said to be countable if there exists a bijection f between $I \to \mathbb{N}$

It turns out that if some polynomial-time algorithm can generate samples in polynomial time, then indistinguishable by polynomial-time sampling is equivalent to computational indistinguishability. The full discussion of this axiom is out of the scope of this thesis. Interested readers may please refer [67, Chapter 3.2]

Next we will define the notion of pseudorandomness. Note that U_n denotes a random variable uniformly distributed over the set of strings of length n. The ensemble $\{U_n\}_{n\in\mathbb{N}}$ is called the standard uniform ensemble. Ensembles that are computationally indistinguishable from a uniform ensemble are called pseudorandom. Thus we have the following definition.

Definition 2.16. Pseudorandom Ensembles [67, Chapter 3.2]: The ensemble$X = \{X_n\}_{n\in\mathbb{N}}$ is called pseudorandom if there exists a uniform ensemble $U = \{U_{l(n)}\}_{n\in\mathbb{N}}$ such that X and U are indistinguishable in polynomial time.

Here $l : \mathbb{N} \to \mathbb{N}$ is a function computable in polynomial time. Note that the length of each string X_n is not necessarily n, whereas we require that the length of U_n be strictly equal to n. In fact, the previous definition requires that, with very high probability, the length of X_n to be equal to $l(n)$. We will now formally define the notion of a pseudorandom generator.

Definition 2.17. Pseudorandom Generator [67, Chapter 3.2] A pseudorandom generator is a deterministic polynomial-time algorithm G satisfying the following two conditions:

1. Expansion: There exists a function $l : \mathbb{N} \to \mathbb{N}$ such that $l(n) > n$ for all $n \in \mathbb{N}$, and $|G(s)| = l(|s|)$ for all $s \in 0, 1$. (Here $|\cdot|$ denotes the length of a binary string)
2. Pseudorandomness: The ensemble $\{G(U_n)\}_{n\in\mathbb{N}}$ is pseudorandom.

The function l is called the expansion factor of G.

The input s to the generator is called its seed or Secret Key. The expansion condition requires that the algorithm G map n-bit long seeds into $l(n)$-bit long strings, with the condition $l(n) > n$. The pseudorandomness condition requires that the output distribution induced by applying algorithm G to a uniformly chosen seed be polynomial-time-indistinguishable from a uniform distribution.

2.5.2 Distinguishing the Distributions $Ber(p_0)$ and $Ber(p_0(1+q_0))$

We will end this section with a famous result proposed in [102], which aims to distinguish two distributions X and Y such that $X \sim Ber(p_0)$ and $Y \sim Ber(p_0(1+q_0))$ using multiple samples.

Theorem 2.18. **[102, Theorem 2]** *We have two distributions X and Y such that $X \sim Ber(p_0)$ and $Y \sim Ber(p_0(1+q_0))$ for all i. Given that there exists an efficient algorithm to extract multiple samples from these distributions, then for small q_0, $n = O\left(\frac{1}{p_0 \cdot q_0^2}\right)$ samples suffice to distinguish X from Y with a constant probability of success. (Note that given two functions $f(n)$ and $g(n)$, we say that $f(n) = O(g(n))$ if there exist constants $c > 0$ and $N > 0$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq N$.)*

Proof. Let N_X, N_Y be the random variables specifying the number of occurrences of 1 in n samples. Then we have already seen that $N_X \sim Bin(n, p_0)$ and $N_Y \sim Bin(n, p_0(1+q_0))$. Their expectations, variances and standard deviations are:

$$\mathbf{E}(N_X) = np_0, \ \mathbf{E}(N_Y) = np_0(1+q_0),$$

$$\mathrm{Var}(N_X) = np_0(1-p_0) \approx np_0, \ \mathrm{Var}(N_Y) = np_0(1+q_0)(1-p_0(1+q_0)) \approx np_0(1+q_0),$$

$$\mathrm{SD}(N_X) \approx \sqrt{np_0}, \ \mathrm{SD}(N_Y) \approx \sqrt{np_0(1+q_0)}.$$

Here $\mathrm{SD}(X)$ denotes the standard deviation of any random variable X. We will consider of that value of n to be sufficient, that induces a difference of at least one standard deviation between the expectations of the two distributions. Thus

$$\begin{aligned} \mathbf{E}(N_Y) - \mathbf{E}(N_X) &\geq \mathrm{SD}(N_X), \\ np_0(1+q_0) - np_0 &\geq \sqrt{np_0} \\ np_0 q_0 &\geq \sqrt{np_0} \\ n &\geq \frac{1}{p_0 q_0^2}. \end{aligned}$$

Thus, $n = O\left(\frac{1}{p_0 \cdot q_0^2}\right)$ is sufficient for the distinguishing attack. □

In [102], this result was used to distinguish the output if the stream cipher RC4 from random. It was proven that the probability that the second byte Z_2 output by RC4 is 0 with probability $\mathsf{Pr}[Z_2 = 0] = \frac{2}{N}$, where $N = 256$. Note that for an ideal cipher $ppr[Z_2 = 0] = \frac{1}{N}$. So for RC4 we have $N_Y \sim Ber\left(\frac{2}{N}\right)$ whereas for an ideal cipher $N_X \sim Ber\left(\frac{1}{N}\right)$. So we have $p_0 = q_0 = \frac{1}{N}$, and by the above result $O(N)$ samples would be sufficient to distinguish RC4 from random.

2.6 Fault Attacks

Fault Attacks belong to a class of Side Channel Attacks where the attacker intentionally tampers with the circuitry of the device implementing any cryptosystem, with the view of gaining any non-trivial information by the output produced by such tampered device.

Ever since, Kocher et al.'s work [95] on Differential Power Analysis, such attacks became very popular among the cryptological community. In fact, power analysis attacks can be regarded as a sub-class of a **non-invasive** side channel attacks, that do not require the attacker to interfere with the circuitry or the electrical signals of the device under attack. Non-invasive attacks are extremely popular because of the relative low capital investment required to perform the attack. For example, power analysis attacks would typically require an oscilloscope to measure power consumption traces, a computer to monitor the attack and a few probes to connect the device to the oscilloscope.

Before the publication of Skorobogatov and Anderson's work [125] on optical fault induction, **invasive** and **semi-invasive** side-channel attacks like fault/error injections required relatively high capital investment. One of the most popular way of introducing errors was by inducing glitches, i.e., introducing voltage transients in the clock or power line of the device under attack. In time, chip manufacturers started to take adequate precautions to prevent such side channel attacks. For example random clock jitter was often introduced to make power analysis more difficult, and circuits were devised that could sense the introduction of glitches and reset the device immediately.

In such scenario, Skorobogatov and Anderson's work [125], was of extremely significant in the field of Side Channel Cryptanalysis. First of all, due to the rapid shrinking of the feature size of a CMOS transistor, any kind of invasive or semi-invasive attack was getting more and more difficult. Furthermore, the work showed that the attack could be carried out by illuminating any flip-flop with a standard camera flashgun or a laser pointer that was bought for $30 and $8 respectively.

The authors of [125], observed that any semiconductor transistor was sensitive to the effect of ionizing radiations such as lasers [71]. Laser energy was known to be able to ionize any integrated circuit's semiconductors if its photon energy was greater than the band gap of the semiconductor. Laser radiation of wavelength 1.06 μm wavelength and 1.17 eV photon energy has a penetration depth of 700 μm was used to ionize the p-n junctions of a semiconductor in [82]. [112] notes that it is possible to move from infrared radiations to visible light as photon absorption increases at higher frequencies. This coupled with the fact that

- modern chips have become considerably thinner, and

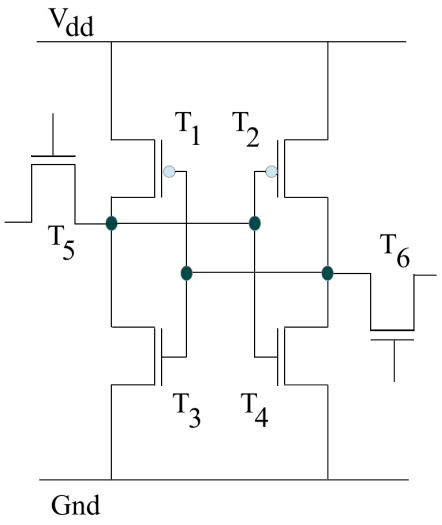


FIGURE 2.2: Structure of a CMOS flip-flop

- considerably less energy is required to ionize transistors with smaller feature size,

have made visible light lasers an attractive choice for inducing faults in semiconductor devices. The authors [125] performed their experiments on a common micro-controller chip PIC16F84 which contains around 68 bytes of on chip memory. The structure of a flip-flop in this chip is shown in Figure 2.2. The flip-flop made with CMOS logic consists of 6 flip-flops $T_1 - T_6$. The authors observed that if T_3 was opened for a very short time using some external stimulus it would cause the flip-flop to change state, i.e., if the flip-flop was storing the Boolean logic 0 it would change to 1 and vice versa. A common camera flash was used to illuminate the chip. The chip was de-packaged and the camera flash was fixed with duct tape over it. By exposing the flip-flops of the chip to light from the camera flash operated remotely from a PC, the authors were able to change the state of the one of the individual flip-flops of the chip.

The only limitation in performing the attack was that the flash did not produce even monochromatic light, so it was difficult to control the area of the chip where the light is applied. To overcome this limitation, the attack was repeated with a laser pointer with similar results.

Ever since, fault attacks have been extensively analyzed by the research community for more than a decade [37, 41]. Block ciphers like Advanced Encryption Standard and the Digital Encryption Standard, Public key cryptosystems like implementations of CRT-RSA, DSA/RSA based signatures, Elliptic Curve Cryptosystems have all come under

the purview of such attacks (please refer to [83] for an extensive analysis of all such attacks). Ever since the work of Hoch and Shamir [76], there has been extensive fault analysis of stream ciphers too.

2.6.1 Fault Models

Over the years, there have been numerous fault attacks reported against stream ciphers [35, 36, 38, 78, 79, 81, 85, 90, 91, 119]. All these attacks are performed under roughly similar adversarial situations, i.e., situations in which the attacker has the ability to inject different types of faults in the underlying cryptographic device. In this part, we will briefly discuss these adversarial situations which we will call **fault models** for convenience.

- The attacker is in possession of the physical device in which the stream cipher has been implemented. He therefore, knows the IV and the keystream generated by the stream cipher. Additionally, in [85], the attacker is assumed to be able to derive keystream sequences off the cipher by inputing IV's of his own choice.
- By applying optical faults, the attacker is assumed to be able to apply bit-flipping faults with a partial control of their location. That is to say, the attacker is able only partially able to control the register location where he injects a bit-flipping fault. Thus, he can not exactly choose the location of the fault. Some attacks like [35, 85], assumes that the attacker, once he faults a particular location, is additionally able to inject faults on that location over and over again without restriction.
- The attacker is able to exercise full control over the timing of application of faults, by choosing the time of the fault trigger by synchronizing it with the I/O signal. Shift registers are regularly clocked, and one keystream bit is computed per clock cycle. Hence, the attacker can identify steps in the execution, and so it is possible for the attacker to inject a fault at any particular point of the operation. Another popular way of controlling the fault timing would be by synchronizing it with the power consumption curves of the device implementing the cryptosystem [52]. This method can be used when there is no possible way to trigger the faults using I/O signals of the device.
- The attacker from time to time resets the cryptographic device to its original state.

In this thesis we will discuss differential fault attacks on selected stream ciphers under different fault models. We will generally start with fault models that assume that the

adversary can apply single-bit flipping faults that can be synchronized with with any particular stage of the cipher operation. Under some cases, we will analyze situations in which the adversary is unable to guarantee that the injected fault has flipped only a single bit and/or situations in which the attacker is unable to synchronize faults with either the I/O signal of the chip or its power consumption trace.

2.7 Scan based Side Channel Attacks

While manufacturing any hardware product, DFT techniques are design efforts that are specifically employed to ensure that a device is testable. Single scan-chains are one of the most popular and effective ways of providing testability to any hardware device. The objective of the scan-chain is to make testing easier by providing a way to set and observe every flip-flop in an IC. Unlike the functional tests that check chip functionality, scan tests cover stuck-at-faults, caused by manufacturing problems. Physical manufacturing defects, such as

- silicon defects, photo-lithography defects, mask contamination
- process variation or defective oxide etc.

may lead to electrical defects such as shorts (bridging faults), opens, transistors stuck on open, changes in threshold voltage etc. which may lead to digital logic being stuck at either the 0 or 1 value at one or many of the flip-flops. It may also lead to slower transitions among the flip-flops causing delay faults which hamper the proper functioning of a cryptosystem.

Scan-chain testing can be done to check whether a chip is functioning normally or not. It provides the designer an easy way to ascertain whether the device has succumbed to the above mentioned defects or not. Scan has been generally accepted as the standard method of testing chips due to high fault coverage and relatively lower area overhead. Inserting scan-chains while designing the chip requires a few additional/multiplexed pins to the primary inputs/outputs to serve as the SCAN-ENABLE, SCAN-IN and SCAN-OUT. Internally, there is little impact on the design since the standard flip-flops (FFs) are replaced by scan-enabled flip-flops (SFFs) (i.e., flip-flops with an input multiplexer) which are then linked to one another creating a shift register (scan-chain). Scan-enabled flip-flop contains a multiplexer to select either a normal mode functioning or a scan mode functioning. By suitably altering the control value to the multiplexer, the chip can be used for normal or test mode of operation. After selecting test mode, the user is able to input test patterns of his choice into the device and thereafter scan out the contents

of all the flip-flops connected to the scan-chain. An example of a scan-chain the Figure 2.3. SCAN-ENABLE selects between normal/functional and test mode operations. Note that the SCAN-ENABLE signal controls each multiplexer, choosing between the normal mode input of the FF or the output of the previous SFF in the chain.

Scan chains allow the tester to control and observe internal states of the circuit by loading/unloading input patterns/test responses. In order to load test patterns, the SCAN-ENABLE signal is asserted and each bit of the pattern is shifted in at each clock. When the entire input pattern is serially loaded, the SCAN-ENABLE signal is deactivated for one or more cycles. During these cycles (capture mode), the input test patterns are applied to the combinatorial logic and the response is stored in the sequential elements. SCAN-ENABLE is activated again (shift-mode) and the internal state can be scanned out and be analyzed by the tester. At the same time the next input pattern is loaded. In other words, using scan chains essentially transforms the circuit into pure combinatorial logic, which is much easier to test than sequential logic. Scan testing achieves the dual objective of testability controllability and observability. Controllability is the ability to change the state of internal nodes using only the primary inputs, while observability means the ability to observe the state of internal nodes using only the primary outputs. Inserting scan-based DFT structures in a design allows the designer to achieve very high fault coverage using Automatic Test Pattern Generation (ATPG) tools during manufacturing tests.

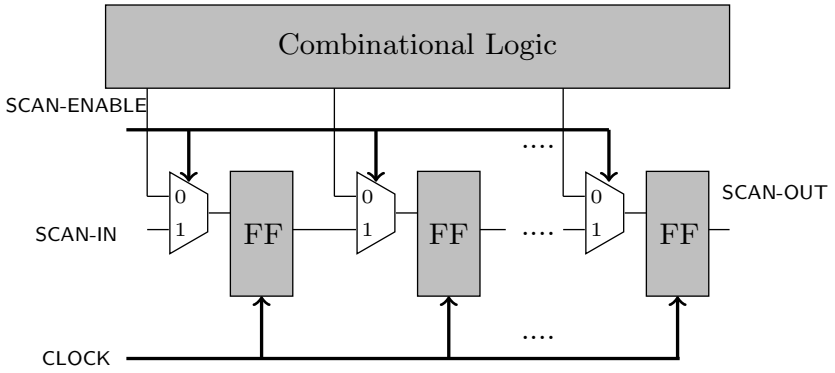


FIGURE 2.3: Example of a Scan-chain using Multiplexers

2.7.1 Introduction to Scan Attacks

Scan-chains are incorporated in a circuit for thorough structural testing. However, it can be targeted by attackers to extract secret information from security chips. The scan chains can act as a side-channel through which confidential information, such as secret keys stored inside the chip, can be recovered.

Though the scan-chain test approach provides better controllability and observability to the test engineer, it can be used by an attacker to read chip internal data, to read stored secret information and to determine the position of all the scan elements in a chain. Only a few pins need to be externally controlled and monitored: the SCAN-IN, the SCAN-ENABLE, and the SCAN-OUT. By observing the scan chain output during the operations of the underlying cryptosystem repeatedly, the secret information can be deduced through knowledge of the encryption algorithm. The ability of the circuit to switch between normal and test mode as well as the possibility to stop the execution of the cryptographic circuit at any time in test mode and scan out the responses is exploited in scan-based attacks.

The following assumptions are made in the differential scan attack presented in this work.

- The SCAN-ENABLE, SCAN-IN, SCAN-OUT pins can be controlled by the attacker.
- Details of the cryptographic algorithm are known to the attacker.
- The time required to execute the target operation is known to the attacker. In other words, the attacker is aware of the operations executed in any single-step of the cryptographic algorithm.
- The attacker is able to manipulate all the public variables in the cipher. In a stream cipher, this is typically the IV.
- The secret key is assumed to be securely stored in memory and not included on the scan chain.
- The attacker is assumed to have physical access to the circuit containing the scan-chain DFT, and hence can control and observe the scan chain contents through the external scan inputs and outputs.

We will present a more comprehensive analysis of such attacks in Chapter 7. In [11], A Scan based attack on the stream cipher Trivium [43] was presented. It will be shown why the same attack can not be extended to MICKEY 2.0 [16], and suggests an alternative strategy to attack MICKEY 2.0 via Scan Chains. Further, in [11], an XOR gate based countermeasure was suggested to protect Scan Chains from cryptanalytic attacks. It will also be shown that this countermeasure may fail to protect the underlying cryptosystem under certain classes of cryptanalytic attacks. A novel Double Feedback XOR-CHAIN countermeasure is proposed that is shown to be secure against the given class of cryptanalytic attacks. It is also proven than that such a Double Feedback XOR-CHAIN structure, like an ordinary Scan Chain, may also be used for DFT purposes.

2.8 Conclusion

This chapter was mainly intended for the purposes of providing the necessary theoretical background required to read this thesis comfortably. Now that we have covered the preliminary topics of mathematics and cryptography that constitute the foundation of this thesis, we can get into the technical results. One may refer back to the preliminaries whenever similar methods in the following chapters are encountered.

Chapter 3

Analysis of RC4 variants

RC4 is the most widely used software stream cipher and is used in popular protocols such as Transport Layer Security (TLS) (to protect Internet traffic) and WEP (to secure wireless networks). Because of its simplicity and speed in software, RC4 is immensely popular among designers and cryptanalysts alike. The cipher was designed by Ron Rivest of RSA Security in 1987, and it remained a trade secret until 1994, when an anonymous user leaked its source code to the Cypherpunks mailing list. Since then the cipher has undergone rigorous cryptanalysis, and a number of weaknesses have been reported against the cipher. Most prominent among them are as follows.

- One of the most famous cryptanalytic result on RC4 was shown by Mantin and Shamir in 2001 [102]. They showed that the second output byte of the cipher was biased towards zero with probability $\frac{2}{N}$, where $N = 256$ is the size of the state array S used in RC4. The bias is significant since for an ideal cipher this probability should have been only $\frac{1}{N}$. This leads to a distinguishing attack on RC4 with around $O(N)$ samples.
- A state recovery attack was proposed by Maximov and Khovratovich in [105]. The attack recovers the internal state of the cipher in around 2^{241} operations.

Due to the aforesaid reasons, there has been extensive research in recent years to come up with RC4-like stream ciphers that while marginally slower in software, would wipe out the known shortcomings of RC4. Many such ciphers like RC4A [114], NGG [111], GGHN [68], VMPC [138], RC4+ [100] have been proposed by various researchers to fulfill this objective. However, most of the aforementioned ciphers have had distinguishing attacks reported against them [104, 115, 131, 132].

In this chapter, we take a closer look at two of the ciphers of this family - GGHN and RC4+. The GGHN stream cipher was presented in CISC 2005 [68]. This cipher has

been motivated from RC4 with the idea to obtain further speed-up by considering word-oriented keystream output instead of byte-oriented ones (in this case 1 word = 4 bytes). We prove that there exist a large number of short cycles of length equal to the length of the state array used in the cipher. Then towards having a theoretical analysis of GGHN type evolution, we study a randomized model of this cipher. Using the thoery of Markov chains, we show how this model evolves to all zero state much faster than what is expected in an ideal cipher.

The RC4+ stream cipher was proposed by Maitra and Paul at Indocrypt 2008 [100]. The authors had claimed that RC4+ ironed out most of the weaknesses of the RC4 stream cipher and was only marginally slower than RC4 in software. In this chapter we show that it is possible to mount a distinguishing attack on RC4+ based on the bias of the first output byte. The distinguisher requires around 2^{26} samples produced by different keys of RC4+. In the second part of the chapter we study the possibility of mounting the differential fault attack on RC4+, along the lines of the fault attack on RC4 proposed by by Biham et. al. in FSE 2005. We will show that that the RC4+ is vulnerable to differential fault attack and it is possible to recover the entire internal state of the cipher at the beginning of the PRGA by injecting around $2^{17.2}$ faults.

3.1 GGHN Stream Cipher

The GGHN (name follows from the first character of the surnames of the four coauthors Gong, Gupta, Hell and Nawaz) cipher [68] was proposed from the motivation of extending RC4 from the byte-oriented model to word-oriented model. This cipher has received attention in literature as evident from the cryptanalytic results by Paul and Preneel (Asiacrypt 2006) [115], Tsunoo, Saito, Kubo and Suzaki (IEEE-IT, 2007) [132] and Kircanski and Youssef (CCDS 2010) [89].

It is needless to mention that RC4 is the most popular stream cipher in the area of cryptology from commercial as well as theoretical interest. The algorithm has two parts: the Key Scheduling Algorithm (KSA) and the Pseudo-Random Generation Algorithm (PRGA), as given below in Algorithm 3.1 and Algorithm 3.2. As can be seen the cipher consists of an array S of 2^8 elements of 1 byte each. The elements in the array represent some permutation on the set $\{0, 1, \ldots, 255\}$.

Since the structure of RC4 is quite elegant, it has attracted in-depth cryptanalysis in many papers (one may refer to the recent papers [101, 105, 122] and the references therein).

Input: Secret Key K.
Output: S-Box S generated by K.

Initialize $S = \{0, 1, 2, \ldots, N-1\}$;
Initialize counter: $j = 0$;
for $i = 0, \ldots, N-1$ **do**
  $j = (j + S[i] + K[i]) \bmod N$;
  Swap $S[i] \leftrightarrow S[j]$;
end

Algorithm 3.1: RC4 KSA

Input: S-Box S, from KSA output.
Output: Pseudorandom stream z.

Initialize the counters: $i = j = 0$;
while *keystream is required* **do**
  $i = (i + 1) \bmod N$;
  $j = (j + S[i]) \bmod N$;
  Swap $S[i] \leftrightarrow S[j]$;
  $z = S[(S[i] + S[j]) \bmod N]$;
end

Algorithm 3.2: RC4 PRGA

One limitation of RC4 is that its output generation is in bytes (8-bits) and it is natural to extend the idea to word oriented (4 or 8 bytes in general) output that will produce the keystream much faster. The 32 or 64-bit processors are ready for such applications. For storing 8-bit permutations, one needs only 2^8 bytes, but for storing 32-bit permutations, a huge array of size 2^{32} words (1 word = 4 bytes, here) is required. As this is impractical, there are several efforts to work with small random arrays, instead of complete random permutations. The GGHN cipher [68] is an example of such a cipher which is indeed as simple and elegant looking as RC4. On the other hand, as it will be shown in this chapter, it is a classical example that a simple and elegant cipher can have quite a few weaknesses. Before proceeding further, let us first describe the GGHN cipher.

Input: S-Box S: an m-bit integer array of $N = 2^n$ locations, initially loaded with certain predefined value;
Input: Key array K: an m-bit integer array of l locations;
Input: An integer r for number of rounds;
Output: Pseudorandom S-Box S and a pseudorandom m-bit integer k;

$j = 0, k = 0, t = 0, M = 2^m$;
while $t < r$ **do**
  for $i = 0, \ldots, N-1$ **do**
    $j = (j + S[i] + K[i \bmod l]) \bmod N$;
    Swap $S[i] \leftrightarrow S[j]$;
    $S[i] = (S[i] + S[j]) \bmod M$;
    $k = (k + S[i]) \bmod M$;
  end
  $t = t + 1$;
end

Algorithm 3.3: GGHN-KSA(n, m)

Input: S-Box S and integer k from GGHN-KSA(n, m);
Output: m-bit Keystream words z

$i = 0, j = 0, M = 2^m$;
while *Keystream is generated* **do**
  $i = (i + 1) \bmod N$;
  $j = (j + S[i]) \bmod N$;
1 $k = (k + S[j]) \bmod M$;
  $z = (S[(S[i] + S[j]) \bmod N] + k) \bmod M$;
2 $S[(S[i] + S[j]) \bmod N] = (k + S[i]) \bmod M$;
end

Algorithm 3.4: GGHN-PRGA(n, m)

Note that the authors define a family of ciphers GGHN(n, m) indexed by the parameters n, m. The version proposed for use is GGHN(8, 32), i.e., the version in which the array S has 2^8 entries each of size 32 bits. Regarding the security of GGHN(8, 32), the following results are available in literature:

- a $2^{32.89}$ distinguisher [115] based on the observation that the least significant bit of the keystream words are biased to zero;
- a $2^{30.02}$ distinguisher [132] based on the equality of the least significant bits of the first two keystream words;
- structural weakness of GGHN$(8, 32)$ has been presented in [89], where it has been shown that out of 8240 bits ($256 \cdot 32$ bits for S, 32 bits for k and $8 \cdot 2$ bits for i, j) of the internal state if 2064 bits can be obtained (a fault attack is presented too on how to obtain those bits) then the complete state can be explored effectively.

3.1.1 Our Results

In Section 3.2, we study the evolution of GGHN PRGA. We first note that once all the words become even, half of the state array will not be updated further. More importantly, we identify a lot of short cycles in this cipher. As a general result, given a GGHN(n, m) cipher, we prove by construction that there are cycles of length 2^n which starts from $i = 0, j = 0, k \equiv -1 \bmod 2^n, S[r] \equiv 1 \bmod 2^n,\ \forall r \in [0, 2^n - 1], r \neq 2$ and $S[2] \equiv 0 \bmod 2^n$ with the constraint that

$$k_0 \equiv -\left(s_0 + 3 \cdot s_1 + \sum_{r=3}^{2^n-1} s_r\right) \bmod 2^m,\ s_2 \equiv -\left(3 \cdot s_1 + \sum_{r=3}^{2^n-1} s_r\right) \bmod 2^m.$$

Thereafter, we noted the following comment related to certain weakness of the cipher pointed out by the authors themselves [68, Section 4.7].

"When all entries are even and k is even, then all outputs as well as all future entries will be even, resulting in a biased keystream. The state update function in RC4(n, m) is not an invertible mapping so it will always be possible to enter one of these weak states. However the probability that all state entries, as well as k are even is very low, 2^{-257}. From this we can conclude that these weak states are of no concern to the security of the cipher."

In [68], the authors used the notation RC4(n, m) to denote GGHN(n, m) as RC4(n, m) and while coming up with the probability 2^{-257}, they considered $n = 8, m = 32$ (note that $N = 2^n, M = 2^m$). Now for the cipher to enter the all even state all the N LSBs of S and the LSB of k must be zero. Assuming that all these $N + 1$ LSBs are identically and independently distributed according to $Ber(\frac{1}{2})$, the joint probability of them being zero is $2^{-(N+1)} = 2^{-257}$, and so the authors assume that i would take around 2^{257} rounds for the cipher to reach this state.

We got motivated to investigate a theoretical model of this cipher where all the indices are chosen uniformly at random. This is presented in Section 3.3, where we study a randomized version of GGHN cipher. Using the theory of Markov chains, we show that the number of iterations to reach the all even state is much less than 2^{N+1}. For $N = 256$, instead of 2^{257}, it is only $2^{142.16}$. In fact, our analysis shows, that the expected number of iterations to reach for the cipher to reach the all zero state is $32 \cdot 2^{142.16} = 2^{147.16}$ only.

3.2 Short Cycles in GGHN(n, m)

Refer to GGHN-PRGA(n, m) as in Algorithm 3.4. Immediately one can note the following result.

Lemma 3.1. *Consider the situation when all the elements of S and k have evolved to even values. Then half of the elements of S which are indexed by odd values will never be modified further.*

Proof. Since all the values of S array are even, $S[i] + S[j]$ is also even and so is $(S[i] + S[j])$ mod N. Thus, in the line marked 2 of Algorithm 3.4, the left hand side $S[(S[i] + S[j])$ mod $N]$ will be $S[l]$ where l is even. Thus only the even indexed locations will be updated and the odd indexed locations will not be modified at all. □

This is indeed a weakness of the design. However, we now present more important results related to short cycles of the cipher. One may note that there are well known short cycles for RC4 which are famous as Finney cycles, but the initial conditions in RC4 PRGA are so chosen that such cycles cannot occur [60]. We now show that the GGHN PRGA(n, m) algorithm may fall into a cycle where a given state repeats after a certain number of iterations. To explain things clearly, we move step by step with several intermediate results and examples. We will first illustrate the case for GGHN$(2, 2)$, i.e., with the array S having 4 cells each containing 2 bit data.

Proposition 3.2. *For the GGHN PRGA$(2, 2)$, the initial condition*

$$S[0] = 1,\ S[1] = 1,\ S[2] = 0,\ S[3] = 1,\ i = j = 0,\ k = 3$$

forms a cycle of length 4*.*

Proof. The algorithm starts from the given state $i = 0,\ j = 0,\ k = 3,\ S[0] = 1,\ S[1] = 1,\ S[2] = 0,\ S[3] = 1$ and goes through the following states before returning to the initial one:

$i = 1,\ j = 1,\ k = 0,\ S[0] = 1,\ S[1] = 1,\ S[2] = 1,\ S[3] = 1,$

$i = 2,\ j = 2,\ k = 1,\ S[0] = 1,\ S[1] = 1,\ S[2] = 2,\ S[3] = 1,$

$i = 3,\ j = 3,\ k = 2,\ S[0] = 1,\ S[1] = 1,\ S[2] = 3,\ S[3] = 1.$

It is interesting to see that the value $S[i] + S[j] \equiv 2 \bmod 4$ is an invariant, and thus at every stage of the cycle, only the value $S[2]$ gets altered in the array S. Also $i = j$ at all the stages of the cycle. □

Now let us present a more generalized result.

Lemma 3.3. *For the GGHN PRGA$(2, m)$ with $m \geq 2$, consider an initial condition of the form $i = j = 0,\ S[0] = s_0 \equiv 1 \bmod 4,\ S[1] = s_1 \equiv 1 \bmod 4,\ S[2] = s_2 \equiv 0 \bmod 4,\ S[3] = s_3 = 1 \bmod 4,\ k = k_0 \equiv 3 \bmod 4$. If*

$$k_0 \equiv -(s_0 + 3s_1 + s_3) \bmod 2^m \text{ and}$$

$$s_2 \equiv -(3s_1 + s_3) \bmod 2^m$$

are satisfied, then a cycle of length 4 will be formed.

Proof. It is clear that the values in S array and k are in the range $[0, \ldots, 2^m - 1]$ and i, j are in $[0, \ldots, 3]$. Because of the restrictions placed on $S[r],\ r = 0, 1, 2, 3$ and k the

values of i and j will always remain same during the evolution. Furthermore, the value $S[i] + S[j] \equiv 2 \bmod 4$ is an invariant, and hence, $S[2]$ is the only location of the array S that undergoes alteration at each step of the cycle as follows. In the following state sequence '+' stands for addition modulo 2^m.

$$\begin{pmatrix} i = 1,\ j = 1,\ k = k_0 + s_1 \\ S[0] = s_0,\ S[1] = s_1,\ S[2] = k_0 + 2s_1,\ S[3] = s_3 \end{pmatrix}$$

$$\begin{pmatrix} i = 2,\ j = 2,\ k = 2k_0 + 3s_1 \\ S[0] = s_0,\ S[1] = s_1,\ S[2] = 3k_0 + 5s_1,\ S[3] = s_3 \end{pmatrix}$$

$$\begin{pmatrix} i = 3,\ j = 3,\ k = 2k_0 + 3s_1 + s_3 \\ S[0] = s_0,\ S[1] = s_1,\ S[2] = 2k_0 + 3s_1 + 2s_3,\ S[3] = s_3 \end{pmatrix}$$

$$\begin{pmatrix} i = 0,\ j = 0,\ k = 2k_0 + 3s_1 + s_3 + s_0 \\ S[0] = s_0,\ S[1] = s_1,\ S[2] = 2k_0 + 3s_1 + s_3 + 2s_0,\ S[3] = s_3 \end{pmatrix}$$

For this to represent a cycle the conditions $2k_0 + 3s_1 + s_3 + s_0 \equiv k_0 \bmod 2^m$ and $2k_0 + 3s_1 + s_3 + 2s_0 \equiv s_2 \bmod 2^m$ must hold. That is to say k_0 needs to satisfy the modular equation $k_0 \equiv -(s_0 + 3s_1 + s_3) \bmod 2^m$ and s_2 needs to satisfy the equation $s_2 \equiv -(3s_1 + s_3) \bmod 2^m$.

Note that as $m \geq 2$, $k_0 \equiv -(s_0+3s_1+s_3) \bmod 4 \equiv 3 \bmod 4$ and $s_2 \equiv -(3s_1+s_3) \bmod 4 \equiv 0 \bmod 4$ that are indeed consistent with the condition presented in the statement of this lemma. □

To explain further, if we choose any arbitrary $s_0, s_1, s_3 \in [0, 2^m - 1]$ satisfying $s_0, s_1, s_3 \equiv 1 \bmod 4$, we can easily calculate s_2 and k from the above equations such that the state $i = 0,\ j = 0,\ k = k_0,\ S[0] = s_0,\ S[1] = s_1,\ S[2] = s_2,\ S[3] = s_3$ is the initial state of a cycle of length 4. For example, in the system GGHN$(2, 8)$ if we take $s_0 = 69,\ s_1 = 141,\ s_3 = 9$, using the above equations we get $k_0 = 11$ and $s_2 = 80$ and so $i = 0,\ j = 0,\ k = 11,\ S[0] = 69,\ S[1] = 141,\ S[2] = 80,\ S[3] = 9$ forms a cycle of length 4.

In fact, for GGHN$(2, m)$, there are other examples of short cycles not covered by Lemma 3.3 for cycles of length 4. One can start with any of the following cases and then proceed similar to Lemma 3.3 to get different conditions:

(i) $i = 0, j = 2, k \equiv 3 \bmod 4, S[0] \equiv 1 \bmod 4, S[1] \equiv 3 \bmod 4, S[2] \equiv 0 \bmod 4, S[3] \equiv 3 \bmod 4$,

(ii) $i = 0, j = 2, k \equiv 0 \bmod 4, S[0] \equiv 3 \bmod 4, S[1] \equiv 0 \bmod 4, S[2] \equiv 3 \bmod 4, S[3] \equiv 2 \bmod 4$,

(iii) $i = 0, j = 0, k \equiv 1 \bmod 4, S[0] \equiv 3 \bmod 4, S[1] \equiv 3 \bmod 4, S[2] \equiv 0 \bmod 4, S[3] \equiv 3 \bmod 4$,

(iv) $i = 0, j = 1, k \equiv 0 \bmod 4, S[0] \equiv 2 \bmod 4, S[1] \equiv 1 \bmod 4, S[2] \equiv 2 \bmod 4, S[3] \equiv 2 \bmod 4$.

Similarly the GGHN$(3, m)$ PRGA has a cycle of length 8 of the form $i = 0,\ j = 7,\ k \equiv 0 \bmod 8$ and the array S equal to $\{0, 3, 0, 0, 3, 4, 0, 3\}$ modulo 8; and also the initial state variables $S[i] = s_i$ and $k = k_0$ need to satisfy the following modular matrix equation.

$$\begin{pmatrix} 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 18 & 0 & 4 & 5 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 15 & 0 & 4 & 5 & 7 & 0 & 6 \\ 0 & 0 & 5 & 0 & 1 & 2 & 0 & 0 & 2 \\ 0 & 0 & 15 & 0 & 4 & 5 & 0 & 0 & 6 \end{pmatrix} \cdot \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ k_0 \end{pmatrix} = \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \\ k_0 \end{pmatrix} \bmod 2^m$$

In these equations s_1, s_2, s_4, s_5 can be chosen from $[0, 2^m - 1]$ such that $s_1 \equiv 3 \bmod 8, s_2 \equiv 0 \bmod 8, s_4 \equiv 3 \bmod 8, s_5 \equiv 4 \bmod 8$. The remaining state variables are then calculated by solving the above equations giving the complete state which forms a cycle of length 2^3. For example, in GGHN$(3, 8)$, letting $s_1 = 3, s_2 = 24, s_4 = 83, s_5 = 20$ we can use the above modular matrix equation to get $k_0 = s_6 = 200, s_3 = 216, s_0 = 16, s_7 = 131$.

Now we present a result for GGHN(n, n).

Lemma 3.4. *The state* $i = 0,\ j = 0,\ k = 2^n - 1, S[r] = 1,\ \forall r \in [0, 2^n - 1], r \neq 2,$ *and* $S[2] = 0$ *forms a cycle of length* 2^n *in GGHN*(n, n) *PRGA algorithm.*

Proof. Given this particular initial state, the algorithm runs through the states as given below

$$i = 1, j = 1, k = 0, S[r] = 1,\ \forall r \in [0, 2^n - 1], r \neq 2, S[2] = 1,$$

$$i = 2, j = 2, k = 1, S[r] = 1,\ \forall r \in [0, 2^n - 1], r \neq 2, S[2] = 2,$$

$$\vdots$$

$$i = 2^n - 1, j = 2^n - 1, k = 2^n - 2, S[r] = 1,\ \forall r \in [0, 2^n - 1], r \neq 2, S[2] = 2^n - 1.$$

before returning to the initial state. As before the quantity $S[i] + S[j] \equiv 2 \bmod 2^n$ is an invariant throughout the cycle, and hence $S[2]$ is the only location of the array S to undergo changes. □

Now we construct a cycle in the most general case for GGHN(n, m).

Theorem 3.5. *In the GGHN(n, m) PRGA algorithm, one can obtain a cycle of length 2^n starting with the initial state $i = 0,\ j = 0,\ k = k_0 \equiv -1 \bmod 2^n, S[r] = s_r \equiv 1 \bmod 2^n,\ \forall r \in [0, 2^n - 1], r \neq 2,$ and $S[2] = s_2 \equiv 0 \bmod 2^n$ under the conditions*

$$k_0 \equiv -\left(s_0 + 3 \cdot s_1 + \sum_{r=3}^{2^n-1} s_r\right) \bmod 2^m \text{ and}$$

$$s_2 \equiv -\left(3 \cdot s_1 + \sum_{r=3}^{2^n-1} s_r\right) \bmod 2^m.$$

Proof. One can check that the value $S[i] + S[j] = 2 \bmod 2^n$ is an invariant. Now the states are evolved as follows. In the following state sequence '+' stands for addition modulo 2^m.

$$\begin{pmatrix} i = j = 1,\ k = k_0 + s_1 \\ S[r] = s_r,\ \forall r \in [0, 2^n - 1], r \neq 2,\ S[2] = k_0 + 2s_1 \end{pmatrix}$$

$$\begin{pmatrix} i = j = 2,\ k = 2k_0 + 3s_1 \\ S[r] = s_r,\ \forall r \in [0, 2^n - 1], r \neq 2,\ S[2] = 3k_0 + 5s_1 \end{pmatrix}$$

$$\begin{pmatrix} i = j = 3,\ k = 2k_0 + 3s_1 + s_3 \\ S[r] = s_r,\ \forall r \in [0, 2^n - 1], r \neq 2,\ S[2] = 2k_0 + 3s_1 + 2s_3 \end{pmatrix}$$

$$\vdots$$

$$\begin{pmatrix} i = j = i_0,\ k = 2k_0 + 3s_1 + \sum_{r=3}^{i_0} s_r \\ S[r] = s_r,\ \forall r \in [0, 2^n - 1], r \neq 2,\ S[2] = 2k_0 + 3s_1 + \sum_{r=3}^{i_0} s_r + s_{i_0} \end{pmatrix}$$

$$\vdots$$

$$\begin{pmatrix} i = j = 0,\ k = 2k_0 + s_0 + 3s_1 + \sum_{r=3}^{2^n-1} s_r \\ S[r] = s_r,\ \forall r \in [0, 2^n - 1], r \neq 2,\ S[2] = 2k_0 + 2s_0 + 3s_1 + \sum_{r=3}^{2^n-1} s_r \end{pmatrix}$$

Thus to get a cycle of length 2^n we need to satisfy $k_0 \equiv 2k_0 + s_0 + 3s_1 + \sum_{r=3}^{2^n-1} s_r \bmod 2^m$ and $s_2 \equiv 2k_0 + 2s_0 + 3s_1 + \sum_{r=3}^{2^n-1} s_r \bmod 2^m$. From which we get

$$k_0 \equiv -\left(s_0 + 3 \cdot s_1 + \sum_{r=3}^{2^n-1} s_r\right) \bmod 2^m \text{ and}$$

$$s_2 \equiv -\left(3 \cdot s_1 + \sum_{r=3}^{2^n-1} s_r\right) \bmod 2^m.$$

□

It is possible to choose $S[r] \equiv 1 \bmod 2^n$ for all $r \in [0, 2^n - 1]$ except $r = 2$. Then one can obtain the values satisfying the conditions mentioned in Theorem 3.5. For example, in GGHN$(8, 32)$ if $s_r = 1, \quad \forall\ r \in [0, 2^n - 1]$ except $r = 2$, then we would obtain $k_0 = 2^{32} - (2^8 + 1)$ and $s_2 = 2^{32} - 2^8$.

This is the first time such short 2^n-length cycles are shown for the cipher GGHN(n, m). That is, the cycles are as short as the length of the S array itself. One may easily see that for $r \neq 2$, $S[r] \equiv 1 \bmod 2^n$, $\forall r \in [0, 2^n - 1]$. Thus, for each location of S array (except the 2nd one), we have 2^{m-n} options. Thus there are at least $(2^{m-n})^{2^n-1} = 2^{(m-n)(2^n-1)}$ initial conditions for which GGHN(n, m) will land into a short cycle equal to the S-array length. This number is however only a small fraction of the total number of possible initial states $(2^{m(2^n+1)})$ of the GGHN PRGA cipher. So the probability, that a randomly initialized GGHN state falls into a short cycle of this kind is atleast

$$p_{cycle} = \frac{(2^{m-n})^{2^n-1}}{2^{m(2^n+1)}}$$

One may also start with several other conditions and explore different kinds of short cycles.

3.3 Evolution of a Randomized variant of GGHN cipher

We have so far studied the exact GGHN cipher. It has been clearly shown that the cipher has weaknesses due to the short cycles. Now to have a view of the evolution of similar kinds of ciphers, we would like to present a theoretical model where the indices are chosen independently and uniformly at random from $[0, N - 1]$. In this regard, we make the assumption that

$$j, (S[i] + S[j]) \bmod N \text{ and } i$$

are chosen independently and uniformly at random from $[0, \ldots, N - 1]$ and we call them a, b, c respectively. In fact, it is clear from the design of the GGHN PRGA [68] that the authors tried to simulate $j, (S[i]+S[j]) \bmod N$ as pseudorandom indices and we consider the same here in the theoretical model. However, the index i has been incremented by one in each step of GGHN [68] and the only difference in our randomized model is that the index i is also taken as a random index. We here consider the evolution from the situation when the KSA is completed and we assume that the values of the array S as

well as k are chosen uniformly at random from the set of m-bit integers. Let us now describe the algorithm which we will call RAND-GGHN-PRGA(n, m).

while *Keystream is generated* **do**
  Select a, b, c uniformly at random from $[0, N-1]$;
  $k = (k + S[a]) \bmod M$;
  $z = (S[b] + k) \bmod M$;
  $S[b] = (k + S[c]) \bmod M$;
end

Algorithm 3.5: RAND-GGHN-PRGA(n, m)

We will first show that there exists a function $f(N)$ such that it is expected that in $f(N)$ steps the array S of RAND-GGHN-PRGA(n, m) will be all even. For this we can work with the following algorithm considering S as a bit-array and k as a single bit. The bits of the S array as well as k are initially chosen uniformly at random from $\{0, 1\}$. Since we are working with the least significant bit only, it is enough to use XOR (i.e., GF(2) addition) as the least significant bit is same for both XOR and modulo M addition. We call the algorithm BIT-RAND-GGHN-PRGA$(n, 1)$.

while *the loop is required to be run* **do**
  Select a, b, c uniformly at random from $[0, N-1]$;
1 $k = (k \oplus S[a])$;
2 $S[b] = (k \oplus S[c])$;
end

Algorithm 3.6: BIT-RAND-GGHN-PRGA$(n, 1)$

We want to find the expected number of iterations after which all the elements in S as well as k become zero. We use results related to Markovian process and one may either refer to either Section 2.4 or [70, Chapter 11] for more details of this technique.

Theorem 3.6. *Following BIT-RAND-GGHN-PRGA$(n, 1)$, let S be a binary array of length N. The elements of S as well as k are filled independently and uniformly at random from $\{0, 1\}$. Then there is a function $f(N)$ such that it is expected that all elements of S as well as k will be zero in $f(N)$ steps.*

Proof. Consider that q denotes the number of 1's in S. Hence it should be sufficient to analyze the Markov chain with state $(q, k) \in \{0, N\} \times \{0, 1\}$. At the t-th stage, denote q_t as the number of 1's in S and k_t as the value of k. The transitions of this chain are

given by

$$k_{t+1} = \begin{cases} 1 \oplus k_t, & \text{with probability } \frac{q_t}{N}, \\ k_t, & \text{otherwise} \end{cases} \tag{3.1}$$

and

$$q_{t+1} = \begin{cases} q_t + 1, & \text{with probability } \frac{q_t}{N}\left(1 - \frac{q_t}{N}\right) \text{ if } k_{t+1} = 0, \\ q_t - 1, & \text{with probability } \frac{q_t}{N}\left(1 - \frac{q_t}{N}\right) \text{ if } k_{t+1} = 0, \\ q_t + 1, & \text{with probability } \left(1 - \frac{q_t}{N}\right)^2 \text{ if } k_{t+1} = 1, \\ q_t - 1, & \text{with probability } \left(\frac{q_t}{N}\right)^2 \text{ if } k_{t+1} = 1, \\ q_t & \text{otherwise.} \end{cases} \tag{3.2}$$

If $q = 0$ and $k = 0$, then the process terminates as it will remain in the same state. We obtain the expected number of steps t to hit $(q_t, k_t) = (0, 0)$ given any initial state.

Note that q follows $Binomial(N, \frac{1}{2})$ and k is uniform in $\{0, 1\}$. It is convenient to write the transition matrix M as $M_f = M_q M_k$, where M_k (respectively M_q) gives the transition probability for Step 1 (respectively Step 2) of Algorithm 3.6. It is easy to see that both M_k and M_q are $(2N + 1) \times (2N + 1)$ matrices.

Now we describe the matrices M_k, M_q following (3.1) and (3.2). Let j be the column index from 1 to $2N+1$. Let $v_j = \lceil \frac{j-1}{2} \rceil$ (where $\lceil \cdot \rceil$ denotes the greatest integer function). The diagonal entries of M_k are $1 - \frac{v_j}{N}$ for the column j. The off-diagonal entries (the matrix entries adjoining the diagonal) of M_k are $\frac{v_j}{N}$ with one entry for each column, i.e. the entry appears just below the diagonal if j is even and just above if j is odd (see Example 3.1). Here the ordering of (q_t, k_t) that we follow is given as $(0, 1)$, $(1, 0)$, $(1, 1)$, $(2, 0)$, $(2, 1)$, …, $(n, 1)$.

The diagonal elements of M_q are $1 - \frac{2v_j}{N}\left(1 - \frac{v_j}{N}\right)$ for j-th column with even j and $1 - \left(\frac{v_j}{N}\right)^2 - \left(1 - \frac{v_j}{N}\right)^2$ for odd j. The other non-zero elements of M_q appear exactly two rows above and two rows below each diagonal element. These values are equal to $\frac{v_j}{N}\left(1 - \frac{v_j}{N}\right)$ for even j in both upper and lower rows. For odd j the values are $\left(\frac{v_j}{N}\right)^2$ and $\left(1 - \frac{v_j}{N}\right)^2$ for upper and lower rows respectively.

Note that, M_k is a (right) stochastic matrix, i.e., each of its column values sum to one, while M_q (and hence M_f also) is stochastic except for second column, whose sum is one minus the probability of transition to $(q = 0, k = 0)$ state.

To calculate the expected survival time of the process, we need the fundamental matrix $F = (I - M_f)^{-1}$. The element F_{ij} gives the expected number of times the process visits state i given the process started from the state j. Summing over the columns of F, we

get the expected total survival time for each initial state (except for which $q = 0, k = 0$, as the survival time is 0 for that case).

Now consider a vector E of length $2N + 1$. Each element of the vector E is the column sum of the matrix F, i.e., $E = \mathbf{1}^T F$. We interpret E as a vector indexed by 1 to $2N + 1$ and the i^{th} entry of E is the just the expected time taken to reach the all zero state given that the chain started initially from the i^{th} state. Also consider the initial state distribution vector v of $2N + 1$ length given by $v[1] = \frac{1}{2^{N+1}}$ and for $q \geq 1$, $v[2q] = v[2q+1] = \binom{N}{q}\frac{1}{2^{N+1}}$. Then $f(N) = E \cdot v$ gives the expected survival time which we denote by $f(N)$. □

Now we will describe the approach of Theorem 3.6 in detail with the following example.

Example 3.1. *Let us consider the case $N = 4$. The entries of M_k, M_q are the transition probabilities of Steps 1 and 2 of Algorithm 3.6 respectively. Here the ordering of the states as $(0,1)$, $(1,0)$, $(1,1)$, $(2,0)$, $(2,1)$, $(3,0)$, $(3,1)$, $(4,0)$ and $(4,1)$. The transition matrices will be as follows.*

$$M_k = \frac{1}{4}\begin{pmatrix} 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 \end{pmatrix}, M_q = \frac{1}{16}\begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 16 & 0 & 6 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 8 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 9 & 0 & 8 & 0 & 9 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 6 & 0 & 16 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 16 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Hence the final transition matrix $M_q M_k$ will be

$$M_f = \frac{1}{64}\begin{pmatrix} 0 & 1 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 30 & 10 & 8 & 8 & 0 & 0 & 0 & 0 \\ 64 & 6 & 18 & 8 & 8 & 0 & 0 & 0 & 0 \\ 0 & 9 & 3 & 16 & 16 & 3 & 9 & 0 & 0 \\ 0 & 9 & 27 & 16 & 16 & 27 & 9 & 0 & 0 \\ 0 & 0 & 0 & 8 & 8 & 10 & 30 & 0 & 0 \\ 0 & 0 & 0 & 8 & 8 & 18 & 6 & 64 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 9 & 0 & 64 \\ 0 & 0 & 0 & 0 & 0 & 3 & 1 & 0 & 0 \end{pmatrix}.$$

The fundamental matrix (elements written upto two decimal places) will be

$$F = \begin{pmatrix} 1.33 & 0.25 & 0.33 & 0.29 & 0.29 & 0.29 & 0.29 & 0.29 & 0.29 \\ 5.33 & 5.97 & 5.33 & 5.65 & 5.65 & 5.65 & 5.65 & 5.65 & 5.65 \\ 5.33 & 3.41 & 5.33 & 4.37 & 4.37 & 4.37 & 4.37 & 4.37 & 4.37 \\ 6.00 & 5.10 & 6.00 & 7.80 & 6.80 & 6.99 & 7.11 & 7.11 & 7.11 \\ 10.00 & 8.02 & 10.00 & 10.76 & 11.76 & 11.57 & 11.45 & 11.45 & 11.45 \\ 5.33 & 4.37 & 5.33 & 6.19 & 6.19 & 7.83 & 7.21 & 7.21 & 7.21 \\ 5.33 & 4.37 & 5.33 & 6.19 & 6.19 & 7.01 & 8.03 & 8.03 & 8.03 \\ 1.33 & 1.09 & 1.33 & 1.55 & 1.55 & 1.83 & 1.93 & 2.93 & 2.93 \\ 0.33 & 0.27 & 0.33 & 0.39 & 0.39 & 0.48 & 0.46 & 0.46 & 1.46 \end{pmatrix}.$$

So E, whose entries are sum of columns, will be

$$E = \mathbf{1}^T F = \begin{bmatrix} 40.33 & 32.87 & 39.33 & 43.19 & 43.19 & 46.02 & 46.52 & 47.52 & 48.52 \end{bmatrix}$$

Here the initial state distribution vector $v = \frac{1}{32}(1, 4, 4, 6, 6, 4, 4, 1, 1)$. Hence $E \cdot v = 41.05$ gives the expected number of steps where all elements of S and also k are zero.

For BIT-RAND-GGHN-PRGA$(n, 1)$, in Table 3.1, we present theoretical bounds and experimental observations on the expected number of iterations required to get all entries of S as well as k to be zero for different values of array length N. The theoretical exercises related to formation of matrices were done with the help of SAGE [129]. For numerical experiments, the programs were written in C language (the random number generator of "GNU project C compiler in Linux environment Ubuntu 11.04" was used to get a, b, c) and the average is taken over 10^5 runs. For the cases $N \geq 32$, we could not perform the experiments because of the long execution time and hence we left the corresponding entries blank. One may note that we get quite close results in theory and experiment.

N	$f(N)$	Experiment
4	41.05	41.02
8	280.49	279.89
12	1463.27	1469.15
16	7118.88	7111.03
20	33836.28	33433.15
32	3423401.56	-
64	619282894484.52	-
128	149191364194358609155 74.98	-
256	623018928847357392507174212136531506445230 9.75	-

TABLE 3.1: Theoretical bounds and experimental values of $f(N)$ for different values of $N = 2^n$ in BIT-RAND-GGHN-PRGA$(n, 1)$.

As $623018928847357392507174212136531506445230 9 \approx 2^{142.16}$, following Table 3.1, we can say that when length of S is 256, all the elements of S as well as k become zero within expected $2^{142.16}$ steps for $N = 256$. Note that, for a properly designed cipher one can expect that it will reach the all even state in $2^{N+1} = 2^{257}$ iterations as there are 256 LSBs of the 256 locations in the state array S and 1 LSB of k. In the randomized model, the expected time to get into such an even state requires much less number of iterations than 2^{257}.

Now consider Algorithm 3.5 and here we have the following result.

Lemma 3.7. *In the algorithm RAND-GGHN-PRGA(n, m), all the elements of S as well as the integer k are expected to be zero in $m \cdot f(N)$ iterations.*

Proof. From Theorem 3.6, for Algorithm 3.6, it is expected that in $f(N)$ iterations all entries of S as well as k will be zero. Now consider Algorithm 3.5. When the Least Significant Bits (LSBs) of all the elements of S and k become zero, it will remain zero for all further steps. We know that, the LSBs of the elements of S and k are expected to be be zero after $f(N)$ steps. Then the operations will not affect the next significant bit and for those bits of S and k, one can expect to get zero in the next $f(N)$ iterations again. Proceeding in this way, after $m \cdot f(N)$ steps all elements S and as well as the integer k will be zero. □

For RAND-GGHN-PRGA(n, m) (in Algorithm 3.5), where $n = 8$ and $m = 32$ (i.e., $N = 2^8$ and $M = 2^{32}$), one can expect after $32 \times 2^{142.16} = 2^{147.16}$ iterations, all entries of S and k will become zero. For an ideally designed cipher one can expect that it will reach the all zero state in expected $2^{(N+1)m} = 2^{8224}$ iterations as there are $256 \cdot 32$ bits of the 256 length state array S and an additional 32 bits of k. In this randomized model, the expected time to get into such an all zero state requires much less number of iterations than 2^{8224}.

3.3.1 Towards estimating the actual GGHN PRGA

It is interesting to see how similar or dissimilar the evolution of RAND-GGHN-PRGA is when compared to the actual PRGA of GGHN. One striking dissimilarity, already noted in Lemma 3.1, is that after the actual PRGA of GGHN the state array S evolves to the all even state and the cells of the state array marked by odd indices do not undergo any further change. Hence the PRGA of actual GGHN does not evolve to the all zero state (unless the odd-indexed places of S had already become zero, which is extremely less probable). Hence, it is not possible to do a similar analysis for the exact cipher.

However one can analyze and compare the evolution of several randomized variants of the original PRGA. In this regard, once again we would like to remind that the variables

$$a, b, c \text{ of the RAND-PRGA-GGHN algorithm correspond to } j, S[i] + S[j], i$$

of actual GGHN PRGA. In the RAND-PRGA-GGHN algorithm we have made the assumption that the variables a, b, c are chosen uniformly at random. One can come up with two other variants of this cipher RAND-GGHN-PRGA$'$ and RAND-GGHN-PRGA$''$ given below which are closer to the actual GGHN PRGA. In RAND-GGHN-PRGA$'$, the variable c (corresponding to i of the original GGHN PRGA) is not selected randomly but incremented by 1 after each iteration as in the original GGHN PRGA. In RAND-GGHN-PRGA$''$, both c and a (corresponding to i and j of the original GGHN PRGA) are not chosen randomly but incremented as in the original cipher. If the variable b too were to be not chosen randomly and incremented as in the original cipher, then the resulting PRGA would be exactly same as the original GGHN PRGA, but as already explained the analysis of a such a cipher can not be performed due to restrictions of the model. As it turns out RAND-GGHN-PRGA$''$ is the model which most closely resembles the original cipher.

$c = 0$;
while *Keystream is generated* **do**
  Select a, b uniformly at random from $[0, N-1]$;
  $c = (c + 1) \bmod N$;
  $k = (k + S[a]) \bmod M$;
  $z = (S[b] + k) \bmod M$;
  $S[b] = (k + S[c]) \bmod M$;
end

Algorithm 3.7: RAND-GGHN-PRGA$'(n, m)$

$c = a = 0$;
while *Keystream is generated* **do**
  Select b uniformly at random from $[0, N-1]$;
  $c = (c + 1) \bmod N$;
  $a = (a + S[c]) \bmod N$;
  $k = (k + S[a]) \bmod M$;
  $z = (S[b] + k) \bmod M$;
  $S[b] = (k + S[c]) \bmod M$;
end

Algorithm 3.8: RAND-GGHN-PRGA$''(n, m)$

To have a view of how the theoretical result of Lemma 3.7 matches with practice we consider the case of $n = 4, m = 16$ for RAND-GGHN-PRGA(n, m). From Table 3.2 we note that the number steps for one additional set of bits to be zero is almost equal for RAND-GGHN-PRGA. This corresponds to Lemma 3.7. In the experiments for RAND-GGHN-PRGA(n, m), we have observed that when the t-th LSBs of each location of the S array and k become all zero for the first time, at that point the $(t + u)$-th LSBs look random for $u > 0$.

We also consider the other two randomized variants of the cipher and note the number of iterations each variant of the cipher takes to reach the all zero state. The results are average of 10^5 independent runs. The LSBs of all the entries of S become zero in expected 7130.88 steps RAND-GGHN-PRGA whereas it takes 10906.15 steps in RAND-GGHN-PRGA$'$ and 9089.36 steps in RAND-GGHN-PRGA$''$; the next significant bit of those become zero in expected 14235.94 steps in RAND-GGHN-PRGA and so on. From experiments, we note that RAND-GGHN-PRGA$'$ evolves to all zero state after more number of steps than RAND-GGHN-PRGA. However, RAND-GGHN-PRGA$''$, the closest model to actual GGHN PRGA, evolves to all zero state much faster than RAND-GGHN-PRGA.

t	RAND-GGHN-PRGA	RAND-GGHN-PRGA$'$	RAND-GGHN-PRGA$''$
0	7130.88	10906.15	9089.36
1	14235.94	21826.94	14517.11
2	21352.26	32824.21	17271.46
3	28483.93	43716.31	18821.33
4	35613.17	54635.29	20092.96
5	42710.16	65527.85	21320.74
6	49791.33	76423.01	22544.97
7	56895.35	87361.42	23766.03
8	63981.16	98236.69	24985.67
9	71040.10	109174.21	26199.41
10	78158.61	120082.29	27418.34
11	85253.82	131058.35	28637.54
12	92328.69	141963.34	29855.91
13	99448.96	152865.53	31075.49
14	106477.86	163751.12	32296.63
15	113568.96	174659.31	33510.00

TABLE 3.2: Average number of steps required for t LSBs to be zero for all the elements of S as well as the integer k. The algorithms considered are RAND-GGHN-PRGA$(4, 16)$, RAND-GGHN-PRGA$'(4, 16)$ and RAND-GGHN-PRGA$''(4, 16)$.

3.4 The RC4+ stream cipher

The RC4+ cipher was proposed by Maitra and Paul at Indocrypt 2008 [100]. The authors had claimed that RC4+ while marginally slower than RC4 in software, would resist all the known distinguishing and state recovery attacks against RC4. To the best of our knowledge, no cryptanalytic advance has been made against this cipher.

The physical structure of RC4+ is the same as that of RC4. It consists of a permutation S of $N = 256$ elements from the integer ring $\mathbb{Z}_{256}$. It also uses two index pointers i, j of size 1 byte each. As in RC4, during the Key Scheduling Algorithm(KSA), S is initialized to the identity permutation and mixed using a Secret Key K of size l bytes (typically $l = 16$). Then, the array S is further scrambled using an l byte IV, after which another layer of zig-zag scrambling is performed. The exact details of the KSA are given in Table 3.3. Note that all addition operations are performed in $\mathbb{Z}_{256}$, and $\oplus$ denotes bitwise-XOR. The array V used in the KSA is defined as

$$V[i] = \begin{cases} IV[127 - i], & \text{if } 128 - l \leq i \leq 127, \\ IV[i - 128], & \text{if } 128 \leq i \leq 127 + l \\ 0, & \text{otherwise.} \end{cases}$$

Input: Secret Key K, Initial Vector IV
Output: Permutation S on $\mathbb{Z}_{256}$

---

for $i = 0$ *to* 255 **do**
  $S[i] = i$;
end
$j \leftarrow 0$
`Key Loading`
for $i = 0$ *to* 255 **do**
  $j \leftarrow j + S[i] + K[i \bmod l]$;
  Swap $S[i], S[j]$;
end
`IV Loading`
for $i = 127$ *to* 0 **do**
  $j \leftarrow$
  $(j + S[i]) \oplus (K[i \bmod l] + V[i])$;
  Swap $S[i], S[j]$;
end
for $i = 128$ *to* 255 **do**
  $j \leftarrow$
  $(j + S[i]) \oplus (K[i \bmod l] + V[i])$;
  Swap $S[i], S[j]$;
end

`Zig-Zag Scrambling`

for $y = 0$ *to* 255 **do**
  if $y \equiv 0 \bmod 2$ **then**
    $i = \frac{y}{2}$;
  end
  else
    $i = 128 - \frac{y+1}{2}$;
  end
  $j \leftarrow j + S[i] + K[i \bmod l]$;
  Swap $S[i], S[j]$;
end

TABLE 3.3: KSA routine for RC4+

The PRGA routine of RC4+ deviates slightly from the simplistic structure of RC4. In order to protect against the well known second output byte bias of Mantin-Shamir [102] and the permutation recovery attack of Maximov and Khovratovich [105], the designers propose to make the output keystream byte functions of a few other locations of the permutation array S. The details of the PRGA routine are given in Table 3.4. Note that $\gg$ and $\ll$ denote right and left bitwise shifts respectively.

Input: Permutation S on $\mathbb{Z}_{256}$
Output: Output Keystream bytes Z

$i = j = 0$;
while *Keystream is required* **do**
  $i \leftarrow i + 1$;
  $j \leftarrow j + S[i]$;
  Swap $S[i], S[j]$;

  $t \leftarrow S[i] + S[j]$;
  $t' \leftarrow (S[i \gg 3 \oplus j \ll 5] + S[i \ll 5 \oplus j \gg 3]) \oplus$ `0xAA`;
  $t'' \leftarrow j + S[j]$;

  $Z_i = (S[t] + S[t']) \oplus S[t'']$;
end

TABLE 3.4: PRGA routine for RC4+

3.4.1 Our Results

In Section 3.5, we will show that the first output byte produced by RC4+ is negatively biased towards 1. In fact we will prove that the probability that the first output byte is equal to 1 is around $\frac{1}{N} - \frac{1}{2N^2}$, where $N = 256$ is the number of elements of the array S used in the design. Using this observation we will mount a distinguishing attack against RC4+ that requires around 2^{26} output keystreams produced by **(a)** Secret Keys chosen uniformly at random or **(b)** any fixed Secret Key used with IVs chosen uniformly at random. In Section 3.6, we revisit the Differential Fault Attack on RC4 proposed by Biham et. al. in FSE 2005 [38]. We explore the possibility of mounting such a fault attack on RC4+. We will show that by injecting around $2^{17.2}$ faults, it is possible to recover the internal state of the cipher efficiently.

3.5 Distinguishing Attack on RC4+

In this section we will prove that the first output byte Z_1 (when the value of the index $i = 1$) is negatively biased towards 1. We will prove that $\mathsf{Pr}(Z_1 = 1) = \frac{1}{N} - \frac{1}{2N^2}$. The

initial state of the RC4+ PRGA is denoted by S_0.

Lemma 3.8. *Let S_0 be a random permutation on $\{0, 1, 2, \ldots, 255\}$. If $S_0[1] = 1$ and $S_0[2]$ is even, then Z_1 can never take the value 1.*

Proof. We refer to the PRGA algorithm in Table 3.4. Initially $i = j = 0$. After the increment operations the new values of i, j are as follows: $i = 0 + 1 = 1$ and $j = 0 + S_0[i] = 0 + S_0[1] = 1$. Since $i = j$ even after the increment operations, the subsequent swap operation does not bring about any change in the array S_0. Thereafter the values of t, t', t'' are calculated as follows:

$$t = S_0[i] + S_0[j] = 2 \cdot S_0[1] = 2.$$

$$\begin{aligned} t' &= (S_0[i \gg 3 \oplus j \ll 5] + S_0[i \ll 5 \oplus j \gg 3]) \oplus \texttt{0xAA} \\ &= (S_0[1 \gg 3 \oplus 1 \ll 5] + S_0[1 \ll 5 \oplus 1 \gg 3]) \oplus \texttt{0xAA} \\ &= (2 \cdot S_0[32]) \oplus \texttt{0xAA} \end{aligned}$$

Finally $t'' = j + S_0[j] = 1 + S_0[1] = 1 + 1 = 2$. Therefore we have $Z_1 = (S_0[2] + S_0[t']) \oplus S_0[2]$. Suppose that $Z_1 = 1$, then we will have

$$(S_0[2] + S_0[t']) \oplus S_0[2] = 1 \quad \Rightarrow \quad S_0[2] + S_0[t'] = S_0[2] \oplus 1$$

Since $S_0[2]$ is even, we must have $S_0[2] \oplus 1 = S_0[2] + 1$. Hence the previous equation reduces to:

$$S_0[2] + S_0[t'] = S_0[2] + 1 \quad \Rightarrow \quad S_0[t'] = 1$$

S_0 is a permutation and hence injective. So $S_0[t'] = S_0[1] = 1$ can only imply that $t' = 1$. Thus we have

$$(2 \cdot S_0[32]) \oplus \texttt{0xAA} = 1$$

The LHS of the above equation is clearly an even number whereas the RHS is odd. This gives rise to a contradiction, and therefore $Z_1 = 1$ can clearly not hold.

□

Corollary 3.9. *The above Lemma would still hold if any even pad instead of* `0xAA` *were used in the design.*

Theorem 3.10. *Let S_0 be a random permutation on $\{0, 1, 2, \ldots, 255\}$. The probability that $Z_1 = 1$ is given by the equation $\mathsf{Pr}(Z_1 = 1) = \frac{1}{N} - \frac{1}{2N^2}$ (where $N = 256$).*

Proof. Let E denote the event: "$S_0[1] = 1$ and $S_0[2]$ is even". Then it is clear that $\mathsf{Pr}[\mathsf{E}] = \frac{\frac{N}{2} \cdot (N-2)!}{N!} \approx \frac{1}{2N}$. From Lemma 3.8, we have $\mathsf{Pr}[Z_1 = 1 | \mathsf{E}] = 0$. By standard

randomness assumptions, we have $\mathsf{Pr}[Z_1 = 1|\mathsf{E}^c] = \frac{1}{N}$ (this has been verified by extensive computer experiments with 2^{20} random keys). Therefore we have

$$\begin{aligned}\mathsf{Pr}[Z_1 = 1] &= \mathsf{Pr}[Z_1 = 1|\mathsf{E}] \cdot \mathsf{Pr}[\mathsf{E}] + \mathsf{Pr}[Z_1 = 1|\mathsf{E}^c] \cdot \mathsf{Pr}[\mathsf{E}^c] \\ &= 0 \cdot \frac{1}{2N} + \frac{1}{N} \cdot \left(1 - \frac{1}{2N}\right) = \frac{1}{N} - \frac{1}{2N^2}.\end{aligned}$$

□

3.5.1 Distinguishing RC4+ from Random Sources

Let X be the probability distribution of Z_1 in an ideal random stream, and let Y be the probability distribution of Z_1 in streams produced by RC4+ for randomly chosen keys. Let the event e denote $Z_1 = 1$, which occurs with probability of $\frac{1}{N}$ in X and $\frac{1}{N} - \frac{1}{2N^2} = \frac{1}{N} \cdot \left(1 - \frac{1}{2N}\right)$ in Y. By using the Theorem 2.18 with $p_0 = \frac{1}{N}$ and $q_0 = -\frac{1}{2N}$, we can conclude that we need about $\frac{1}{p_0 q_0^2} = 4 \cdot N^3 = 2^{26}$ output samples to reliably distinguish the two distributions.

3.5.2 Experimental Results

By performing extensive computer simulations with **(a)** one billion random keys, and **(b)** a fixed key with one billion random IVs, the probability $\mathsf{Pr}[Z_1 = 1]$ was found to be around $2^{-8} - 2^{-17.03}$. This is consistent with the theoretical value of $\frac{1}{N} - \frac{1}{2N^2}$ proven in Theorem 3.10.

3.6 Differential Fault Analysis of RC4+

In [38], a Differential Fault Attack and an Impossible Fault Attack of the RC4 stream cipher was proposed. The Impossible Fault Attack uses random faults on the i or j indices of the RC4 PRGA to drive the cipher into a special state called Finney state [60]. The Finney states are called impossible states because they can not occur under normal mode of operation of RC4 and hence the unusual name of the attack. By injecting around 2^{16} faults on either the i or j register, the cipher is expected to enter a Finney State. From observing the faulty output bytes of RC4 it is possible to assess if the cipher has indeed entered a Finney State. Since any Finney state cycles back after $255 \cdot 256 = 65280$ iterations of the cipher, the attacker selects one of the interleaved cycles in the output stream as the internal state. Once the internal state is obtained at some point in time, it is possible to backtrack and find the initial state at the beginning

of the PRGA. Note that, since the PRGA update operations of RC4 and RC4+ are exactly similar, an impossible fault attack on RC4+ may also be carried out using the same techniques outlined in [38].

Applying the Differential Fault Attack (DFA) of [38] to RC4+, however, is not so straightforward. Before proceeding, we note that the PRGA of RC4 is exactly the same as that of RC4+, the only difference being that RC4 outputs $S[t]$ instead of $(S[t] + S[t']) \oplus S[t'']$. We will state in brief the DFA algorithm in [38].

A. Perform a key setup (KSA) with the unknown key and run the RC4 PRGA for around 1000 iterations, and record the output stream Z_i, $(1 \leq i \leq 1000)$ for later analysis.

B. Process the following 256 times with l being set from 0 to 255, giving 256 faulty output streams

1. Restart the cipher and perform a key setup with the same unknown key.
2. Make a fault in $S[l]$.
3. Run the RC4 PRGA 30 steps, and record the faulty output stream $Z_i^1[l]$ for later analysis.

C. Repeat Step **B** with fault injection in k^{th} $(2 \leq k \leq 1000)$ PRGA iteration instead of just after key setup. Record the faulty keystream sequence $Z_i^k[l]$ in each case (thus $Z_i^k[l]$ is the faulty i^{th} keystream byte when the location $S[l]$ has been faulted at PRGA round k).

For any i, the output byte Z_i is a function of just 3 locations of the S array: $i,\ j,\ S[i] + S[j]$. So evidently, the output byte of all the $Z_i^i[l]$'s (note $Z_i^i[l]$ is the first output byte obtained after faulting $S[l]$ at round i), except for three of them, are the same as in the faultless output byte Z_i. The identification of these three streams leak the values of $i,\ j,\ S[i] + S[j]$, but not which is which. Of course, the value of i is always known, thus the only task is to identify which is j and which is $S[i] + S[j]$. After the values of $j,\ S[i] + S[j]$ are obtained for sufficiently many PRGA rounds i, a cascade guessing technique is employed in [38] to eliminate incorrect guesses of j from $j,\ S[i] + S[j]$ and thereafter reconstruct the initial permutation S. For more details, we refer the reader to [38].

However in RC4+, the output byte is a function of 7 locations of the S array: $i,\ j,\ S[i] + S[j],\ j + S[j],\ i \gg 3 \oplus j \ll 5,\ i \ll 5 \oplus j \gg 3, (S[i \gg 3 \oplus j \ll 5] + S[i \ll 5 \oplus j \gg 3]) \oplus$ `0xAA`. Therefore repeating the above procedure in the case of RC4+ would leak a maximum of 7 indices in each round, of which only the value of i is known with certainty. The

values of the other 6 indices can not be assigned with certainty. Thus, on the face of it, performing DFA on RC4+ seems to be more difficult than RC4. However as we will see in Section 3.6.1, this is not so.

3.6.1 Inferring the values of j in each round

As we have seen, performing steps **A, B, C** for RC4+, leaks the values of 6 indices. Although the attacker knows that these are the values of the indices $j,\ S[i]+S[j],\ j+S[j],\ i \gg 3 \oplus j \ll 5,\ i \ll 5 \oplus j \gg 3, (S[i \gg 3 \oplus j \ll 5] + S[i \ll 5 \oplus j \gg 3]) \oplus$ `0xAA`, he is unable to ascertain which of these 6 values correspond to which index. We will later see in Section 3.6.2, that if the attacker can correctly establish the value of only the index j, it will be enough to reconstruct the permutation S at the beginning of the PRGA. Before we outline our strategy to find the value of j, we will look at a result that will help us build the attack.

Lemma 3.11. *For any value of i, consider two values $j_1,\ j_2$. If $i \gg 3 \oplus j_1 \ll 5 = i \gg 3 \oplus j_2 \ll 5$, and $i \ll 5 \oplus j_1 \gg 3 = i \ll 5 \oplus j_2 \gg 3$, then $j_1 = j_2$.*

Proof. Rearranging the terms in both equations we get $(j_1 \oplus j_2) \ll 5 = 0 = (j_1 \oplus j_2) \gg 3$. Then, $j_1 \oplus j_2 = 0$ is the only solution to the equation and so $j_1 = j_2$. □

3.6.1.1 Ascertaining j

For any round i, the attacker has with him 6 values corresponding to the indices $j,\ S[i]+S[j],\ j+S[j],\ i \gg 3 \oplus j \ll 5,\ i \ll 5 \oplus j \gg 3, (S[i \gg 3 \oplus j \ll 5] + S[i \ll 5 \oplus j \gg 3]) \oplus$ `0xAA`. Let us call these six values $k_1, k_2, \ldots, k_6$. He of course does not know the correspondence between the $k_1, \ldots, k_6$ and the indices. Without loss of generality let k_1 be the correct value of j. Then evaluating the functions $i \gg 3 \oplus k_1 \ll 5$ and $i \ll 5 \oplus k_1 \gg 3$ will lead to two of the values in $k_2, k_3, \ldots, k_6$ i.e. those corresponding to $i \gg 3 \oplus j \ll 5$ and $i \ll 5 \oplus j \gg 3$. The probability that any other $k_a,\ 2 \leq a \leq 6$ will on evaluating $i \gg 3 \oplus k_a \ll 5$ and $i \ll 5 \oplus k_a \gg 3$ will lead to two elements of $\{k_1, k_2, \ldots, k_6\}$ is very low. Therefore given any i the strategy will be as follows

- For $a = 1$ to 6

    1. Compute $M_a = i \gg 3 \oplus k_a \ll 5$ and $N_a = i \ll 5 \oplus k_a \gg 3$.
    2. If $M_a, N_a \in \{k_1, k_2, k_3, k_4, k_5, k_6\}$ then $j = k_a$.

The strategy of the attacker will be to determine the values of j for around 602 consecutive values of i. As will be seen in Section 3.6.2, this will suffice to reconstruct the permutation S at the beginning of the PRGA.

3.6.1.2 Error Analysis

Lemma 3.11 guarantees that any value k_a different j, when used to calculate M_a, N_a will result in values not equal to both $i \gg 3 \oplus j \ll 5$ and $i \ll 5 \oplus j \gg 3$. Therefore, a confusion will only occur when some value $k_a \neq j$ on evaluating $i \gg 3 \oplus k_a \ll 5$ and $i \ll 5 \oplus k_a \gg 3$ also leads to two elements of $\{k_1, k_2, \ldots, k_6\}$ (which are not equal to $i \gg 3 \oplus j \ll 5$ and $i \ll 5 \oplus j \gg 3$). In such an event the attacker must guess one from the multiple values of j extracted by the algorithm. Experiments with 2^{20} random keys show that in the first 602 rounds there are around 5 to 6 confusions on average, and each confusion usually gives no more than 2 values of j to choose from. The attacker can simply guess the values of j during these rounds and use it in the algorithm for state recovery that will be discussed in the next subsection.

3.6.1.3 Fault Requirement

As we will see in the next subsection, around 602 values of j are required to reconstruct S. Since each round requires 256 faults, the total fault requirement is around $602 \times 256 \approx 2^{17.23}$.

3.6.2 Reconstructing the permutation S

We will now present the Algorithm 3.9 that will be used to reconstruct the state S. The technique used here is similar to the algorithm presented in [51]. The algorithm works under the principle that if j_1, j_2 are the values of j in two successive PRGA rounds then the the value of $S[i_1]$ is given as $j_2 - j_1$.

We assume that the algorithm starts from PRGA round t armed with M values of j in consecutive PRGA rounds. First, a two dimensional array *acc* is used, whose r-th row contains the triplet $(i_r,\ j_r,\ z_r)$. After each subsequent round $t + r$, the algorithm reverts to the initial round t and in the process uses new entries to check if the array *guess* (which is the temporary array used to guess the state S) can be populated further. Thereafter the algorithm again performs a *forward pass* up to the round $t + r + 1$ to further populate the array *guess* as much as possible. The strategy is formally presented in Algorithm 3.9.

Input: $(i_t, j_t), \{(i_{t+r}, j_{t+r}, z_{t+r} : r = 1, \ldots, M-1)\}$.
Output: Permutation array S_{t+m} for some $m \in [0, M-1]$.

$numKnown \leftarrow 0$;
$m \leftarrow 0$;
for u *from* 0 *to* $N-1$ **do**
  $guess[u] \leftarrow EMPTY$;
end
$acc[0][0] \leftarrow i_t$;
$acc[0][1] \leftarrow j_t$;
for u *from* 1 *to* $M-1$ **do**
  $acc[u][0] \leftarrow i_{t+u}$;
  $acc[u][1] \leftarrow j_{t+u}$;
  $acc[u][2] \leftarrow z_{t+u}$;
end
repeat
  $i_{t+m+1} \leftarrow acc[t+m+1][0]$, $j_{t+m+1} \leftarrow acc[t+m+1][1]$,
  $z_{t+m+1} \leftarrow acc[t+m+1][2]$;
  if $guess[i_{t+m+1}] = EMPTY$ **then**
    $guess[i_{t+m+1}] \leftarrow j_{t+m+1} - j_{t+m}$;
  end
  $backtrack(t+m, t)$;
  $processForward(t, t+m+1)$;
  $m \leftarrow m+1$;
  $numKnown \leftarrow$ Number of non-empty entries in the array $guess$;
until $numKnown = N-1$ *OR* $m = M-1$;
if $numKnown = N-1$ **then**
  Fill the remaining single EMPTY location of the array $guess$;
  for u *from* 0 *to* $N-1$ **do**
    $S_{t+m}[u] \leftarrow guess[u]$;
  end
end

Algorithm 3.9: The algorithm for state recovery with backward and forward passes.

Algorithm 3.9 uses two subroutines. The subroutine $backtrack(r, t)$ presented in Algorithm 3.10 performs a backward pass, tracing all state information back from the current round r to a previous round $t < r$. The subroutine $processForward(r, t)$, presented in Algorithm 3.11 evolves the state information in the forward direction from a past round r to the current round $t > r$. Note that Algorithm 3.9 returns the array S_{t+m} (the value of S at PRGA round $t+m$) where m is the minimal value for which S_{t+m} can be fully constructed. Thereafter the value of S at any previous round can be easily calculated as the state update of RC4+, like RC4, is one-one and invertible.

Experimental Results. We present some experimental evidences. Experimental result showing the average number of bytes recovered (over 100 random simulations) against the number of rounds used is shown in Table 3.5. It shows that around 602 consecutive

Subroutine $backtrack(r, t)$

repeat
  $i_r \leftarrow acc[r][0]$;
  $j_r \leftarrow acc[r][1]$;
  $swap(guess[i_r], guess[j_r])$;
  $r \leftarrow r - 1$;
until $r = t$;

Algorithm 3.10: Subroutine *backtrack*

Subroutine $processForward(r, t)$

repeat
  $i_r = acc[r][0]$;
  $j_r = acc[r][1]$;
  $z_r = acc[r][2]$;
  $t_r = S_r[i_r] + S_r[j_r]$;
  $t'_r = (S_r[i_r \gg 3 \oplus j_r \ll 5] + S_r[i_r \ll 5 \oplus j_r \gg 3]) \oplus 0xAA$;
  $t''_r = j_r + S_r[j_r]$;
  $swap(guess[i_r], guess[j_r])$;
  if $\Big(guess[i_r] \neq EMPTY \wedge guess[j_r] \neq EMPTY \wedge guess[t_r] \neq EMPTY \wedge guess[i_r \gg 3 \oplus j_r \ll 5] \neq EMPTY \wedge guess[i_r \ll 5 \oplus j_r \gg 3] \neq EMPTY \wedge guess[t'_r] \neq EMPTY\Big)$ **then**
    if $guess[t''_r] = EMPTY$ **then**
      $guess[t''_r] \leftarrow z_r \oplus \big(guess[t_r] + guess[t'_r]\big)$;
    end
  end
  if $\Big(guess[i_r] \neq EMPTY \wedge guess[j_r] \neq EMPTY \wedge guess[t_r] \neq EMPTY \wedge guess[i_r \gg 3 \oplus j_r \ll 5] \neq EMPTY \wedge guess[i_r \ll 5 \oplus j_r \gg 3] \neq EMPTY \wedge guess[t''_r] \neq EMPTY\Big)$ **then**
    if $guess[t'_r] = EMPTY$ **then**
      $guess[t'_r] \leftarrow \big(z_r \oplus guess[t''_r]\big) - guess[t_r]$;
    end
  end
  if $\Big(guess[i_r] \neq EMPTY \wedge guess[j_r] \neq EMPTY \wedge guess[i_r \gg 3 \oplus j_r \ll 5] \neq EMPTY \wedge guess[i_r \ll 5 \oplus j_r \gg 3] \neq EMPTY \wedge guess[t'_r] \neq EMPTY \wedge guess[t''_r] \neq EMPTY\Big)$ **then**
    if $guess[t_r] = EMPTY$ **then**
      $guess[t_r] \leftarrow \big(z_r \oplus guess[t''_r]\big) - guess[t'_r]$;
    end
  end
  $r \leftarrow r + 1$;
until $r = t$;

Algorithm 3.11: Subroutine *processForward*

values of j are required to reconstruct the entire of S.

Rounds M	100	200	300	400	500	602
#Bytes Recovered	84	144	194	233	249	255

TABLE 3.5: No. of rounds vs. average no. of bytes recovered for Algorithm 3.9.

3.7 Conclusion

In this chapter we have analyzed the GGHN [68] and RC4+ [100] stream ciphers and presented several cryptanalytic results related to them. In the case of GGHN, our main motivation was to study the evolution of the cipher and we show that the cipher has several weaknesses that include a large family of short cycles of length equal to the length of the state array. We also concentrated on a theoretical model of the cipher (referred as RAND-GGHN-PRGA) in which the indices of the cipher are chosen uniformly at random from a certain range of values. Our analysis shows that such model evolves to all zero state much faster than what is expected in an ideal cipher. We have also presented experimental results related to two other models RAND-GGHN-PRGA$'$ and RAND-GGHN-PRGA$''$, the second one being very close to the actual GGHN PRGA and we found that it evolves to the all-zero state much faster. However, theoretical modelling of these two requires further investigation. The analysis in this chapter may be used by the future designers to consciously avoid the GGHN kind of evolution while designing a modified version of RC4.

Secondly, we have also presented some weaknesses of the RC4+ stream cipher. First, a distinguishing attack requiring around 2^{26} output samples was presented, based on the bias of the first output byte. Thereafter, a Differential Fault Attack requiring around $2^{17.2}$ faults was reported against the cipher. The results show that designing reinforcements to strengthen RC4 is not an easy task. It would be worthwhile to discover a design paradigm that not only rids RC4 of its weaknesses but also preserves its innate simplicity.

Chapter 4

Related Key-IV pairs of Grain

The Grain family of stream ciphers is one of the candidates in the eStream [116] hardware portfolio. It is because of its simplicity and elegance in design that it has attracted considerable attention from cryptologists worldwide. The family consists of three ciphers-Grain v1, Grain 128, Grain 128a, and each cipher presents unique challenges in terms of cryptanalysis. Although several researchers have analyzed the structure and evolution of the Grain family and its variants, cryptanalysis of Grain remains an open problem. This chapter provides an insight into the current status quo vis-a-vis the security of the Grain family and concludes with a list of open problems.

In this chapter, we explain how one can obtain Key-IV pairs for Grain family of stream ciphers that can generate output key-streams which are either

- Almost similar in the initial segment, or
- Exact shifts of each other throughout the generation of the stream.

Let l_P be the size of the pad used during the key loading of Grain. For the first case, we show that in expected 2^{l_P} invocations of the Key Scheduling Algorithm (this will be explained in Section 4.1.1) and its reverse routine, one can obtain two related Key-IV pairs that can produce same output bits in 75 (respectively 112 and 115) selected positions among the initial 96 (respectively 160) bits for Grain v1 (respectively Grain-128 and Grain-128a).

For Grain v1 and Grain-128, a similar idea works in showing that given any Key-IV, one can obtain another related Key-IV in expected 2^{l_P} trials such that the related Key-IV pairs produce shifted key-streams. We also provide an efficient strategy to obtain related Key-IV pairs that produce exactly i-bit shifted key-streams for small i. Our technique

pre-computes certain equations that help in obtaining such related Key-IV pairs in 2^i expected trials. The aforementioned related Key-IV pairs are obtained by exploiting the fact that, not Grain v1 and Grain-128 employ symmetric padding. Based on this symmetric padding, Lee et al. presented a chosen IV related Key attack on Grain v1 and Grain-128 at ACISP 2008 [96]. Thereafter, the designers introduced Grain-128a having an asymmetric padding. As a result, the method applied to obtain Key-IV pairs that produce shifted keystream pairs for Grain v1 and Grain-128 can not be extended to Grain-128a. We present a new method that succeeds that in obtaining related Key-IV pairs for Grain-128a that produce shifted keystreams.

4.1 Grain family of stream ciphers

The Grain v1 stream cipher is in the hardware profile of the eStream portfolio [116] that has been designed by Hell, Johansson and Meier in 2005 [73]. It is a synchronous bit oriented cipher, although it is possible to achieve higher throughput at the expense of additional hardware. The physical structure of Grain is simple as well as elegant and it has been designed so as to require low hardware complexity.

It would be worthwhile to mention that the initial design proposal [72] (let us call it Grain v0) to the Phase 1 of the eStream project was a little different from the current description of Grain v1. Two cryptanalytic advances were reported against Grain v0.

- The first was a state recovery attack proposed by Berbain et. al. in [31]. The attack took advantage of the fact that the NFSR update function had a resiliency of 1, i.e., it was possible to approximate the NFSR update function by a linear function of one variable. This lead to a probabilistic relation between the LFSR state variables and the output keystream of Grain v0. By accumulating a sufficient number of such equations the system was solved using a Maximum Likelihood Decoding method like the Fast Walsh Transform.
- A distinguishing attack requiring $2^{61.4}$ keystream bits and $O(2^{40})$ memory and time was reported in [88]. This attack again took advantage of the low resiliency of the NFSR update function and the output function of Grain v0.

Following these attacks on the initial design of the cipher, the modified version Grain v1 [73] was proposed. The designers made 2 significant changes to the design of Grain v0. First, the NFSR update function was changed to a boolean function with high resiliency. Second, a set of bits from the NFSR were linearly XOR-ed to the output function. These changes were able to successfully thwart the aforementioned attacks. Later, the

designers came up with a second version of Grain, i.e., Grain-128 [74] that uses 128 bit Key. Thereafter, cipher Grain-128a [12] was designed for the dual purpose of message authentication alongside message encryption, i.e., given an arbitrary length message, the cipher along with the encrypted ciphertext produces a 32 Message Authentication Code, for the purpose of preservation of message integrity.

4.1.1 Structure of ciphers in Grain family

The exact structure of the Grain family is explained in Figure 4.1. It consists of an n-bit LFSR and an n-bit NFSR. Certain bits of both the shift registers are taken as inputs to a combining Boolean function, whence the key-stream is produced. The update function of the LFSR is given by the equation $y_{t+n} = f(Y_t)$, where $Y_t = [y_t, y_{t+1}, \ldots, y_{t+n-1}]$ is an n-bit vector that denotes the LFSR state at the t^{th} clock interval and f is a linear function on the LFSR state bits obtained from a primitive polynomial in $GF(2)$ of degree n. The NFSR state is updated as $x_{t+n} = y_t \oplus g(X_t)$. Here, $X_t = [x_t, x_{t+1}, \ldots, x_{t+n-1}]$ is an n-bit vector that denotes the NFSR state at the t^{th} clock interval and g is a non-linear function of the NFSR state bits.

The output key-stream is produced by combining the LFSR and NFSR bits as $z_t = h'(X_t, Y_t) = \bigoplus_{a \in A} x_{t+a} \oplus h(X_t, Y_t)$, where A is some fixed subset of $\{0, 1, 2, \ldots, n-1\}$.

Key Loading Algorithm (KLA)

The Grain family uses an n-bit key K, and an m-bit initialization vector IV, with $m < n$. The key is loaded in the NFSR and the IV is loaded in the 0^{th} to the $(m-1)^{th}$ bits of the LFSR. The remaining m^{th} to $(n-1)^{th}$ bits of the LFSR are loaded with some fixed pad $P \in \{0,1\}^{n-m}$. Hence at this stage, the $2n$ bit initial state is of the form $K||IV||P$. Note that $n - m = l_P$ denotes the length of the pad used in the design. These terms will be used interchangeably in the course of this chapter.

Key Scheduling Algorithm (KSA)

After the KLA, for the first $2n$ clocks, the key-stream produced at the output point of the function h' is XOR-ed to both the LFSR and NFSR update functions, i.e., during the first $2n$ clock intervals, the LFSR and the NFSR bits are updated as $y_{t+n} = z_t \oplus f(Y_t)$, $x_{t+n} = y_t \oplus z_t \oplus g(X_t)$.

Pseudo-Random key-stream Generation Algorithm (PRGA)

After the completion of the KSA, z_t is used as the Pseudo-Random key-stream bit. It is no longer XOR-ed to the LFSR and the NFSR. Therefore during this phase, the LFSR and NFSR are updated as $y_{t+n} = f(Y_t), x_{t+n} = y_t \oplus g(X_t)$.

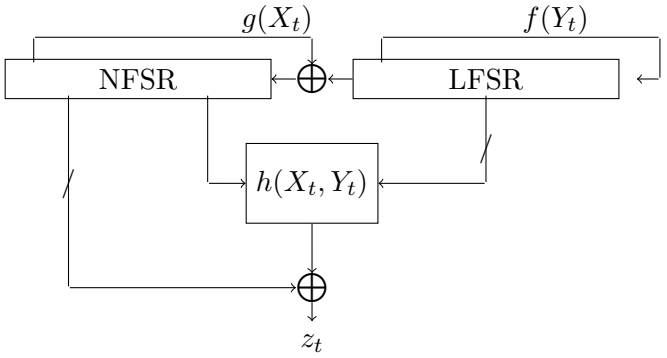


FIGURE 4.1: Structure of Stream Cipher in Grain Family

4.2 Complete Mathematical Description of the ciphers

4.2.1 Grain v1

Grain v1 consists of an 80 bit LFSR and an 80 bit NFSR. It uses an 80-bit Key and a 64-bit IV, and a 16-bit pad $P =$ `0x ffff`. Certain bits of both the shift registers are taken as inputs to a combining Boolean function, whence the key-stream is produced. The update function of the LFSR is given by the equation

$$y_{t+80} = y_{t+62} \oplus y_{t+51} \oplus y_{t+38} \oplus y_{t+23} \oplus y_{t+13} \oplus y_t \stackrel{\Delta}{=} f(Y_t).$$

The NFSR state is updated as follows

$$x_{t+80} = y_t \oplus g(x_{t+63}, x_{t+62}, x_{t+60}, x_{t+52}, x_{t+45}, x_{t+37}, x_{t+33}, x_{t+28}, x_{t+21}, x_{t+15} \\ x_{t+14}, x_{t+9}, x_t),$$

where $g(x_{t+63}, x_{t+62}, \ldots, x_t)$

$$\begin{aligned}\triangleq \quad g(X_t) \;=\; & x_{t+62} \oplus x_{t+60} \oplus x_{t+52} \oplus x_{t+45} \oplus x_{t+37} \oplus x_{t+33} \oplus x_{t+28} \oplus x_{t+21} \oplus \\ & x_{t+14} \oplus x_{t+9} \oplus x_t \oplus x_{t+63}x_{t+60} \oplus x_{t+37}x_{t+33} \oplus x_{t+15}x_{t+9} \oplus \\ & x_{t+60}x_{t+52}x_{t+45} \oplus x_{t+33}x_{t+28}x_{t+21} \oplus x_{t+63}x_{t+45}x_{t+28}x_{t+9} \oplus \\ & x_{t+60}x_{t+52}x_{t+37}x_{t+33} \oplus x_{t+63}x_{t+60}x_{t+21}x_{t+15} \oplus \\ & x_{t+63}x_{t+60}x_{t+52}x_{t+45}x_{t+37} \oplus x_{t+33}x_{t+28}x_{t+21}x_{t+15}x_{t+9} \oplus \\ & x_{t+52}x_{t+45}x_{t+37}x_{t+33}x_{t+28}x_{t+21}.\end{aligned}$$

The output key-stream is produced by combining the LFSR and NFSR bits as follows

$$z_t = \bigoplus_{a \in A} x_{t+a} \oplus h(y_{t+3}, y_{t+25}, y_{t+46}, y_{t+64}, x_{t+63}) \triangleq \bigoplus_{a \in A} x_{t+a} \oplus h(X_t, Y_t)$$

where $A = \{1, 2, 4, 10, 31, 43, 56\}$ and $h(s_0, s_1, s_2, s_3, s_4)$

$= s_1 \oplus s_4 \oplus s_0s_3 \oplus s_2s_3 \oplus s_3s_4 \oplus s_0s_1s_2 \oplus s_0s_2s_3 \oplus s_0s_2s_4 \oplus s_1s_2s_4 \oplus s_2s_3s_4.$

4.2.2 Grain-128

The cipher uses an 128-bit Key and a 96-bit IV, and a 32-bit pad $P =$ `0x ffff ffff`. The LFSR of Grain-128 is updated as

$$y_{t+128} = y_{t+96} \oplus y_{t+81} \oplus y_{t+70} \oplus y_{t+38} \oplus y_{t+7} \oplus y_t,$$

where the NFSR is updated as

$$\begin{aligned}x_{t+128} = y_t \oplus & x_t \oplus x_{t+26} \oplus x_{t+56} \oplus x_{t+91} \oplus x_{t+96} \oplus x_{t+3}x_{t+67} \oplus x_{t+11}x_{t+13} \oplus \\ & x_{t+17}x_{t+18} \oplus x_{t+27}x_{t+59} \oplus x_{t+40}x_{t+48} \oplus x_{t+61}x_{t+65} \oplus x_{t+68}x_{t+84}.\end{aligned}$$

The output key-stream bit is produced as

$$z_t = \bigoplus_{j \in B} x_{t+j} \oplus y_{t+93} \oplus h(x_{t+12}, y_{t+8}, y_{t+13}, y_{t+20}, x_{t+95}, y_{t+42}, y_{t+60}, y_{t+79}, y_{t+95})$$

where $B = \{2, 15, 36, 45, 64, 73, 89\}$ and $h(s_0, \ldots, s_8) = s_0s_1 \oplus s_2s_3 \oplus s_4s_5 \oplus s_6s_7 \oplus s_0s_4s_8$.

4.2.3 Grain-128a

The cipher uses an 128-bit Key and a 96-bit IV, and an asymmetric 32-bit pad $P =$ `0x ffff fffe`. The LFSR update functions of Grain-128 and Grain-128a are the same. There is a slight difference in the NFSR update function and the output function.

The NFSR update function for Grain-128a is given by

$$\begin{aligned} x_{t+128} =& y_t \oplus x_t \oplus x_{t+26} \oplus x_{t+56} \oplus x_{t+91} \oplus x_{t+96} \oplus x_{t+3}x_{t+67} \oplus x_{t+11}x_{t+13} \oplus \\ & x_{t+17}x_{t+18} \oplus x_{t+27}x_{t+59} \oplus x_{t+40}x_{t+48} \oplus x_{t+61}x_{t+65} \oplus x_{t+68}x_{t+84} \oplus \\ & x_{t+88}x_{t+92}x_{t+93}x_{t+95} \oplus x_{t+22}x_{t+24}x_{t+25} \oplus x_{t+70}x_{t+78}x_{t+82}. \end{aligned}$$

The output key-stream bit is generated as

$$z_t = \bigoplus_{j \in B} x_{t+j} \oplus y_{t+93} \oplus h(x_{t+12}, y_{t+8}, y_{t+13}, y_{t+20}, x_{t+95}, y_{t+42}, y_{t+60}, y_{t+79}, y_{t+94}),$$

where the function h and the set B are same as defined for Grain-128.

Authentication

As already mentioned, the Grain-128a cipher gives the user an option of producing a 32 bit Message Authentication Code (MAC) along with the encrypted ciphertext. We will describe the mechanism as explained in [13].

The basic design follows a Toeplitz matrix based Universal Hash Function Construction. Assume that we have a message of length L defined by the bits $m_0, \ldots, m_{L-1}$. Set $m_L = 1$. To provide authentication, two registers, called accumulator and shift register of size 32 bits each, are used (See Figure 4.2). The content of accumulator and shift register at time t are denoted by $a_t^0, \ldots, a_t^{31}$ and $r_t, \ldots, r_{t+31}$, respectively. The accumulator is initialized through $a_0^t = z_t, 0 \leq t \leq 31$ and the shift register is initialized through $r_t = z_{32+t}, 0 \leq t \leq 31$. The shift register is updated as $r_{t+32} = z_{64+2t+1}$. The accumulator is updated as $a_{t+1}^j = a_t^j \oplus m_t r_{t+j}$ for $0 \leq j \leq 31$ and $0 \leq t \leq L$. The final content of accumulator, $a_{L+1}^0, \ldots, a_{L+1}^{31}$ is the 32 bit tag used as the MAC.

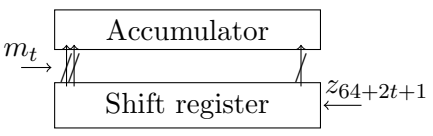


FIGURE 4.2: Authentication mechanism in Grain-128a

4.3 Reversible KSA and PRGA of the Grain family

One may note that given any arbitrary state and the information about its evolution (the number of clocks in KSA or PRGA), one can calculate the corresponding state S_0^K at the beginning of the KSA. This is because the state update functions in both the KSA and PRGA in the Grain family are one-to-one and invertible. Hence one can construct the KSA^{-1} routine that given an input $2n$ bit vector denoting the internal state of the cipher at the end of the KSA, returns the $2n$ bit vector giving internal state of the cipher at the beginning of the KSA. One can similarly describe a PRGA^{-1} routine that inverts one round of the PRGA.

Given the primitive polynomial of the Grain LFSR, the feedback function f is of the form

$$f(Y_t) = y_t \oplus f'(Y_t'),$$

where $Y_t' = [y_{t+1}, \ldots, y_{t+n-1}]$ is an $(n-1)$-bit vector obtained from Y_t by removing the first term. The NFSR update function g is of the form

$$g(X_t) = x_t \oplus g'(X_t'),$$

where $X_t' = [x_{t+1}, \ldots, x_{t+n-1}]$ is an $(n-1)$-bit vector obtained from X_t by removing the first term x_t. This implies that the functions g', f' does not depend on the terms x_t, y_t respectively. This is necessary as well as sufficient for the state update function of the NFSR and LFSR to be one-one [63]. In fact, it has been shown in [63], that any feedback shift register with update functions of the above form, would lead to a state sequence diagram that would be branchless, i.e., if each state were regarded as a node in a connected graph, then the in-degree and out-degree of each node would be exactly 1. Due to this, the state update maps of the Grain family of ciphers during both the KSA and the PRGA are one to one and invertible, i.e., given any particular state, during any iteration of the KSA or the PRGA, it is possible to determine the previous state. Given the NFSR and LFSR state after the completion of KSA, Algorithm 4.1 will determine the NFSR and LFSR state at the beginning of the KSA. One can similarly invert the PRGA. Given the NFSR and LFSR state during any clock interval of the PRGA, Algorithm 4.2 will determine the LFSR and NFSR state in the preceding clock interval.

Input: State $S_0 = x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}$
Output: State $S_0^K = x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}$

---

for $2n$ *clocks* **do**
  $l_j = y_{n-1}$ and $n_j = x_{n-1}$
  $y_i = y_{i-1}$ and $x_i = x_{i-1}$ for $i = n-1, n-2, \ldots, 1$
  $z = \bigoplus_{a \in A} x_a \oplus h(x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1})$
  $y_0 = z \oplus l_j \oplus f'(y_1, \ldots, y_{n-1})$
  $x_0 = z \oplus n_j \oplus y_0 \oplus g'(x_1, \ldots, x_{n-1})$
end

Algorithm 4.1: KSA^{-1} routine for the Grain Family

Input: State $S_0 = x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}$
Output: The preceding state $x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}$ of the PRGA

$l_j = y_{n-1}$ and $n_j = x_{n-1}$
$y_i = y_{i-1}$ and $x_i = x_{i-1}$ for $i = n-1, n-2, \ldots, 1$
$y_0 = l_j \oplus f'(y_1, \ldots, y_{n-1})$
$x_0 = n_j \oplus y_0 \oplus g'(x_1, \ldots, x_{n-1})$

Algorithm 4.2: One round PRGA^{-1} routine for the Grain family

In other words, given any arbitrary state and the information about its evolution (the number of clocks in KSA or PRGA), one can calculate the corresponding state at the beginning of the KSA.

4.4 Existing cryptanalytic results on the Grain family

4.4.1 Distinguishing Attacks

There is till date no distinguishing attack against the full versions of any of the ciphers in this family which works for all possible Key-IV pairs. However such attacks have been reported against variants of Grain in which the number of rounds in the **KSA** have been reduced. In [58], a distinguisher based on the d-monomial test was reported against Grain-128 when the number of initialization rounds was reduced from 256 to 192 rounds. In [128], a distinguishing attack on a variant of Grain-128 that uses 246 initialization rounds was presented, which works for less than half of the keys. In [14], cube testers were used in order to distinguish Grain-128 from random for up to 237 initialization rounds. In [97], cube testers were used to distinguish a variant of Grain-128a, that uses 189 out of the 256 rounds.

In [137], a distinguishing attack against the full versions of Grain v1 and Grain-128 was reported. But the attack worked for an extremely small fraction 2^{-n} of the entire Key-IV space. The attack exploited the situation that for about 2^{-n} fraction of Key-IVs the LFSR goes into the all-zero state just after the completion of the KSA and never comes out of it during the PRGA. In such a scenario, the NFSR becomes autonomous and a distinguishing attack can be mounted using the linear biases of the NFSR update function g.

4.4.2 Key recovery Attacks

In [61], a key-recovery attack was reported against a variant with of Grain-128 with 180 initialization rounds. In [44], the sliding property was used to speedup exhaustive search by a factor of two. In [94], conditional differential cryptanalysis was used to distinguish and recover 5 Key bits i) a variant of Grain v1 that uses 104 of the 160 KSA rounds and ii) a variant of Grain-128 that employs 213 out of the 256 KSA rounds. This attack works for all Key-IV pairs. A Time-Memory tradeoff attack was reported in [39] using the low sampling resistance of the cipher. In [108], the low sampling resistance was combined with the fact that output function of Grain v1 is 2-normal, to propose an improved TMD-Tradeoff attack.

4.4.3 Cube Attacks

Cube attacks was first introduced by Dinur and Shamir in [55] and have become a useful tool to cryptanalyze stream ciphers. In [56], three cryptanalytic advances were made against Grain-128. The first attack ran in practical time complexity and recovered the full 128-bit key when the number of initialization rounds was reduced to 207. The second attack worked on a Grain-128 variant with 250 initialization rounds and was faster than exhaustive search by a factor of about 2^{28}. An attack against the full version of Grain-128 was also presented which was able to recover the full key only when it belonged to a subset of 2^{118} of all possible keys. This third attack is faster than exhaustive search by a factor of about 2^{15}.

An improved attack over [56] was presented in [57], in which an attack faster than exhaustive search by 2^{38} for all possible keys, was proposed.

4.4.4 Fault Attacks

Fault attacks on stream ciphers have gained attention ever since the work of Hoch and Shamir [76] describing such an attack was published and such attacks have been

successfully employed against a number of stream ciphers. In [35] fault attacks against Grain-128 was reported under the assumption that a fault at a random LFSR location could be reproduced more than once. [85] repeated the attack on Grain-128 under the same assumptions but targeted the NFSR for injecting faults.

4.4.5 Slide based Related Key Attacks

In both Grain v1 and Grain-128, the symmetric padding of all ones is used during the initialization of the internal state of the cipher, before the Key-IV mixing. Due to this symmetric padding, slide attacks based on the observation that one can obtain Key-IV pairs that produce ϵ-bit shifted keystream with probability $2^{-2\epsilon}$, were reported in [44].

The symmetric padding used in the initialization of Grain v1 and Grain-128 was also exploited in [96] to mount a chosen IV related Key attack. Their main idea is to use related Keys and chosen IVs to obtain shifted keystream and then to carefully study the scenario to obtain the Secret Key bits. However due to the asymmetric nature of the pad used in Grain-128a, such analysis can not be extended to this cipher.

4.4.6 Other results

A TMD Tradeoff attack was proposed in [136], but some of the assumptions used in this paper were erroneous and as such the validity of this attack is questionable.

4.4.7 Our results

Our results in this chapter are motivated towards studying how given a Key-IV, one can efficiently obtain another Key-IV so that the generated output key-streams are

- Almost similar in the initial segment or
- Exact shifts of each other throughout the key-stream generation.

We call these Key-IV pairs "related" following [44, Section 3].

Since the Grain family of stream ciphers are essentially finite state machines, we can make several interesting observations. Any pair of internal states during the key-stream production stage (say S_0 and $S_{0,\Delta}$) that differ only in a few bit positions (say not exceeding three), produce very similar key-stream bits at-least in the first few output rounds. The idea therefore is to come up with two distinct Key-IV pairs (K, IV) and $(K, IV)_\Delta$,

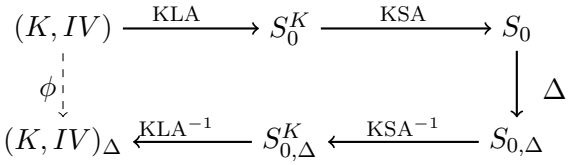


FIGURE 4.3: Construction of the Related Key-IV function.

so that after the key initialization round, produce the states $S_0, S_{0,\Delta}$ respectively. These Key-IV pairs would then produce key-stream which would be initially very similar to one another.

On the other hand, since the state update functions of the cipher are one-to-one and invertible, two distinct Key-IV pairs (K, IV) and (K', IV') will never produce exactly the same state S after the kSA. However, it may be possible that the Key-IV pair (K', IV'), after producing a certain number of output bits (say i), lands on the state S which is the same state that (K, IV) lands on after Key initialization. Since Grain is a finite state machine, the key-stream produced by (K', IV') after these i rounds is exactly the same as that produced by (K, IV). These Key-IV pairs will then produce i bit shifted key-stream.

Though our work does not have any immediate implication towards breaking any cipher of the Grain family, the observations are relevant in cryptographic scenario.

4.5 Related Key-IV pairs in Grain family

Let us now explain our interpretation of related Key-IV pairs. This is in line of what is explained in [44, Section 3]. For this we need the construction of the related Key-IV function ϕ as illustrated in Figure 4.3. Note that we require both routines KSA and KLA to be uniquely reversible for a successful construction of ϕ. The goal of constructing ϕ is to obtain a pair of related Key-IVs (K, IV) and $(K, IV)_\Delta$ such that they produce either almost similar initial key-streams or shifted key-streams throughout the generation. With explicit construction of such functions ϕ, we will show that given any Key-IV (K, IV) in the Grain family, it is possible to find related Key-IV pair $(K, IV)_\Delta$.

4.5.1 Search for related Key-IV pairs in Grain v1

The non-linear function h in Grain v1, that takes inputs from both the linear and non-linear registers to produce the key-stream, taps the 64^{th} bit of the LFSR and no bit in between 65^{th} and the 79^{th}. This implies that if there exist two initial states S_0

and $S_{0,\Delta} \in \{0,1\}^{160}$ during the PRGA, such that S_0 and $S_{0,\Delta}$ differ in only a few bit positions, then there is a good possibility that some initial bits (may not be contiguous) of the key-stream will be same. We explain the complete scenario with a specific case here, when the last bit of the two states are different, and all other bits are identical. So, this is single-bit differential for the state S_0. However, there are many other such possibilities that may also be explored.

Let us consider that S_0 and $S_{0,\Delta}$ differ only in the 79^{th} LFSR position. In such a case, it is easy to check that they will produce identical key-stream for some initial PRGA rounds. We will also show that it is possible to produce Key-IV pairs (K_0, IV_0) and (K_1, IV_1) with $K_i \in \{0,1\}^{80}$ and $IV_i \in \{0,1\}^{64}$ so that after key-scheduling the pair (K_0, IV_0) produces the initial state S_0 of the PRGA and the pair (K_1, IV_1) produces the initial state $S_{0,\Delta}$.

First, we will look at a method to compute such related pairs (K_0, IV_0) and (K_1, IV_1). The method works because the KSA is invertible, i.e., given an initial state of the key-stream production stage it is possible to back-track and determine the Key-IV pair that produced it. The following method is, in principle, similar to the technique used by Zhang et al [137]. The basic idea is to generate at random Key-IV pair $K_0, IV_0 \in \{0,1\}^{80} \times \{0,1\}^{64}$ and calculate the initial state S_0 of the PRGA. Then after flipping the y_{79} bit of S_0 we produce the state $S_{0,\Delta}$ and backtrack to find out if there exists a Key-IV pair that produces $S_{0,\Delta}$. We will state the algorithm formally now.

Output: Key-IV pair's that produces almost similar initial key-stream or Failure

Randomly choose a Key-IV pair $(K, IV) \in \{0,1\}^{80} \times \{0,1\}^{64}$;
Obtain the initial state of the KSA $S_0^K = [K \,||\, IV \,||\, \texttt{0x ffff}]$;
Run the KSA for 160 clocks to produce an initial state $S_0 \in \{0,1\}^{160}$;
Construct $S_{0,\Delta}$ from S_0 by flipping the bit y_{79};
Compute $S_{0,\Delta}^K = \text{KSA}^{-1}(S_{0,\Delta})$ as the KSA routine is invertible;
if $S_{0,\Delta}^K$ *is of the form* $[\tilde{K} \,||\, \tilde{IV} \,||\, \texttt{0x ffff}\,]$ **then**
  Return (K, IV) and $(K, IV)_\Delta = (\tilde{K}, \tilde{IV})$ as the related Key-IV pairs;
end
else
  Return failure;
end

Algorithm 4.3: Search for related Key-IV pairs in Grain v1

Given that l_P is the length of the Pad P (which is a specific pattern among all the l_P bit patterns), it is expected that we will be able to obtain related Key-IV pairs in $2^{l_P} = 2^{16}$

runs of the algorithm (as if obtaining the specific pattern through random search). The next thing that we need to check is the propagation of the single-bit differential into the key-stream of the cipher during the PRGA. This is described in the following technical result.

Theorem 4.1. *For Grain v1, the two initial states* $S_0, S_{0,\Delta} \in \{0,1\}^{160}$ *which differ only in the* 79^{th} *position of the LFSR, produce identical output bits in 75 specific positions among the initial 96 key-stream bits produced during the PRGA.*

Proof. Any input differential introduced in the 79^{th} LFSR position takes 15 clocks before appearing at the 64^{th} position, and hence the first 15 bits z_0 to z_{14} will be exactly the same. In the 16^{th} round, the differential arrives at the 64^{th} position of the LFSR, which contributes an input to the Boolean function h and hence this bit may be different. Hereafter, the differential proceeds to the 63^{rd} LFSR position, which does not provide an input to h and hence in this round the output is the same. In the next round the differential is at the 62^{nd} position, which although does not feed the output function h, provides an input to the LFSR update function, due to which a difference reappears in the 79^{th} position. This new difference will now affect the key-stream after 15 rounds. Thus by keeping track of the propagation of the differential for the first 96 PRGA rounds it is possible to determine which rounds produce the same output bit. At all rounds numbered $k \in [0, 95] \setminus \{15, 33, 44, 51, 54, 57, 62, 69, 72, 73, 75, 76, 80, 82, 83, 87, 90, 91, 93, 94, 95\}$, the difference exists only in positions that do not provide input to the Boolean function h and hence at these clocks the key-stream bit produced by the two states are essentially the same. At all other clock rounds the difference appears at positions which provide input to h. Hence the key-stream produced at these clocks may be different. After 96 rounds the input difference is fed to the non-linear update function g of the NFSR, and hereafter the propagation of the difference would depend on the particular NFSR state at that point. □

4.5.2 Examples of related Key-IV pairs in Grain v1

In case of a practical search for related pairs of Key-IV, we notice that the Algorithm 4.3 is expected to run 2^{16} times for obtaining one pair of related Key-IV's. Now, this invocation may be accomplished in many ways. First we consider the example for the situation as mentioned in Algorithm 4.3.

Multiple Key-IV trials with a Fixed Differential.

Consider a fixed differential Δ for all the Key-IV pairs. In this case, Algorithm 4.3 needs to run expected 2^{16} times with different randomly chosen (K, IV)'s to obtain a related Key-IV pair (K, IV) and $(K, IV)_\Delta$. In case of Grain v1, this Δ represents 'flipping the last bit of the LFSR'. With expected 2^{16} queries in each case, we obtained related Key-IV pairs during the experiments. One such example is as follows.

Key	*IV*	*S*
bf6689cead5ece39758c	bdfa0025ac44a4fe	52f71a93959ff900ffa9 15c61a47522fffaf8a77
e166bc5aa1952733ab2a	aed6838b948399a0	52f71a93959ff900ffa9 15c61a47522fffaf8a76

One can check that out of the initial 96 key-stream bits, 75 specific bits are same as per Theorem 4.1 and in particular, 78 are same in this case.

Single Key-IV trial with a Multiple Differentials.

Now suppose a more practical situation, where a single pair of Key-IV is provided, and one has to produce a related pair of Key-IV corresponding to the one given. In this situation, one may experiment with different values of Δ. If around 2^{16} different Δ's can be used, then given a specific (K, IV) a related Key-IV pair may be expected. In this case, the expected number of invocations of KSA^{-1} is 2^{16}, one for each Δ. However, the number of KSA invocation is only one as we have only a single Key-IV pair for the complete strategy.

In case of Grain v1, we first observed that the single-bit change in the LFSR results towards related initial bits of the key-stream. Similar situation is expected to happen for 1, 2 or 3-bit changes in the LFSR, as the changes are still minor compared to the total size of the state. Thus, we chose a simple family of Δ where a single bit differential is introduced in at most 3 bits out of the 75 bits of the LFSR; the LFSR has 80 bits and we exclude the 4 bits $(3^{rd}, 25^{th}, 46^{th}, 64^{th})$ that go to the Boolean function h and the 0^{th} bit that goes to the NFSR. One may note that $\binom{75}{1} + \binom{75}{2} + \binom{75}{3} > 2^{16}$ and thus it is expected to obtain a related Key-IV pair.

Below we present an example, where the states differ in three bit positions of the LFSR, namely $47, 52, 54$. Out of the initial 80 key-stream bits produced, 55 are same.

Key	*IV*	*S*
bde8d3c319ff4d234706	f363180e262b6cc5	a74e7c7799b00f3c94e1 bf0315b589691f82085a
b223a57ce1578708677a	371d2d93363b014b	a74e7c7799b00f3c94e1 bf0315b589681582085a

4.5.3 Related Key-IV's in Grain-128

The structure of Grain-128 is similar to Grain v1. The only differences are in the update functions of the LFSR, NFSR, the combining output function and the sizes of the LFSR and NFSR. The key is loaded in the NFSR and the IV is loaded in the 0^{th} to the 95^{th} bits of the LFSR. The remaining 96^{th} to 127^{th} bits of the LFSR are loaded with 1's (the 32-bit pad P, i.e., $l_P = 32$). Here 256 rounds of KSA are executed after which the key-stream is produced. As in Grain v1, here too, the KSA is invertible. After an expected number of $2^{l_P} = 2^{32}$ trials two related Key-IV pairs (K, IV) and $(K, IV)_\Delta$ can be found. For these Key-IV pairs, the KSA gives initial states S_0 and $S_{0,\Delta}$ that differ only in the 127^{th} bit position.

Propagation of the Differential.

The following result describes the differential propagation characteristics of the single-bit differential Δ in case of Grain-128.

Theorem 4.2. *For the Grain-128 stream cipher, two initial states $S_0, S_{0,\Delta} \in \{0,1\}^{256}$ which differ only in the 127^{th} position of the LFSR, produce identical output bits in 112 specific positions among the initial 160 key-stream bits produced during the PRGA.*

Proof. As described in Theorem 4.1, a similar analysis applies in the case of Grain-128. By tracking the evolution of the single bit differential introduced at the 127^{th} LFSR position, it is possible to determine the clock rounds for which the output key-stream is exactly similar for the related Key-IV pairs which give rise to such a differential after the KSA. In Grain-128, the output key-stream for the following rounds numbered

$$\begin{aligned} k \in [0, 159] \setminus \{&32, 34, 48, 64, 66, 67, 79, 80, 81, 85, 90, 92, 95, 96, 98, 99, 106, 107, \\ &112, 114, 117, 119, 122, 124, 125, 126, 128, 130, 131, 132, 138, 139, \\ &142, 143, 144, 145, 146, 148, 149, 150, 151, 153, 154, 155, 156, 157, \\ &158, 159\} \end{aligned}$$

produced by S_0 and $S_{0,\Delta}$ are identical. □

Below we present a related Key-IV pair for Grain-128. One can verify that out of the first 160 keystream bits produced by the Key-IV pairs in the given example, 112 specific bits are same as per Theorem 4.2 and 132 bits are same in total.

Key	*IV*	*S*
60287a5ecf99724716a83bf81a9735cf	62b6f21aa5d6511f43cb51f0	7bb026436bc29b585e676e90961830e0 7e86e48d2370eeda43ddd098a4b3e7d2
dc260a0042112620772443311b933f08	c026cf1526950adee08fbe14	7bb026436bc29b585e676e90961830e0 7e86e48d2370eeda43ddd098a4b3e7d3

4.5.4 Related Key-IV's in Grain-128a

As already mentioned, the LFSR update functions of Grain-128 and Grain-128a are the same. There is a slight difference in the NFSR update function and the output function. Also Grain-128a uses the pad `0x ffff fffe` in the last 32 bits of the LFSR instead of the all 1 pad.

It is known that the first 64 output bits of Grain-128a are used for initializing the MAC and thereafter each alternative bit is used as the key-stream and the other bit is used for constructing the MAC. Let us first refer to all these bits as as output bits (these are referred as pre-output stream in [12]) and then analyze the exact scenario.

Propagation of the Differential.

The following result describes the differential propagation characteristic of the single-bit differential Δ in case of Grain-128a.

Theorem 4.3. *For the Grain-128a stream cipher, two initial states $S_0, S_{0,\Delta} \in \{0,1\}^{256}$ which differ only in the 127^{th} position of the LFSR, produce identical output bits in 115 specific positions among the initial 160 output bits produced during the PRGA.*

Proof. The proof follows a similar analysis as done for Grain v1 and Grain-128. The output bits for rounds

$$\begin{aligned} k \in [0, 159] \setminus \{&33, 34, 48, 65, 66, 67, 80, 81, 85, 91, 92, 95, 97, 98, 99, 106, 107, 112, \\ &114, 117, 119, 123, 124, 125, 127, 128, 129, 130, 131, 132, 138, 139, \\ &142, 143, 144, 145, 146, 149, 150, 151, 154, 155, 156, 157, 159\} \end{aligned}$$

produced by S_0 and $S_{0,\Delta}$ are identical. □

In case of Grain-128a, the *Pad* is of length $l_P = 32$ bits, and the number of bits that are same in the key-streams produced by S_0 and $S_{0,\Delta}$ is 115. Thus, the complexity of getting related pairs in these cases is expected 2^{32}. Moreover, 11 bits in the key-stream, the bits $\{34, 66, 81, 92, 98, 124, 128, 130, 145, 150, 156\}$ are always different in Z and Z_Δ.

This is because at these rounds the difference appears on one of the NFSR state bits which are linearly added to the output of the $h(\cdot)$ function to produce the keystream.

Out of the initial 160 output bits from Grain-128a, the initial 64 are used for MAC. Thus we are now left with $160 - 64 = 96$ bits. Again out of those, half of them will be used for MAC and half of them will be used as key-stream bits. Thus, we have actually considered 48 key-stream bits. The first 160 output bits are indexed as 0 to 159. Among them the even numbered bits from 64 to 159 are the key-stream bits. One may note that given two related Key-IV pairs, for the initial 48 key-stream bits, 30 will be exactly same, 8 will be exactly complement of each other and rest 10 cannot be determined before-hand.

Below we present a related Key-IV pair for Grain-128a. One can verify that out of the first 48 keystream bits produced by the Key-IV pairs in the given example, 30 specific bits are same as per Theorem 3 and 33 bits are same in total.

Key	*IV*	*S*
54fd23a7e54f8fb096a45189b65f0fff	5a7fb7b76c303592b74422c3	36a0589046e177ae325a4b60154084cd fc74e3c99cad9a2f2fcbf394d44f15fd
1c21c39e9404b1c347ee8dc594f3d040	9db86204107b9ac4d401cc2d	36a0589046e177ae325a4b60154084cd fc74e3c99cad9a2f2fcbf394d44f15fc

4.6 Occurrence of Key-IV pairs that produce shifted key-streams

The size of the Key-IV space in Grain being $\{0,1\}^n \times \{0,1\}^m$, one may expect that the cipher produces 2^{n+m} different key-streams. However, many of these key-streams are finite bit-position shifts of one another, that is natural in this kind of design. We have already noted that both the KSA and PRGA routines in the Grain family are invertible. Thus, given any Key-IV in the Grain family, it may be possible to find another Key-IV pair that produces a bit-shifted key-stream.

Let $\psi : \{0,1\}^{2n} \to \{0,1\}^{2n}$ be the state update function during the PRGA of Grain. The goal is to construct a *related key function* $\phi : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n \times \{0,1\}^m$ such that (K_0, IV_0) and $\phi(K_0, IV_0)$ produce shifted key-streams. The construction of ϕ in our constrained model of the stream cipher is as in Algorithm 4.4.

Thus, given any Key-IV in the Grain family, it is possible to find another Key-IV such that both of them produce key-streams which are finite bit shifts of one another, in an expected 2^{l_P} iterations where l_P is the length of the pad P. This holds under the assumption that, after the reverse KSA routine the last l_P bits of the LFSR are uniformly

Output: Key-IV pair that produces shifted key-stream or return failure

Randomly choose a Key-IV pair $(K, IV) \in \{0,1\}^n \times \{0,1\}^m$;
Obtain the initial state of the KSA $S_0^K = [K \,||\, IV \,||\, P]$;
Run the KSA for $2n$ clocks to produce an initial state $S_0 \in \{0,1\}^{2n}$;
Initialize $i = 1$;
1 Construct the state S_i by running one more round of the PRGA;
Reverse the KSA routine to generate the initial state $S_i^K = \text{KSA}^{-1}(S_i)$;
if S_i^K *is of the form* $[\tilde{K} \,||\, \tilde{IV} \,||\, P]$ **then**
  Return related Key-IV pair (K, IV), $(K, IV)_i = (\tilde{K}, \tilde{IV})$, and the shift i;
  $i \leftarrow i + 1$;
end
if i *is greater than some predefined threshold* **then**
  Return failure;
end
Return to step 1 and repeat after running another round of the PRGA;

Algorithm 4.4: Related Key-IV function ϕ for shifted key-streams in Grain

distributed. Hence, it is expected that we will be able to obtain related Key-IV pairs in 2^{l_P} iterations of the loop.

We present a set of examples here for each of the stream cipher in the Grain family. Given the Key-IV pairs in column 1 of the following table, column 2 gives a related Key-IV pair that produces shifted key-stream. Column 3 gives the length of the shift.

Grain	Key-IV	Key-IV	Shift
v1	4567b66f51b956542319 96b81c6c97ed8853	f0f9d3bc4f2d0001e11d 67e95df014caf50a	72343 $\approx 2^{16.14}$
128	fca5c3705794a26266f58d06f7e87b9f cf74e27475fc36e159069606	990aa66d1d816db4d81cf42ab62937b2 54345cb47fed0997dc1a73d4	236757088 $\approx 2^{27.82}$
128a	2b953abc7427e1c260b2995039766123 81a25f710a9a24aed1644d9f	01f8cda5aa35dece20154a986e24e4d8 4bf4f64d462d379453928a7a	2642097831 $\approx 2^{31.30}$

4.6.1 Improved strategy over [44] for small shift

In [44], Key-IV pairs in Grain v1, that produce shifted key-streams, were demonstrated. The idea is as follows. First we demonstrate the idea for for 1-bit shift for simplicity of explanation. If $K_0 \in \{0,1\}^n$ and $IV_0 \in \{0,1\}^m$ denote a Key-IV, then the initial state of the KSA is denoted by $B_0 = K_0$ and $C_0 = IV_0||P$. After the first round of KSA, the updated initial states are denoted by $B_1||C_1$.

- If C_1 can be written in the form $IV_1||P$ for $IV_1 \in \{0,1\}^m$, then $B_1||C_1 = K_1||IV_1||P$ is another valid initial state of the KSA. So if the KSA starts with the state $B_1||C_1$ instead of $B_0||C_0$, it may produce one bit-shifted key-streams.
- An added sufficiency condition is required. The 1^{st} output bit produced by the KSA initial state B_0, C_0 during the PRGA must be 0. This is required to ensure that the state after the $2n^{th}$ round of the KSA using $B_1||C_1$, is the same as the state after the 1^{st} PRGA round using (B_0, C_0).

If both the above conditions are satisfied then (K_0, IV_0) and (K_1, IV_1) will indeed produce 1-bit shifted key-streams. Both the events have a probability of occurrence of $\frac{1}{2}$ and hence a related Key-IV pair may be found with probability $\frac{1}{4}$ by randomly choosing Key-IV pairs. This idea extends to i rounds, so that two Key-IV pairs which produce i-bit shifted key-stream may be obtained with probability $(\frac{1}{4})^i$.

However, in this section we show that the probability may be improved to $(\frac{1}{2})^i$ by explicitly characterizing the structure of the Key-IV which, in every round of KSA out of those i rounds, produce valid KSA internal states.

To analyze this, let us study the KSA in more detail. Given a key IV pair $K_0 = x_0, x_1, \ldots, x_{n-1}$ and $IV_0 = y_0, y_1, \ldots, y_{m-1}$, the state update function during the KSA can be presented in the following way. Consider that $x_i^{[j]}$ ($y_i^{[j]}$) is the value in the i^{th} cell of the NFSR (LFSR) in the j^{th} KSA round. Denote $B_0 \triangleq x_0^{[0]}, x_1^{[0]}, \ldots, x_{n-1}^{[0]}$, $C_0 \triangleq y_0^{[0]}, y_1^{[0]}, \ldots, y_{m-1}^{[0]}||P = y_0^{[0]}, y_1^{[0]}, \ldots, y_{n-1}^{[0]}$.

Input: B_0, C_0

Output: B_i, C_i, for $i = 1$ to u

---

for $i = 1$ *to* u **do**

$y^{[i]} \leftarrow f(Y^{[i-1]})$ where $Y^{[i-1]} = y_0^{[i-1]}, y_1^{[i-1]}, \ldots, y_{n-1}^{[i-1]}$

$x^{[i]} \leftarrow y_0^{[i-1]} \oplus g(X^{[i-1]})$ where $X^{[i-1]} = x_0^{[i-1]}, x_1^{[i-1]}, \ldots, x_{n-1}^{[i-1]}$

$z^{[i]} \leftarrow \bigoplus_{a \in A} x_a^{[i-1]} \oplus h(X^{[i-1]}, Y^{[i-1]})$

$B_i = (x_0^{[i]}, x_1^{[i]}, \ldots, x_{n-2}^{[i]}, x_{n-1}^{[i]}) \leftarrow (x_1^{[i-1]}, x_2^{[i-1]}, \ldots, x_{n-1}^{[i-1]}, x^{[i]} \oplus z^{[i]})$

$C_i = (y_0^{[i]}, y_1^{[i]}, \ldots, y_{n-2}^{[i]}, y_{n-1}^{[i]}) \leftarrow (y_1^{[i-1]}, y_2^{[i-1]}, \ldots, y_{n-1}^{[i-1]}, y^{[i]} \oplus z^{[i]})$

end

Algorithm 4.5: Obtaining Grain KSA Relations

In Grain v1, for $B_1||C_1$ to represent a valid initial state of the KSA it must be of the form $[\tilde{K} \,||\, \tilde{IV} \,||\, P]$, Thus following Algorithm 4.5, this will occur if $y^{[1]} \oplus z^{[1]} = 1$ in the

first iteration of the KSA (where $y^{[i]}, z^{[i]}$ are as defined in Algorithm 4.5). This implies

$$y_{62} \oplus y_{51} \oplus y_{38} \oplus y_{23} \oplus y_{13} \oplus y_0 \oplus \bigoplus_{a \in A} x_a \oplus h(y_3, y_{25}, y_{46}, y_{64}, x_{63}) = 1, \tag{4.1}$$

that evaluates to

$$\begin{aligned} y_{62} =&((y_{25} \oplus x_{63} \oplus 1)y_3 \oplus y_{25}x_{63} \oplus x_{63} \oplus 1)y_{46} \oplus y_{23} \oplus y_{25} \oplus x_{10} \oplus y_{38} \oplus y_{51} \oplus \\ & x_1 \oplus x_2 \oplus x_{31} \oplus x_{43} \oplus x_4 \oplus x_{56} \oplus y_0 \oplus y_3 \oplus y_{13} \oplus 1. \end{aligned}$$

This covers the case of [44] for 1-bit shift. Now, towards the extension, let us consider the case for 2-bit shift.

For both $B_1||C_1$ and $B_2||C_2$ to be valid initial states, in addition to (4.1), we need the following condition ($y^{[2]} \oplus z^{[2]} = 1$) to hold:

$$y_{63} \oplus y_{52} \oplus y_{39} \oplus y_{24} \oplus y_{14} \oplus y_1 \oplus \bigoplus_{a \in A} x_{a+1} \oplus h(y_4, y_{26}, y_{47}, y_{65}, x_{64}) = 1. \tag{4.2}$$

Solving (4.1) and (4.2), we obtain

$$\begin{aligned} y_{62} =&((y_{25} \oplus x_{63} \oplus 1)y_3 \oplus y_{25}x_{63} \oplus x_{63} \oplus 1)y_{46} \oplus y_{23} \oplus y_{25} \oplus x_{10} \oplus y_{38} \oplus y_{51} \oplus \\ & x_1 \oplus x_2 \oplus x_{31} \oplus x_{43} \oplus x_4 \oplus x_{56} \oplus y_0 \oplus y_3 \oplus y_{13} \oplus 1, \end{aligned}$$

$$\begin{aligned} y_{63} =&((y_{26} \oplus x_{64} \oplus 1)y_4 \oplus y_{26}x_{64} \oplus x_{64} \oplus 1)y_{47} \oplus y_{24} \oplus y_{26} \oplus x_{11} \oplus y_{39} \oplus y_{52} \oplus \\ & x_2 \oplus x_3 \oplus x_{32} \oplus x_{44} \oplus x_5 \oplus x_{57} \oplus y_1 \oplus y_4 \oplus y_{14} \oplus 1. \end{aligned}$$

Similarly, by solving together equations of the form $y^{[i]} \oplus z^{[i]} = 1$ for each successive round of the KSA, we would be able to determine the necessary conditions that need to be satisfied for each successive internal state $(B_i||C_i), i = 1, 2, \ldots$ to be valid initial states of the Grain v1 KSA. In the Appendix at the end of this chapter, we present the equations required for 11-bit shift. It can be seen in the Appendix that the solutions to the equations, so obtained, for the individual IV variables y_i are themselves quite complicated algebraic equations, and thus computing them by randomly selecting the degrees of freedom r_i may take some finite computational time T. In such an event it would be more appropriate to express the computational complexity as $T \cdot 2^i$. Using mathematical tools like SAGE [129], we could arrive at the solution to the simultaneous equations $y^{[i]} \oplus z^{[i]} = 1$ for upto $i = 12$. Beyond $i = 12$, we coould not obtain solutions in a reasonable time period and we would need further investigations.

Satisfying these equations are necessary but not sufficient to find a chain of Key-IV pairs that produce shifted key-streams. In order for valid Key-IV pairs derived from $B_0||C_0, B_1||C_1, \ldots, B_i||C_i$ to produce shifted key-streams, the first i output bits produced

by the Key-IV derived from $B_0||C_0$ during the PRGA must be zero. By randomly choosing Key-IV pairs satisfying the above conditions, it is expected that after 2^i trials one such pair will be obtained that outputs i zeros in the first i rounds of the PRGA. This is precisely the complexity of the routine needed to find a chain of i such related Key-IV pairs. Thus it improves the complexity of 2^{2i} presented in [44, Section 3].

Example 4.1. *In the Grain family of stream cipher given the Key-IV pairs in column* 1 *of the following table, column* 2 *gives a related Key-IV pair that produces shifted keystream. Column* 3 *gives the length of the shift. We could successfully obtained related Key-IV pairs for shifts upto 12. Below, we provide examples for 12-bit shifts.*

Grain	Key-IV	Key-IV	Shift
v1	`8ca87875d334c9de694a` `5246f9d65f5eaef9`	`87875d334c9de694abbc` `6f9d65f5eaef9fff`	12
128	`b8d3dac27cbfeae545a508e9e551c095` `bba4d4a0465a4448627e22ed`	`3dac27cbfeae545a508e9e551c095753` `4d4a0465a4448627e22edfff`	12

Non-applicability of such analysis on Grain-128a

It has already been pointed out in [44, Section 3.4] that such a strategy will not work if the self-similarity of the pad (initialization constant) is eliminated. Subsequently, this strategy has been implemented in Grain 128a [12]. Grain-128a resists this due to the asymmetric nature of the pad P used during the KLA. In this cipher, the pad length is 32 bits and the value of $P =$`0x ffff fffe`, i.e., it consists of 31 ones followed by a zero. Therefore after one round of KSA the last 32 bits of the LFSR may either be `0x ffff fffc` or `0x ffff fffd` depending on whether the feedback value was 0 or 1. A similar analysis for the first 32 rounds of the KSA will show that it is not possible for the last 32 bits of the LFSR to have the value $P =$`0x ffff fffe` in any of these rounds. This is because P is such that it cannot be written in the form $P_s||A$, where P_s is any s bit suffix of P and A is any $(32 - s)$-bit string over $\{0, 1\}$. It is however possible that in the 33^{rd} round of the KSA, the last 32 bits of the LFSR is equal to P. However, finding such Key-IV pairs by solving equations as above may not be possible in real time due to large degree and number of monomials in these equations. If one attempts to find such Key-IV pairs by choosing the initial states randomly, then too the complexity of the task is expected to be $(2^{32})^2 = 2^{64}$. However in Section 4.7, we will provide a way around this difficulty for Grain-128a.

4.7 Key-IV Pairs producing Shifted Keystream in Grain-128a

In [44], a method to obtain Key-IV pairs K, IV and K', IV' in Grain v1 and Grain-128, that produce ϵ-bit shifted keystream bits by performing a random experiment $2^{2\epsilon}$ times was presented. The complexity was improved to 2^{ϵ} in the work described in Section 4.6.1. Both these techniques utilized the fact that the padding P used in Grain v1 and Grain-128 was symmetric, i.e. a string of all ones. And in [44], it was suggested that the method would fail if an asymmetric padding was used. This is precisely the strategy employed in Grain-128a, where the padding is $P =$`0x ffff fffe` is a set of 31 ones followed by a single zero.

In this section, we explain how despite of the asymmetric nature of P, one can obtain related Key-IV pairs K, IV and K', IV' in Grain-128a such that they produce exactly 32-bit shifted keystream by running a random experiment 2^{32} times. We begin by noting that the state update functions in both the KSA and PRGA in the Grain family are one-to-one and invertible. This is because the state update functions of the NFSR and the LFSR can be written in the form

$$g(x_0, x_1, \ldots, x_{127}) = x_0 \oplus g'(x_1, \ldots, x_{127})$$

$$f(y_0, y_1, \ldots, y_{127}) = y_0 \oplus f'(y_1, \ldots, y_{127}).$$

This implies that one can construct the KSA^{-1} routine that takes a $2n$ bit vector S_i denoting the internal state of the cipher at any i^{th} round of the KSA, returns the $2n$ bit vector S_{i-1} denoting the internal state of the cipher at the previous round of the KSA. The same is true for the PRGA. A detailed description of the KSA^{-1} routine are given in Algorithm 4.1.

Given this information, our strategy to find related Key-IV pairs in Grain-128a will be as follows. Let $K = (k_0.k_1, k_2, \ldots, k_{127})$ be the Key. We choose a 96-bit IV of the form

$$IV = (v_0, v_1, \ldots, v_{63}, \underbrace{1, 1, \ldots, 1, 0}_{32})$$

Therefore the initial state

$$\begin{aligned} S = K||IV||P =& (s_0, s_1, \ldots, s_{255}) \\ =& (k_0, \ldots, k_{127},\ v_0, \ldots, v_{63}, \underbrace{1, 1, \ldots, 1, 0}_{32}, \underbrace{1, 1, \ldots, 1, 0}_{32}). \end{aligned}$$

If we apply the KSA^{-1} to S, 32 times, then we get the following internal state;

$$S' = (a_0, a_1, \ldots, a_{31}, k_0, k_1, \ldots, k_{95},\ b_0, b_1, \ldots, b_{31}, v_0, v_1, v_{63}, 1, \ldots, 1, 0).$$

where the values of a_i, b_i for $0 \leq i \leq 31$ are given by polynomial functions in $k_0, \ldots, k_{127}$, $v_0, \ldots, v_{63}$. The exact form of these functions can be found out by executing the KSA^{-1} routine 32 times.

Note that S' is a valid initial state for Grain-128a, since it is of the form $K'||IV'||P$, where the value of $K' = (a_0, a_1, \ldots, a_{31}, k_0, k_1, \ldots, k_{95})$ and the corresponding value of $IV' = (b_0, b_1, \ldots, b_{31}, v_0, v_1, v_{63})$. Therefore if one were to initialize Grain-128a with K', IV' then the internal state of the cipher after the KSA round $32 + t$ will be the same as the internal state after t rounds of initialization with K, IV. This would be true for all $t \leq 224$. After this, the cipher initialized with K', IV' would enter the PRGA phase while the one initialized with K, IV would still be in the KSA phase. As we have already seen, in the Grain family of ciphers, the output bit feedback to the internal state, is discontinued after the KSA. Therefore the state updates in the next 32 rounds are not guaranteed to be identical. The situation has been explained pictorially in Fig. 4.4.

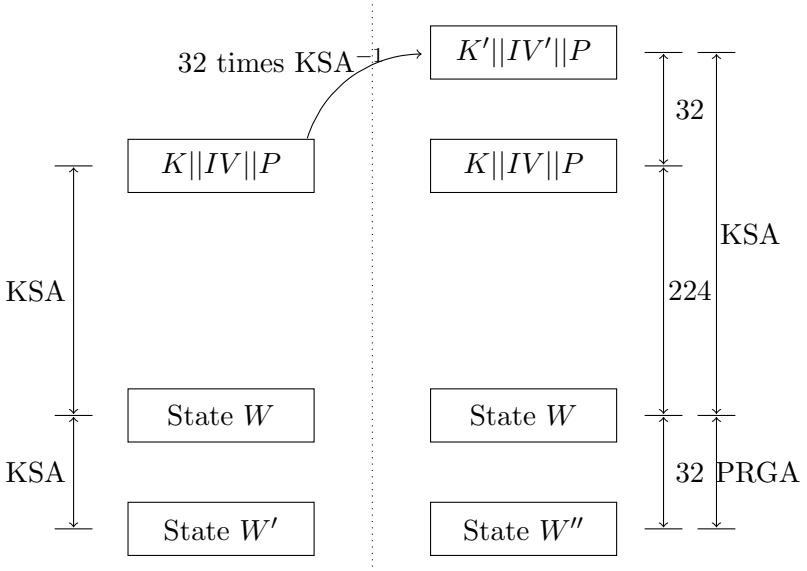


FIGURE 4.4: Construction of Related Key-IV pairs in Grain Family

For the state updates to be identical in the next 32 rounds, it is necessary and sufficient that the cipher initialized with K', IV' produces zero keystream bits for each of these 32 rounds. After this, both systems run in PRGA mode and so if the internal state of the cipher with K, IV just after the KSA is equal to the internal state of the cipher with K', IV' after 32 PRGA rounds, then they will remain the same forever thereafter. In such a situation the $(32 + t)^{th}$ PRGA state produced by K', IV' will be equal to the

t^{th} PRGA state produced by K, IV for all $t > 0$. In such a situation it is natural that K', IV' and K, IV will produce 32 bit shifted keystream bits.

Now if we choose random values of $K \in \{0,1\}^{128}$ and $IV = V||P$ with $V \in \{0,1\}^{64}$, then it is expected that in one out of 2^{32} trials we will obtain a K', IV' which produces an all zero output stream in the first 32 PRGA rounds. If so, K', IV' and K, IV will produce 32 bit shifted keystream bits. The arguments are formalized in Algorithm 4.6.

Output: Key-IV pairs K', IV' and K, IV that generate 32 bit shifted keystream

$s \leftarrow 0$;
while $s = 0$ **do**
  Choose $K \in_R \{0,1\}^{128}$, $V \in_R \{0,1\}^{64}$;
  $IV \leftarrow V||P$;
  Run $\text{KSA}^{-1}(K||IV||P)$ routine for 32 clocks and produce state $S' = (K'||IV'||P)$;
  if K', IV' *produces all zero keystream bits in the first* 32 *PRGA rounds* **then**
    $s \leftarrow 1$;
    Return (K, IV) and (K', IV');
  end
end

Algorithm 4.6: Constructing Key-IV pairs that generate 32 bit shifted keystream

Example 4.2. *In the following table, we present two Key-IV pairs that generate* 32*-bit shifted keystreams for Grain-128a. It can be seen that the second Key-IV pair has been obtained by the right shifting the first Key-IV pair by* 32 *bits. The pairs were found in around* 2^{32} *random trials using Algorithm 4.6. It should be noted that output bits given in the table includes the bits used for authentication and encryption.*

Pair	Key	IV	Output bits
1	9bbe 7e2b b99d 1477	5a7c 21e9 3a77	41d5c1f0387c
	0317 9f3b a1aa 8c70	52ce ffff fffe	3bf64e031725
2	f32a 7bd3 9bbe 7e2b	032d 0fee 5a7c	0000000041d5c1f0387c
	b99d 1477 0317 9f3b	21e9 3a77 52ce	3bf64e031725

4.7.1 Key-IV pairs producing Keystream with smaller shifts

In Section 4.6.1, the difficulty of finding Key-IV pairs in Grain-128a that produce keystream bits with small shifts was mentioned. Although we have been able to obtain Key-IV pairs producing 32 bit shifted keystream segments, one natural question arises

as to whether it is possible to obtain Key-IV pairs that produce keystream segments with shift smaller than 32 bits.

We will recap the basic idea of Section 4.6.1 that worked well for Grain v1, Grain-128. If $K_0 \in \{0,1\}^n$ and $IV_0 \in \{0,1\}^m$ denote a Key-IV (note $n = 80, 128$ and $m = 64, 96$ for Grain v1 and Grain-128 respectively), then the initial state of the KSA is denoted by $B_0 = K_0$ and $C_0 = IV_0||P$. After the first round of KSA, the updated initial states are denoted by $B_1||C_1$. Now if

(i) C_1 can be written in the form $IV_1||P$ for $IV_1 \in \{0,1\}^m$, then $B_1||C_1 = K_1||IV_1||P$ is another valid initial state of the KSA. So if the KSA starts with the state $B_1||C_1$ instead of $B_0||C_0$, it may produce one bit-shifted keystreams provided

(ii) the 1^{st} output bit produced during the PRGA must be 0 that appears from the KSA initial state B_0, C_0. This is required to ensure that the state after the $2n^{th}$ round of the KSA using $B_1||C_1$, is the same as the state after the 1^{st} PRGA round using (B_0, C_0).

If these conditions are satisfied then (K_0, IV_0) and (K_1, IV_1) will indeed produce 1-bit shifted keystreams. These events have a probability of occurrence of $\frac{1}{2}$ each and hence a related Key-IV pair could be found with probability $\frac{1}{4}$ by randomly choosing Key-IV pairs. This idea of the attack extends to r rounds, so that two Key-IV pairs which produce ϵ-bit shifted keystream could be obtained with probability $(\frac{1}{4})^\epsilon$. The probability of success could be increased to $(\frac{1}{2})^\epsilon$ by explicitly characterizing the form of $B_0||C_0$ for which the first condition **(i)** is satisfied with probability 1.

Due to the asymmetric padding used in Grain-128a, this procedure was not applicable to this cipher. However, it is also possible to obtain two Key-IV pairs in Grain-128a K_1, IV_1 and K_2, IV_2 that produce ϵ-bit shifted keystream bits (where $1 \leq \epsilon \leq 31$) by using a modification of the idea in Section 4.6.1.

We choose a random Key-IV pair K_1, IV_1 and run the KSA algorithm for 256 rounds to get the state $S_0 \in \{0,1\}^{256}$. Thereafter we run the PRGA ϵ ($0 \leq \epsilon \leq 31$) rounds to get the state S_ϵ. The KSA inverse is then run on S_ϵ to get a state S'_ϵ. If S'_ϵ is of the form $K_2||IV_2||P$ for some $K_2 \in \{0,1\}^{128}$ and $IV_2 \in \{0,1\}^{96}$, then it is easy to see that K_1, IV_1 and K_2, IV_2 will produce ϵ bit shifted keystream. Note that if the first ϵ bits produced by K_1, IV_1 are all zeros then the S'_ϵ obtained in this algorithm, will be simply the state which is obtained by loading the cipher with K_1, IV_1 and running the KSA for ϵ rounds. If $\epsilon < 32$, then due to the asymmetric padding of Grain-128a, S'_ϵ will never be of the form $K_2||IV_2||P$. So in order for the algorithm to succeed we need the following events to occur

1. The first ϵ output bits produced by K_1, IV_1 should not be all zero. This occurs with probability $1 - 2^{-\epsilon}$.

2. S'_ϵ should be of the form $K_2||IV_2||P$. This occurs with probability $2^{-l_P} = 2^{-32}$.

Hence, the success probability of the algorithm is $2^{-\epsilon} \cdot 0 + (1-2^{-\epsilon}) \cdot 2^{-32} = (1-2^{-\epsilon}) \cdot 2^{-32}$ So, in around $\frac{2^{32}}{1-2^{-\epsilon}}$ random trials we can to find these slid pairs. In the following table, we present two Key-IV pairs that generate ϵ-bit shifted keystreams (for $\epsilon = 1, 2$)

Pair	Key	IV	Shift ϵ
(i)	5b28 340e e5d0 1d6d	da8f 4f5a 811e	1
	2ede 55bb 2213 a1d3	9f39 1e9f 39ae	
(ii)	6570 e009 580e 59c2	a658 3ef4 e98a	
	5b9d 1ae3 4600 ef1b	307d 3896 1274	
(i)	aecf e876 4dd8 522c	5c53 4d90 50c4	2
	37d0 7f94 bfa6 199b	0868 c408 683f	
(ii)	f29d 3b7b afca 48d0	4eb3 52a2 d12b	
	bccb cb9d 3bdc 47f1	2362 a3ca a1ec	

4.8 Conclusion

In this chapter, we have studied a model of stream cipher where the key and IV are directly loaded in the state variables and the remaining part of the state is filled up with some kind of padding. For the Grain family, given the length of the padding l_P, it was shown that given any Key-IV one can easily construct another pair with expected 2^{l_P} time complexity that produces same bits at a significant amount of initial key-stream. With expected 2^{16} invocations of the KSA and its inverse routine, we could recover related Key-IV pairs of Grain v1 that produce key-streams with 75 identical bits out of the first 96 bits. The effect of our work on Grain-128 and Grain-128a is also similar and we could obtain related Key-IV pairs for both the ciphers within an expected complexity of 2^{32} such that the two output streams match at 112 and 115 bits out of the first 160 bits in Grain-128 and Grain-128a respectively.

Further, we have studied the related Key-IV pairs of Grain that produce shifted key-streams.

- We demonstrate how one can obtain a related Key-IV in expected 2^{l_P} trials for any given Key-IV such that the pair can generate key-streams that are finite shifts of one another. This idea works for all the versions of Grain.

- We could also construct Key-IV pairs that produce i-bit shifted key-streams in 2^i trials and our experiments work for upto $i = 12$. This is applicable for Grain v1 and Grain-128, but not for Grain-128a.

For, Grain-128a we used a slightly modified approach to obtain

- Related Key-IV pairs that generates 32 bit shifted keystream, in an expected 2^{32} random trials.
- We could also construct Key-IV pairs that produce ϵ-bit shifted key-streams using $\frac{2^{32}}{1-2^{-\epsilon}}$ random trials.

As all the algorithms described in this chapter run in time proportional to 2^{l_p}, it stands to reason that one of the ways of preventing such analysis on a design level, would have been to increase the value of l_p. This is precisely the approach used in stream ciphers like Trivium [43] and ZUC [8].

Another popular approach used to prevent such analysis altogether is the ones used in KATAN [45] and Quark [15], where update of two shift registers would be controlled by a third register which is usually initialized to a fixed constant at the start of operations. Performing this analysis on then would require a simultaneous synchronization of the third register for the related Key-IV pair, which is not possible as it always starts with a fixed constant. This of course requires extra hardware and hence increases the area and power consumption of the device implementing the cipher.

Appendix: Solution of equation system in Section 4.6.1

The solution for the system $y^{[i]} \oplus z^{[i]} = 1$ for $i = 1, \ldots, 11$ as described in Section 4.6.1 is as follows.

$$\begin{aligned} x_i &= r_{i+1} \quad 0 \leq i \leq 79 \\ y_i &= r_{i+81} \quad 0 \leq i \leq 52, \end{aligned}$$

where $r_1, r_2, \ldots, r_{133}$ are the degrees of freedom.

$$\begin{aligned}
y_{53} &= \Big((r_{108} \oplus r_{66} \oplus 1)r_{129} \oplus 1\Big)r_{86} \oplus \Big(r_{108}r_{66} \oplus r_{66} \oplus 1\Big)r_{129} \oplus r_{106} \oplus r_{108} \oplus r_{121} \oplus r_{13} \\
&\quad \oplus r_{34} \oplus r_4 \oplus r_{46} \oplus r_5 \oplus r_{59} \oplus r_7 \oplus r_{83} \oplus r_{96}
\end{aligned}$$

$$\begin{aligned}
y_{54} &= \Big((r_{109} \oplus r_{67} \oplus 1)r_{130} \oplus 1\Big)r_{87} \oplus \Big(r_{109}r_{67} \oplus r_{67} \oplus 1\Big)r_{130} \oplus r_{107} \oplus r_{109} \oplus r_{122} \oplus r_{14} \\
&\quad \oplus r_{35} \oplus r_{47} \oplus r_5 \oplus r_6 \oplus r_{60} \oplus r_8 \oplus r_{84} \oplus r_{97}
\end{aligned}$$

$$\begin{aligned}
y_{55} &= \Big((r_{110} \oplus r_{68} \oplus 1)r_{131} \oplus 1\Big)r_{88} \oplus \Big(r_{110}r_{68} \oplus r_{68} \oplus 1\Big)r_{131} \oplus r_{108} \oplus r_{110} \oplus r_{123} \oplus r_{15} \oplus \\
&\quad r_{36} \oplus r_{48} \oplus r_6 \oplus r_{61} \oplus r_7 \oplus r_{85} \oplus r_9 \oplus r_{98}
\end{aligned}$$

$$\begin{aligned}
y_{56} &= \Big((r_{111} \oplus r_{69} \oplus 1)r_{132} \oplus 1\Big)r_{89} \oplus \Big(r_{111}r_{69} \oplus r_{69} \oplus 1\Big)r_{132} \oplus r_{10} \oplus r_{109} \oplus r_{111} \oplus r_{124} \\
&\quad \oplus r_{16} \oplus r_{37} \oplus r_{49} \oplus r_{62} \oplus r_7 \oplus r_8 \oplus r_{86} \oplus r_{99}
\end{aligned}$$

$$\begin{aligned}
y_{57} &= \Big((r_{112} \oplus r_{70} \oplus 1)r_{133} \oplus 1\Big)r_{90} \oplus \Big(r_{112}r_{70} \oplus r_{70} \oplus 1\Big)r_{133} \oplus r_{100} \oplus r_{11} \oplus r_{110} \oplus r_{112} \\
&\quad \oplus r_{125} \oplus r_{17} \oplus r_{38} \oplus r_{50} \oplus r_{63} \oplus r_8 \oplus r_{87} \oplus r_9
\end{aligned}$$

$$
\begin{aligned}
y_{58} = {} & \Big[(r_{71}\oplus 1)r_{91}\oplus r_{71}\oplus 1\Big]r_{106}\oplus\Big[(r_{71}\oplus 1)r_{91}\oplus r_{71}\oplus 1\Big]r_{108}\oplus\Big[(r_{71}\oplus 1)r_{91}\oplus r_{71}\\
& \oplus 1\Big]r_{83}\oplus\Big[(r_{71}\oplus 1)r_{91}\oplus r_{71}\oplus 1\Big]r_{96}\oplus\Big[(r_{71}\oplus 1)r_{91}\oplus(r_{71}\oplus r_{91})r_{113}\oplus r_{71}\oplus 1\Big]\\
& r_{121}\oplus\Big[(r_{71}\oplus 1)r_{91}\oplus(r_{71}\oplus r_{91})r_{113}\oplus\Big((r_{66}\oplus 1)r_{71}\oplus((r_{66}\oplus 1)r_{71}\oplus r_{66}\oplus 1)r_{91}\oplus\\
& ((r_{71}\oplus 1)r_{91}\oplus r_{71}\oplus 1)r_{108}\oplus((r_{66}\oplus 1)r_{71}\oplus(r_{66}\oplus 1)r_{91}\oplus(r_{71}\oplus r_{91})r_{108})r_{113}\oplus r_{66}\\
& \oplus 1\Big)r_{129}\oplus r_{71}\oplus 1\Big]r_{86}\oplus\Big[(r_{66}\oplus 1)r_{71}\oplus\Big((r_{66}\oplus 1)r_{71}\oplus r_{66}\oplus 1\Big)r_{91}\oplus\Big((r_{66}r_{71}\\
& \oplus r_{66})r_{91}\oplus r_{66}r_{71}\oplus r_{66}\Big)r_{108}\oplus\Big((r_{66}\oplus 1)r_{71}\oplus(r_{66}\oplus 1)r_{91}\oplus(r_{66}r_{71}\oplus r_{66}r_{91})r_{108}\Big)\\
& r_{113}\oplus r_{66}\oplus 1\Big]r_{129}\oplus\Big[(r_{71}\oplus r_{91})r_{106}\oplus(r_{71}\oplus r_{91})r_{108}\oplus(r_{71}\oplus r_{91})r_{83}\oplus(r_{71}\oplus r_{91})\\
& r_{96}\oplus\Big(r_{13}\oplus r_{34}\oplus r_4\oplus r_{46}\oplus r_5\oplus r_{59}\oplus r_7\Big)r_{71}\oplus\Big(r_{13}\oplus r_{34}\oplus r_4\oplus r_{46}\oplus r_5\oplus r_{59}\\
& \oplus r_7\Big)r_{91}\oplus 1\Big]r_{113}\oplus\Big[r_{13}\oplus r_{34}\oplus r_4\oplus r_{46}\oplus r_5\oplus r_{59}\oplus r_7\Big]r_{71}\oplus\Big[\Big(r_{13}\oplus r_{34}\oplus r_4\oplus\\
& r_{46}\oplus r_5\oplus r_{59}\oplus r_7\Big)r_{71}\oplus r_{13}\oplus r_{34}\oplus r_4\oplus r_{46}\oplus r_5\oplus r_{59}\oplus r_7\oplus 1\Big]r_{91}\oplus r_{10}\oplus r_{101}\oplus r_{111}\\
& \oplus r_{12}\oplus r_{126}\oplus r_{13}\oplus r_{18}\oplus r_{34}\oplus r_{39}\oplus r_4\oplus r_{46}\oplus r_5\oplus r_{51}\oplus r_{59}\oplus r_{64}\oplus r_7\oplus r_{88}\oplus r_9
\end{aligned}
$$

$$
\begin{aligned}
y_{59} = {} & \Big[(r_{72}\oplus 1)r_{92}\oplus r_{72}\oplus 1\Big]r_{107}\oplus\Big[(r_{72}\oplus 1)r_{92}\oplus r_{72}\oplus 1\Big]r_{109}\oplus\Big[(r_{72}\oplus 1)r_{92}\oplus r_{72}\oplus 1\Big]\\
& r_{84}\oplus\Big[(r_{72}\oplus 1)r_{92}\oplus r_{72}\oplus 1\Big]r_{97}\oplus\Big[(r_{72}\oplus 1)r_{92}\oplus(r_{72}\oplus r_{92})r_{114}\oplus r_{72}\oplus 1\Big]r_{122}\oplus\\
& \Big[(r_{72}\oplus 1)r_{92}\oplus(r_{72}\oplus r_{92})r_{114}\oplus\Big((r_{67}\oplus 1)r_{72}\oplus((r_{67}\oplus 1)r_{72}\oplus r_{67}\oplus 1)r_{92}\oplus((r_{72}\oplus 1)\\
& r_{92}\oplus r_{72}\oplus 1)r_{109}\oplus\big((r_{67}\oplus 1)r_{72}\oplus(r_{67}\oplus 1)r_{92}\oplus(r_{72}\oplus r_{92})r_{109}\big)r_{114}\oplus r_{67}\oplus 1\Big)r_{130}\\
& \oplus r_{72}\oplus 1\Big]r_{87}\oplus\Big[(r_{67}\oplus 1)r_{72}\oplus((r_{67}\oplus 1)r_{72}\oplus r_{67}\oplus 1)r_{92}\oplus\Big((r_{67}r_{72}\oplus r_{67})r_{92}\oplus r_{67}\\
& r_{72}\oplus r_{67}\Big)r_{109}\oplus\Big((r_{67}\oplus 1)r_{72}\oplus(r_{67}\oplus 1)r_{92}\oplus(r_{67}r_{72}\oplus r_{67}r_{92})r_{109}\Big)r_{114}\oplus r_{67}\oplus 1\Big]\\
& r_{130}\oplus\Big[(r_{72}\oplus r_{92})r_{107}\oplus(r_{72}\oplus r_{92})r_{109}\oplus(r_{72}\oplus r_{92})r_{84}\oplus(r_{72}\oplus r_{92})r_{97}\oplus\Big(r_{14}\oplus r_{35}\\
& \oplus r_{47}\oplus r_5\oplus r_6\oplus r_{60}\oplus r_8\Big)r_{72}\oplus\Big(r_{14}\oplus r_{35}\oplus r_{47}\oplus r_5\oplus r_6\oplus r_{60}\oplus r_8\Big)r_{92}\oplus 1\Big]r_{114}\\
& \oplus\Big[r_{14}\oplus r_{35}\oplus r_{47}\oplus r_5\oplus r_6\oplus r_{60}\oplus r_8\Big]r_{72}\oplus\Big[(r_{14}\oplus r_{35}\oplus r_{47}\oplus r_5\oplus r_6\oplus r_{60}\oplus r_8)r_{72}\\
& \oplus r_{14}\oplus r_{35}\oplus r_{47}\oplus r_5\oplus r_6\oplus r_{60}\oplus r_8\oplus 1\Big]r_{92}\oplus r_{10}\oplus r_{102}\oplus r_{11}\oplus r_{112}\oplus r_{127}\oplus r_{13}\oplus\\
& r_{14}\oplus r_{19}\oplus r_{35}\oplus r_{40}\oplus r_{47}\oplus r_5\oplus r_{52}\oplus r_6\oplus r_{60}\oplus r_{65}\oplus r_8\oplus r_{89}
\end{aligned}
$$

$$
\begin{aligned}
y_{60} = {} & (r_{73}\oplus 1)r_9\oplus\Big[(r_{73}\oplus 1)r_{93}\oplus r_{73}\oplus 1\Big]r_{108}\oplus\Big[(r_{73}\oplus 1)r_{93}\oplus r_{73}\oplus 1\Big]r_{110}\oplus\Big[(r_{73}\oplus 1)\\
& r_{93}\oplus r_{73}\oplus 1\Big]r_{98}\oplus\Big[(r_{73}\oplus 1)r_{93}\oplus(r_{73}\oplus r_{93})r_{115}\oplus r_{73}\oplus 1\Big]r_{123}\oplus\Big[(r_{73}\oplus 1)r_{93}\oplus\\
& (r_{73}\oplus r_{93})r_{115}\oplus r_{73}\oplus 1\Big]r_{85}\oplus\Big[(r_{73}\oplus 1)r_{93}\oplus(r_{73}\oplus r_{93})r_{115}\oplus\Big((r_{68}\oplus 1)r_{73}\oplus((r_{68}\oplus\\
& 1)r_{73}\oplus r_{68}\oplus 1)r_{93}\oplus((r_{73}\oplus 1)r_{93}\oplus r_{73}\oplus 1)r_{110}\oplus\big((r_{68}\oplus 1)r_{73}\oplus(r_{68}\oplus 1)r_{93}\oplus(r_{73}\\
& \oplus r_{93})r_{110}\big)r_{115}\oplus r_{68}\oplus 1\Big)r_{131}\oplus r_{73}\oplus 1\Big]r_{88}\oplus\Big[(r_{68}\oplus 1)r_{73}\oplus((r_{68}\oplus 1)r_{73}\oplus r_{68}\oplus 1)r_{93}\\
& \oplus\Big((r_{68}r_{73}\oplus r_{68})r_{93}\oplus r_{68}r_{73}\oplus r_{68}\Big)r_{110}\oplus\Big((r_{68}\oplus 1)r_{73}\oplus(r_{68}\oplus 1)r_{93}\oplus(r_{68}r_{73}\oplus r_{68}\\
& r_{93})r_{110}\Big)r_{115}\oplus r_{68}\oplus 1\Big]r_{131}\oplus\Big[r_{15}\oplus r_{36}\oplus r_{48}\oplus r_6\oplus r_{61}\oplus r_7\Big]r_{73}\oplus\Big[(r_{73}\oplus r_{93})\\
& r_{108}\oplus(r_{73}\oplus r_{93})r_{110}\oplus(r_{73}\oplus r_{93})r_{98}\oplus\Big(r_{15}\oplus r_{36}\oplus r_{48}\oplus r_6\oplus r_{61}\oplus r_7\Big)r_{73}\oplus\Big(r_{15}\\
& \oplus r_{36}\oplus r_{48}\oplus r_6\oplus r_{61}\oplus r_7\oplus r_9\Big)r_{93}\oplus r_{73}r_9\oplus 1\Big]r_{115}\oplus\Big[(r_{73}\oplus 1)r_9\oplus\Big(r_{15}\oplus r_{36}\oplus\\
& r_{48}\oplus r_6\oplus r_{61}\oplus r_7\Big)r_{73}\oplus r_{15}\oplus r_{36}\oplus r_{48}\oplus r_6\oplus r_{61}\oplus r_7\oplus 1\Big]r_{93}\oplus r_{103}\oplus r_{11}\oplus r_{113}\oplus\\
& r_{12}\oplus r_{128}\oplus r_{14}\oplus r_{15}\oplus r_{20}\oplus r_{36}\oplus r_{41}\oplus r_{48}\oplus r_{53}\oplus r_6\oplus r_{61}\oplus r_{66}\oplus r_7\oplus r_{90}
\end{aligned}
$$

$$
\begin{aligned}
y_{61} = {} & (r_{74} \oplus 1)r_{10} \oplus \Big[(r_{74} \oplus 1)r_{94} \oplus r_{74} \oplus 1\Big]r_{109} \oplus \Big[(r_{74} \oplus 1)r_{94} \oplus r_{74} \oplus 1\Big]r_{111} \oplus \Big[(r_{74} \oplus 1)r_{94} \\
& \oplus r_{74} \oplus 1\Big]r_{99} \oplus \Big[(r_{74} \oplus 1)r_{94} \oplus (r_{74} \oplus r_{94})r_{116} \oplus r_{74} \oplus 1\Big]r_{124} \oplus \Big[(r_{74} \oplus 1)r_{94} \oplus (r_{74} \oplus r_{94}) \\
& r_{116} \oplus r_{74} \oplus 1\Big]r_{86} \oplus \Big[(r_{74} \oplus 1)r_{94} \oplus (r_{74} \oplus r_{94})r_{116} \oplus \Big((r_{69} \oplus 1)r_{74} \oplus ((r_{69} \oplus 1)r_{74} \oplus r_{69} \\
& \oplus 1)r_{94} \oplus ((r_{74} \oplus 1)r_{94} \oplus r_{74} \oplus 1)r_{111} \oplus \big((r_{69} \oplus 1)r_{74} \oplus (r_{69} \oplus 1)r_{94} \oplus (r_{74} \oplus r_{94})r_{111}\big)r_{116} \\
& \oplus r_{69} \oplus 1\Big)r_{132} \oplus r_{74} \oplus 1\Big]r_{89} \oplus \Big[(r_{69} \oplus 1)r_{74} \oplus ((r_{69} \oplus 1)r_{74} \oplus r_{69} \oplus 1)r_{94} \oplus \Big((r_{69}r_{74} \oplus r_{69}) \\
& r_{94} \oplus r_{69}r_{74} \oplus r_{69}\Big)r_{111} \oplus \Big(r_{69} \oplus 1)r_{74} \oplus (r_{69} \oplus 1)r_{94} \oplus (r_{69}r_{74} \oplus r_{69}r_{94})r_{111}\Big)r_{116} \oplus r_{69} \\
& \oplus 1\Big]r_{132} \oplus \Big[r_{16} \oplus r_{37} \oplus r_{49} \oplus r_{62} \oplus r_7 \oplus r_8\Big]r_{74} \oplus \Big[(r_{74} \oplus r_{94})r_{109} \oplus (r_{74} \oplus r_{94})r_{111} \oplus \\
& (r_{74} \oplus r_{94})r_{99} \oplus \Big(r_{16} \oplus r_{37} \oplus r_{49} \oplus r_{62} \oplus r_7 \oplus r_8\Big)r_{74} \oplus \Big(r_{10} \oplus r_{16} \oplus r_{37} \oplus r_{49} \oplus r_{62} \\
& \oplus r_7 \oplus r_8\Big)r_{94} \oplus r_{10}r_{74} \oplus 1\Big]r_{116} \oplus \Big[r_{74} \oplus 1)r_{10} \oplus \Big(r_{16} \oplus r_{37} \oplus r_{49} \oplus r_{62} \oplus r_7 \oplus r_8\Big)r_{74} \\
& \oplus r_{16} \oplus r_{37} \oplus r_{49} \oplus r_{62} \oplus r_7 \oplus r_8 \oplus 1\Big]r_{94} \oplus r_{104} \oplus r_{114} \oplus r_{12} \oplus r_{129} \oplus r_{13} \oplus r_{15} \oplus r_{16} \oplus r_{21} \\
& \oplus r_{37} \oplus r_{42} \oplus r_{49} \oplus r_{54} \oplus r_{62} \oplus r_{67} \oplus r_7 \oplus r_8 \oplus r_{91}
\end{aligned}
$$

$$
\begin{aligned}
y_{62} = {} & \Big(r_{106} \oplus r_{64} \oplus 1)r_{84} \oplus r_{106}r_{64} \oplus r_{64} \oplus 1\Big)r_{127} \oplus r_{104} \oplus r_{106} \oplus r_{11} \oplus r_{119} \oplus r_{132} \oplus r_2 \oplus r_3 \oplus \\
& r_{32} \oplus r_{44} \oplus r_5 \oplus r_{57} \oplus r_{81} \oplus r_{84} \oplus r_{94} \oplus 1
\end{aligned}
$$

$$
\begin{aligned}
y_{63} = {} & \Big((r_{107} \oplus r_{65} \oplus 1)r_{85} \oplus r_{107}r_{65} \oplus r_{65} \oplus 1\Big)r_{128} \oplus r_{105} \oplus r_{107} \oplus r_{12} \oplus r_{120} \oplus r_{133} \oplus r_3 \oplus r_{33} \\
& \oplus r_4 \oplus r_{45} \oplus r_{58} \oplus r_6 \oplus r_{82} \oplus r_{85} \oplus r_{95} \oplus 1
\end{aligned}
$$

Chapter 5

Differential Fault Analysis of Grain

In this chapter, we will study a differential fault attack (DFA) against the Grain family of stream ciphers. We will look at a set of three attacks on the Grain family of stream ciphers, each of which is mounted under different experimental setups in which the attacker is granted varying degrees of freedom. We have already seen in Chapter 2 that most fault attacks require an adversary to physically alter the logic at some random register location of a given cryptographic device. We will begin with an adversary who exercises maximum control over the precision and timing of faults and then build up to the case where the adversary requires least control over fault injections. In this way, in some sense, we will strengthen the nature of attack over the Grain family. It would be worthwhile to mention, that all the attacks work due to certain properties of the output functions and corresponding choices of the LFSR taps employed in the Grain family. More specifically we have used certain cryptographic properties of the derivatives of the output function h used in the Grain family. In each of the attack techniques, we will first present methods to identify the register locations of a randomly applied fault and then construct set of linear/non-linear equations to obtain the contents of the LFSR and the NFSR.

5.1 Introduction

Fault attacks have received serious attention in cryptographic literature for more than a decade [37, 41]. Such attacks have successfully cryptanalyzed block ciphers like Advanced Encryption Standard and the Digital Encryption Standard, Public key cryptosystems like implementations of CRT-RSA, DSA/RSA based signatures, Elliptic Curve

Cryptosystems (please refer to [83] for an extensive analysis of all such attacks). Fault attacks on stream ciphers have gained momentum ever since the work of Hoch and Shamir [76] and this model of cryptanalysis, though optimistic, has successfully been employed against a number of proposals. Fault attacks study the mathematical robustness of a cryptosystem in a setting that is weaker than its original or expected mode of operation. A typical attack scenario consists of an adversary who injects a random fault (using laser shots/clock glitches or optical faults using a camera flashgun [124, 125]) in a cryptographic device as a result of which one or more bits of its internal state are altered. The faulty output from this altered device is then used to deduce information about its internal state/secret key. In order to perform the attack, the adversary requires certain privileges like the ability to re-key the device, control the timing and/or location of the fault etc. The more privileges the adversary is granted, the more the attack becomes impractical and unrealistic.

The Grain family of stream ciphers [12, 73, 74] has received a lot of attention from the cryptological community as it is in the hardware profile of eStream [116]. Prior to the publication of the series of works related to the differential fault analysis of the Grain family described in this chapter, two results on the fault analysis of Grain-128 had already been published [35, 85]. Both these attacks were performed under a similar experimental setup. We will begin by briefly discussing the setup.

- The attacker is in possession of the physical device in which the stream cipher has been implemented. He therefore, knows the IV and the keystream generated by the stream cipher. Additionally, in [85], the attacker is assumed to be able to derive keystream sequences off the cipher by inputing IV's of his own choice.
- By applying optical faults, the attacker is assumed to be able to apply bit-flipping faults with a partial control of their location in te register. Now, in order to this, the attacker uses a dummy device that is architecturally similar or close to the target device. During a preliminary fault setup stage, he attacks the dummy device. He scans this device by injecting faults on different areas and analyzes the corresponding faulty outputs. Finally the attacker replaces the test device with the target device and attempts to inject faults with respect to the previous setup. The number of additional adjustments is as small as the test device is close to the target device.
- In this way, the attacker is able only partially able to control the register location where he injects a bit-flipping fault. That is to say, he can not exactly choose the location of the fault. But once he is able to fault any particular location, he is able to inject faults on that location over and over again without restriction.

- The attacker is able to exercise full control over the timing of application of faults, by choosing the time of the fault trigger by synchronizing it with the I/O signal. Shift registers are regularly clocked, and one keystream bit is computed per clock cycle. Hence, the attacker can identify steps in the execution, and so it is possible for the attacker to inject a fault at any particular point of the operation. Another popular way of controlling the fault timing would be by synchronizing it with the power consumption curves of the device implementing the cryptosystem [52]. This method can be used when there is no possible way to trigger the faults using I/O signals of the device.
- The attacker from time to time resets the cryptographic device to its original state.

To sum up, the attacker is able to flip exactly one bit lying in some register location without choosing its location but at a chosen point in time. He then has the option of either re-injecting the fault at the same location, or resetting the device to apply a fresh fault at some other random register location. In addition, [35] assumes that the attacker is able to restrict its faults to the LFSR of Grain-128, whereas [85] assumes that the faults are restricted to the NFSR. However, both the attacks described in [35, 85] are applicable only to Grain-128, whose algebraic structure is arguably the simplest in the Grain family. The output function h of Grain-128, is the sum of a quadratic bent function and a degree 3 monomial, whereas the NFSR update function g is the sum of a simple quadratic bent function and some linear terms. This is a lot simpler as compared to Grain v1 which has a complicated degree 3 output function h and an NFSR update function g with an algebraic degree of 6. The fact that the methodology of [35, 85] can not be extended towards Grain v1 or Grain-128a was the primary motivation to pursue research in this area.

5.1.1 Fault Attacks on other Stream Ciphers

We have already known about two fault attacks on Grain-128 [35, 85] and the Differential and Impossible fault attack on RC4 [38]. Apart from these, there has been extensive fault analysis on most of the ciphers in final portfolio of eStream. We list some of them below.

Trivium A differential fault attack of Trivium was first published by Hojs'ık and Rudolf in [78]. The attack required around 43 random faults on an average and took advantage of the relatively simple quadratic output functions of the cipher. This attack was later improved by the same authors in [79] using a floating model of the cipher. The attack now required only around 3.2 faults on average. In [109],

SAT solvers were first used to propose an improved fault attack that required as little as 2 faults to cryptanalyze the cipher. In [81], the cipher is attacked under a more restrictive fault model that does not assume that the attacker can synchronize the timing of fault injection.

Sosemanuk A fault attack on Sosemanuk was first presented in [119]. The attack, which recovers the secret inner state of the cipher, requires around 6144 faults, and a time complexity equivalent to around 2^{48} Sosemanuk iterations and a storage of around $2^{38.17}$ bytes. The attack was later improved in [99]. The new attack required 4608 faults, time complexity of $2^{35.16}$ Sosemanuk iterations and $2^{23.46}$ bytes of storage.

Rabbit A fault attack on Rabbit was first presented in [91]. The attack requires around $128-256$ faults, a precomputed table of size $2^{41.6}$ bytes and recovers the complete internal state of Rabbit in about 2^{38} steps. The attack was later improved in [36]. In this attack, it was shown that by modifying modular additions of the next-state function, 32 faulty outputs were enough for recovering the whole internal state of the cipher in $O(2^{34})$ iterations.

HC-128 A fault attack on HC-128 was first presented in [90].To perform the attack, the authors exploit the fact that some of the inner state words in HC-128 may be utilized several times without being updated. The attack requires about 7968 faults and recovers the complete internal state of HC-128 by solving a set of 32 systems of linear equations over $GF(2)$ in 1024 variables.

5.1.2 Our Results

In the remainder of this chapter we will show how an attacker with varying degrees of freedom can approach the problem of performing a fault attack on all the three ciphers of the Grain family. In Section 5.3, we will begin with a powerful attacker who in addition to a dummy device also possesses the ability to synchronize and control the timing of his faults, so that he is able to fault exactly one random register location over and over again. Then, in Section 5.4, we will explore a more realistic situation in which the attacker does not possess the ability to fault any random register location more than once. Finally, in Section 5.4, we present our strongest attack yet on the Grain family in which

- we show a drastic reduction in the number of faults required to perform the attack and

- explore the situation when a resource-constrained attacker is

  1. unable to synchronize the timing of fault injection.
  2. unable to guarantee that an injected fault flips the logic at only a single register location. We consider the case when an applied fault may with a uniform probability distribution, affect the logic values at upto 3 consecutive register locations.

Also, in all our attacks we assume that the attacker is unable to restrict fault injections to either only the LFSR as in [35] or only the NFSR as in [85]. We assume that a randomly injected fault has has equal probability of hitting either the LFSR or the NFSR. This to the best of our knowledge is the best and the most realistic fault attack reported against the Grain family.

5.2 Obtaining the Location of the Fault

Central to the attack procedures that we shall describe, is the ability of the attacker to deduce the location of a randomly injected fault by simply examining the faulty keystream produced as a result of it (we will consider the single bit-flip model for the time being and build up to the multi-bit flip model later in the chapter). Unless otherwise stated, we will assume that the attacker injects the fault just at the **start of the PRGA** (we will consider the case of unsynchronized faults later in the chapter). Now, in order to deduce the fault location, we will define a set of Signature vectors for every register location in the cipher and compare the difference of the faulty and fault-free keystreams with these vectors. Thereafter, after a process of matching and elimination the attacker would be able, with high probability, conclusively able to determine the location of the injected fault.

For example, assume that the attacker injects a fault at the beginning of the PRGA stage of Grain v1. Suppose, that the attacker wants to preclude the possibility that the fault has been injected in the 79^{th} location of the LFSR. Let $S_0, S_{0,\Delta_{79}} \in \{0,1\}^{160}$ be the initial states of the PRGA which differ only in the 79^{th} LFSR position. Then, we have already seen in Theorem 4.1, that the 2 states produce identical output bits in 68 specific positions among the initial 80 keystream bits produced during the PRGA. That is to say, if an input differential is introduced in the 79^{th} LFSR position via a bit-flipping fault, then at all rounds numbered

$$k \in [0, 79] \setminus \{15, 33, 44, 51, 54, 57, 62, 69, 72, 73, 75, 76\},$$

the difference exists in positions that do not provide input to the Boolean function h and hence at these clocks the keystream bit produced by the faulty and faultless states are essentially the same. At all other clock rounds the difference appears at positions which provide input to h. Hence the keystream produced at these clocks may be different. So, if the attacker observes that 15^{th} faulty and fault-free keystream bits are different, he can immediately conclude that the fault could not have been injected in the 79^{th} LFSR location. Note, that the attacker needs to do a similar analysis for all the 160 register locations and then come to a conclusion about the actual fault location. In order to do that one must devise a generalization of the above ideas, which we proceed to describe next.

5.2.1 Differential Grain

In this subsection, we will define a tool to compute the differential trails of any cipher in the Grain family. Therefore let us define a generalized1 Grain stream cipher which will cover the descriptions of Grain v1, Grain-128 and Grain-128a as well. We already know that any cipher in the Grain family consists of an n-bit LFSR and an n-bit NFSR (see Figure 4.1). The update function of the LFSR is given by the equation

$$y_{t+n} = f(Y_t) = y_t \oplus y_{t+f_1} \oplus y_{t+f_2} \oplus \cdots \oplus y_{t+f_a},$$

where $Y_t = [y_t, y_{t+1}, \ldots, y_{t+n-1}]$ is an n-bit vector that denotes the LFSR state at the t^{th} clock interval and f is a linear function on the LFSR state bits obtained from a primitive polynomial in $GF(2)$ of degree n. The NFSR state is updated as

$$x_{t+n} = y_t \oplus g(X_t) = y_t \oplus g(x_t, x_{t+g_1}, x_{t+g_2}, \ldots, x_{t+g_b}).$$

Here, $X_t = [x_t, x_{t+1}, \ldots, x_{t+n-1}]$ is an n-bit vector that denotes the NFSR state at the t^{th} clock interval and g is a non-linear function of the NFSR state bits. The output key-stream is produced by combining the LFSR and NFSR bits as

$$z_t = x_{t+l_1} \oplus x_{t+l_2} \oplus \cdots \oplus x_{t+l_c} \oplus y_{t+i_1} \oplus y_{t+i_2} \oplus \cdots \oplus y_{t+i_d} \oplus$$
$$h(y_{t+h_1}, y_{t+h_2}, \ldots, y_{t+h_e}, x_{t+j_1}, x_{t+j_2}, \ldots, x_{t+j_w}).$$

Here h is another non-linear combining Boolean function. So it is clear that Grain v1, Grain-128 and Grain-128a are particular instances of the generalized Grain cipher.

Let $S_0 = [X_0||Y_0] \in \{0,1\}^{2n}$ be the initial state of the generalized Grain PRGA and S_{0,Δ_ϕ} be the initial state which differs from S_0 in some register location $\phi \in [0, 2n-1]$.

Now, if $\phi \in [0, n-1]$, we will consider ϕ to be the ϕ^{th} LFSR location, and if $\phi \in [n, 2n-1]$, we will consider ϕ to be the $(\phi - n)^{th}$ NFSR location.

The task is to ascertain how the corresponding internal states in the t^{th} round S_t and S_{t,Δ_ϕ} will differ from each other, for some integer $t > 0$. One such tool appeared in [35], but our approach is improved and more involved. We present the following algorithm which we will call D-Grain that takes as input the difference location $\phi \in [0, n-1]$ and the round r, and returns

(i) a set of r integer arrays χ_t, for $0 \leq t < r$, each of length $c + d$,

(ii) a set of r integer arrays Υ_t, for $0 \leq t < r$, each of length $e + w$ and

(iii) an integer array ΔZ of length r.

Note that as already defined in the description of generalized Grain, d, c are the number of LFSR, NFSR bits which are linearly added to the output function h. And e, w are the number of LFSR, NFSR bits that are input to the function h.

Now consider the corresponding generalized differential engine Δ_ϕ-Grain with an n-cell LFSR ΔL and an n-cell NFSR ΔN. All the elements of ΔL and ΔN are integers. We will denote the t^{th} round state of ΔL as $\Delta L_t = [u_t, u_{t+1}, \ldots, u_{t+n-1}]$ and that of ΔN as $\Delta N_t = [v_t, v_{t+1}, \ldots, v_{t+n-1}]$. Initially all the elements of $\Delta N, \Delta L$ are set to 0, with the only exception that

- If $\phi \in [0, n-1]$, then the cell numbered ϕ of ΔL is set to 1.
- Else if $\phi \in [n, 2n-1]$, then the cell numbered $\phi - n$ of ΔN is set to 1.

The initial states $\Delta N_0, \Delta L_0$ are indicative of the difference between S_0 and S_{0,Δ_ϕ} and we will show that the t^{th} states $\Delta N_t, \Delta L_t$ are indicative of the difference between S_t and S_{t,Δ_ϕ}. ΔL updates itself as

$$u_{t+n} = u_t + u_{t+f_1} + u_{t+f_2} + \cdots + u_{t+f_a} \bmod 2$$

and ΔN updates itself as

$$v_{t+n} = u_t + 2 \cdot OR(v_t, v_{t+g_1}, v_{t+g_2}, \ldots, v_{t+g_b}).$$

The rationale behind the update functions will be explained later. Here OR is a map from $\mathbb{Z}^{b+1} \to \{0, 1\}$ which roughly represents the logical 'or' operation and is defined as

$$OR(k_0, k_1, \ldots, k_b) = \begin{cases} 0, & \text{if } k_0 = k_1 = k_2 = \cdots = k_b = 0, \\ 1, & \text{otherwise.} \end{cases}$$

Let

$$\chi_t = [v_{t+l_1}, v_{t+l_2}, \ldots, v_{t+l_c}, u_{t+i_1}, u_{t+i_2}, \ldots, u_{t+i_d}]$$

$$\Upsilon_t = [u_{t+h_1}, u_{t+h_2}, \ldots, u_{t+h_e}, v_{t+j_1}, v_{t+j_2}, \ldots, v_{t+j_w}].$$

Note that $\chi_t(\Upsilon_t)$ is the set of cells in Δ_ϕ-Grain which corresponds to the bits which are linearly added to the output function h (input to h) in the t^{th} PRGA stage of the actual cipher.

If $\mathbf{V}$ is a vector having non-negative integral elements, then $\mathbf{V} \sqsubseteq \beta$, (for some positive integer β), implies that all elements of $\mathbf{V}$ are less than or equal to β. The t^{th} key-stream element Δz_t produced by this engine is given as

$$\Delta z_t = \begin{cases} 0, & \text{if } \Upsilon_t = \mathbf{0} \text{ AND } \chi_t \sqsubseteq 1 \text{ AND } \parallel \chi_t \parallel \text{ is even} \\ 1, & \text{if } \Upsilon_t = \mathbf{0} \text{ AND } \chi_t \sqsubseteq 1 \text{ AND } \parallel \chi_t \parallel \text{ is odd} \\ 2, & \text{otherwise.} \end{cases}$$

Here $\mathbf{0}$ denotes the all zero vector, and $\parallel \cdot \parallel$, as defined in Section 2.1.2, denotes the number of non-zero elements in a vector. Initially $\Delta N_0, \Delta L_0$ represent the difference of S_0 and S_{0,Δ_ϕ}. As the PRGA evolves, the only non-zero element (having value 1) of ΔL propagates and so does the difference between S_t and S_{t,Δ_ϕ}. Since the LFSR in Grain is updated by a linear function, whenever the difference between S_t and S_{t,Δ_ϕ} is fed back via the update function, a 1 is fed back in ΔL. Now when the difference between S_t and S_{t,Δ_ϕ} propagates to some NFSR tap location g_i (for some value of t), then this difference may or may not be fed back, depending on the nature of the Boolean function g and the current state S_t. Hence in such a case the propagation of the differential is probabilistic. Note that in all such situations, either the integer 2 or 3 is fed back in ΔN as is apparent from the update equation for v_{t+n}. Therefore whenever

1. some cell in ΔL_t or ΔN_t is 0, it implies that the corresponding bits are equal in S_t and S_{t,Δ_ϕ} with probability 1;
2. some cell in ΔL_t or ΔN_t is 1, it implies that the corresponding bits are different in S_t and S_{t,Δ_ϕ} with probability 1;
3. some cell in ΔL_t or ΔN_t is 2 or 3, it implies that the corresponding bits are different in S_t and S_{t,Δ_ϕ} with some probability $0 < p_d < 1$.

Also, note that whenever Υ_t is $\mathbf{0}$, it implies that all the bits of S_t and S_{t,Δ_ϕ} that provide inputs to the non-linear function h are the same (for all choices of S_0). Whenever all elements of χ_t are less than or equal to 1, it implies that each one of the elements of S_t

and S_{t,Δ_ϕ} which linearly adds on to the output function h to produce the output key-stream bit is either equal or different with probability 1. When both these events occur, the key-stream bits produced by S_t and S_{t,Δ_ϕ} are definitely the same if $\| \chi_t \|$ is an even number, as an even number of differences cancel out in GF(2). When this happens, Δ_ϕ-Grain outputs $\Delta z_t = 0$. If $\| \chi_t \|$ is an odd number, then the key-stream bits produced by S_t and S_{t,Δ_ϕ} are different with probability 1. In this case $\Delta z_t = 1$. In all other cases, the difference of the key-stream bits produced by S_t and S_{t,Δ_ϕ} is equal to 0 or 1 with some probability, and then $\Delta z_t = 2$. We describe the routine D-Grain(ϕ, r) in Algorithm 5.1 which returns the arrays χ_t, Υ_t for $0 \leq t < r$ and $\Delta Z = [\Delta z_0, \ldots, \Delta z_{r-1}]$.

5.2.2 The routine FLocl(E_ϕ)

Let $S_0 \in \{0,1\}^{2n}$ be the initial state of the Grain family PRGA described in Section 5.2.1 and S_{0,Δ_ϕ} be the initial state resulting after injecting a single bit fault in some register location $\phi \in [0, 2n-1]$ at the beginning of the PRGA. Let $Z = [z_0, z_1, \ldots, z_{2n-1}]$ and $Z^\phi = [z_0^\phi, z_1^\phi, \ldots, z_{2n-1}^\phi]$ be the first $2n$ key-stream bits produced by S_0 and S_{0,Δ_ϕ} respectively. Define a $2n$ bit vector E_ϕ over GF(2) defined as follows. Let E_ϕ be the bitwise logical XNOR (complement of XOR) of Z and Z_ϕ, i.e.,

$$E_\phi = 1 \oplus Z \oplus Z^\phi.$$

Similarly we define $\overline{E}_\phi = 1 \oplus E_\phi$. We will now describe the fault location identification routine FLocl(E_ϕ) the task of which is to determine the fault location ϕ by analyzing the vector E_ϕ.

Note that, in Grain-128a due to the provision of optional authentication, the entire Z and Z_ϕ may not be available to the attacker. Thus, we will deal with the case of Grain-128a separately.

Grain v1 and Grain-128 Since S_0 can have 2^{n+m} values (each arising from a different combination of the n bit key and m bit IV, the remaining $n-m$ padding bits are fixed), each of these choices of S_0 may lead to different patterns of E_ϕ. The bitwise logical AND of all such vectors E_ϕ is denoted as the First Signature vector $\mathcal{Q}_\phi^1$ for the fault location ϕ. Similarly the bitwise logical AND of all such vectors $\overline{E}_\phi$ is denoted as the Second Signature vector $\mathcal{Q}_\phi^2$ for the fault location ϕ. Note that whenever $\mathcal{Q}_\phi^1(i)$ $(\mathcal{Q}_\phi^2(i))$ is 1 for any $i \in [0, 2n-1]$ this implies that the i^{th} key-stream bit produced by S_0 and S_{0,Δ_ϕ} is equal (different) for all choices of S_0.

Input: ϕ: An fault location $\in [0, 2n-1]$, an integer $r(>0)$;

Output: An integer array ΔZ of r elements;

Output: Two integer arrays χ_t, Υ_t for $0 \leq t < r$;

$[u_0, u_1, \ldots, u_{n-1}] \leftarrow \mathbf{0}, [v_0, v_1, \ldots, v_{n-1}] \leftarrow \mathbf{0}$;

$t \leftarrow 0$;

if $\phi \in [0, n-1]$ **then**
  $u_\phi = 1$;
end
else
  $v_{\phi-n} = 1$;
end

while $t < r$ **do**
  /* The modification for Proof of Lemma 5.1 goes here */;
  $\Upsilon_t \leftarrow [u_{h_1}, u_{h_2}, \ldots, u_{h_e}, v_{j_1}, v_{j_2}, \ldots, v_{j_w}]$;
  $\chi_t \leftarrow [v_{l_1}, v_{l_2}, \ldots, v_{l_c}, u_{i_1}, u_{i_2}, \ldots, u_{i_d}]$;
  if $\Upsilon_t = \mathbf{0}$ *AND* $\chi_t \sqsubseteq 1$ **then**
    if $|\chi_t|$ *is EVEN* **then**
      $\Delta z_t \leftarrow 0$;
    end
    if $|\chi_t|$ *is ODD* **then**
      $\Delta z_t \leftarrow 1$;
    end
  end
  else
    $\Delta z_t \leftarrow 2$;
  end
  $t_1 \leftarrow u_0 + u_{f_1} + u_{f_2} + \ldots + u_{f_a} \bmod 2$;
  $t_2 \leftarrow u_0 + 2 \cdot OR(v_0, v_{g_1}, v_{g_2}, \ldots, v_{g_b})$;
  $[u_0, u_1, \ldots, u_{n-2}, u_{n-1}] \leftarrow [u_1, u_2, \ldots, u_{n-1}, t_1]$;
  $[v_0, v_1, \ldots, v_{n-2}, v_{n-1}] \leftarrow [v_1, v_2, \ldots, v_{n-1}, t_2]$;
  $t = t + 1$;
end
$\Delta Z = [\Delta z_0, \Delta z_1, \ldots, \Delta z_{r-1}]$;
Return $[\chi_0, \chi_1, \ldots, \chi_{r-1}]$, $[\Upsilon_0, \Upsilon_1, \ldots, \Upsilon_{r-1}]$, ΔZ

Algorithm 5.1: D-GRAIN(ϕ, r)

This implies that if $\Delta_\phi Z = [\Delta_\phi z_0, \Delta_\phi z_1, \ldots, \Delta_\phi z_{2n-1}]$ is the third output of the routine D-GRAIN$(\phi, 2n)$, then

$$\mathcal{Q}^1_\phi(i) = \begin{cases} 1, & \text{if } \Delta_\phi z_i = 0, \\ 0, & \text{otherwise.} \end{cases} \qquad \mathcal{Q}^2_\phi(i) = \begin{cases} 1, & \text{if } \Delta_\phi z_i = 1, \\ 0, & \text{otherwise.} \end{cases}$$

Grain-128a Grain-128a has a different encryption strategy in which the first 64 key-stream bits and every alternate key-stream bit thereof is used to construct the message authentication code and therefore unavailable to the attacker. To circumvent this problem, in Grain-128a every re-keying is followed by a fault injection at the beginning of round 64 of the PRGA instead of round 0. Hence the vectors Z, Z^ϕ are defined as $Z = [z_{64}, z_{66}, \ldots, z_{318}]$ and $Z^\phi = [z^\phi_{64}, z^\phi_{66}, \ldots, z^\phi_{318}]$. As before, we define $E(\phi) = 1 \oplus Z \oplus Z^\phi$ and $\overline{E}(\phi) = 1 \oplus E(\phi)$ and $\mathcal{Q}^1_\phi, \mathcal{Q}^2_\phi$ are defined as the bitwise AND of all possible $E(\phi), \overline{E}(\phi)$ respectively. Note that if a fault is applied at a random LFSR location ϕ at the 64^{th} PRGA round, then the t^{th} state of Δ_ϕ-Grain will align itself with the $(64+t)^{th}$ state of Grain-128a. This implies that if $\Delta_\phi Z = [\Delta_\phi z_0, \Delta_\phi z_1, \ldots, \Delta_\phi z_{255}]$ is the third output of the routine D-Grain$(\phi, 256)$, then

$$\mathcal{Q}^1_\phi(i) = \begin{cases} 1, & \text{if } \Delta_\phi z_{2i} = 0, \\ 0, & \text{otherwise.} \end{cases} \qquad \mathcal{Q}^2_\phi(i) = \begin{cases} 1, & \text{if } \Delta_\phi z_{2i} = 1, \\ 0, & \text{otherwise.} \end{cases}$$

5.2.3 First and Second Signature Vectors $\mathcal{Q}^1_\phi, \mathcal{Q}^2_\phi$

The task for the fault identification routine is to determine the value of ϕ given the vector E_ϕ. For any element $V \in \{0,1\}^l$, define the set

$$B_V = \{i : 0 \leq i < l,\ V(i) = 1\}$$

i.e. B_v is the support of of V. Now define a relation $\preceq$ in $\{0,1\}^l$ such that for any two elements $V_1, V_2 \in \{0,1\}^l$, we will have $V_1 \preceq V_2$ if $B_{V_1} \subseteq B_{V_2}$. Now we check the elements in B_{E_ϕ} and $B_{\overline{E}_\phi}$. By definition, these are the PRGA rounds i during which $z_i = z^\phi_i$ and $z_i \neq z^\phi_i$ respectively. By the definition of the first and second Signature vector proposed above, we know that for the correct value of ϕ, $B_{\mathcal{Q}^1_\phi} \subseteq B_{E_\phi}, B_{\mathcal{Q}^2_\phi} \subseteq B_{\overline{E}_\phi}$ and hence $\mathcal{Q}^1_\phi \preceq E_\phi, \mathcal{Q}^2_\phi \preceq \overline{E}_\phi$. So our strategy would be to search over all the $2n$ first Signature vectors and formulate the first candidate set

$$\Psi_{0,\phi} = \{\psi : 0 \leq \psi \leq 2n-1,\ \mathcal{Q}^1_\psi \preceq E_\phi\}.$$

If $|\Psi_{0,\phi}|$ is 1, then the single element in $\Psi_{0,\phi}$ will give us the fault location ϕ. If not, we then formulate the second candidate set

$$\Psi_{1,\phi} = \{\psi : \psi \in \Psi_{0,\phi},\ \mathcal{Q}^2_\psi \preceq \overline{E}_\phi\}.$$

If $|\Psi_{1,\phi}|$ is 1, then the single element in $\Psi_{1,\phi}$ will give us the fault location ϕ. If $\Psi_{1,\phi}$ has more than one element, we will be unable to decide conclusively at this stage.

Experimental Results

After comparing the differential vector E^{ϕ} with the first and second signature vectors of all the $2n$ register locations using the strategy outlined above. After the comparison with the signature vectors the routine will either output

1. The LFSR or NFSR location ϕ of the induced fault, OR
2. If $|\Psi_{1,\phi}| > 1$, then it outputs a failure message $\perp$.

We performed computer experiments by simulating random single bit faults for 2^{20} randomly chosen Key-IVs. The probability that this algorithm identifies the correct fault location in the LFSR or the NFSR i.e. $\mathsf{Pr}(|\Psi_{1,\phi}| = 1)$ is around 0.99 for Grain v1, 1.00 for Grain-128 and 0.81 for Grain-128a.

5.2.4 Improving the success probabilities: Third and Fourth signature Vectors

While the probabilities of success of fault location identification are very high (close to 1) for both Grain v1 and Grain-128, it is around 0.81 for Grain-128a. One of the reasons why the success probability relatively low for Grain-128a is because in the event the cipher is used for the dual purpose of authentication and encryption, it does not make each and every key-stream bit directly available to the adversary. As has been explained, the key-stream bits of the first 64 rounds and every alternate round thereafter contribute to the computation of the MAC and is not directly available to the adversary. This limits the information available to the location identification algorithm and hence the slightly low success probability.

This leaves plenty of room for improving the success probabilities towards 1. We already know that given a single bit fault in the internal state of the cipher, the faulty and the faultless key-streams at certain PRGA rounds are guaranteed to be equal and they are also guaranteed to be different at certain other rounds. However there may be situations when the difference of the faulty and faultless key-stream bits at a certain PRGA round i, i.e., $z_i \oplus z_i^{\phi}$ is deterministically equal or unequal to the difference of the faulty and faultless key-stream bits $z_j \oplus z_j^{\phi}$ at some other PRGA round j even though the difference of these bits at either rounds i or j themselves is not guaranteed to be either 0 or 1.

That is to say if $\delta_{t_0} = z_{t_0} \oplus z_{t_0}^{\phi}$, and $\delta_{t_1} = z_{t_1} \oplus z_{t_1}^{\phi}$ for two integers t_0, t_1, then there exist certain values of ϕ, t_0 and t_1, for which

A. $\delta_{t_0} = \delta_{t_1}$ always holds but $\delta_{t_0} = 0,\ \delta_{t_1} = 0$ or $\delta_{t_0} = 1,\ \delta_{t_1} = 1$ does not hold for all values of S.

B. $\delta_{t_0} = 1 \oplus \delta_{t_1}$ always holds but $\delta_{t_0} = 0,\ \delta_{t_1} = 1$ or $\delta_{t_0} = 1,\ \delta_{t_1} = 0$ does not hold for all values of S.

We will first use the tool D-GRAIN(ϕ, r) proposed in Section 5.2.1 that can be used to analyze all the 3 versions of Grain. Briefly recalling, D-GRAIN(ϕ, r) is an algorithm that performs simple truncated differential analysis of the Grain cipher. It takes two inputs: **(a)** the fault location $\phi \in [0, 2n-1]$ in either the LFSR or NFSR, and **(b)** the number of PRGA rounds r for which the analysis is to be performed. The algorithm initializes a differential engine Δ_ϕ-GRAIN, which consists of an n-integer LFSR and NFSR with the same taps as a given version of Grain, but with different update functions. Table 5.1 presents a comparison.

	Grain cipher
LFSR Update	$y_{t+n} = y_t \oplus y_{t+f_1} \oplus y_{t+f_2} \oplus \cdots \oplus y_{t+f_a}$
NFSR Update	$x_{t+n} = y_t \oplus g(x_t, x_{t+g_1}, x_{t+g_2}, \ldots, x_{t+g_b})$
	Δ_ϕ-GRAIN
LFSR Update	$u_{t+n} = u_t + u_{t+f_1} + u_{t+f_2} + \ldots + u_{t+f_a} \bmod 2$
NFSR Update	$v_{t+n} = u_t + 2 \cdot OR(v_t, v_{t+g_1}, \ldots, v_{t+g_b})$

TABLE 5.1: The engine Δ_ϕ-GRAIN

Let us denote the symbols $S_t = X_t || Y_t$ and $S_t^\phi = X_t^\phi || Y_t^\phi$ the corresponding internal states at round t, which differed in the LFSR location ϕ at the beginning of the PRGA. Also let η_t, θ_t (resp. $\eta_t^\phi, \theta_t^\phi$) be the t^{th} round vectors of S_t (resp. S_t^ϕ) that contribute to the output key-stream bit as a linear mask and input to the function h respectively. Then it has been shown in Section 5.2.1, that if the i^{th} element of χ_t (resp. Υ_t) is

(1) 0, then the i^{th} bits of η_t and η_t^ϕ (θ_t and θ_t^ϕ) is equal for all values of S_0,

(2) 1, then the i^{th} bits of η_t and η_t^ϕ (θ_t and θ_t^ϕ) is unequal for all values of S_0,

(3) 2 or 3, then the difference between the i^{th} bits of η_t and η_t^ϕ (θ_t and θ_t^ϕ) is probabilistic.

Similarly, if Δz_t is 0 or 1, it implies that z_t and z_t^ϕ are respectively equal or unequal for all S_0. However if this output is 2 then the difference is probabilistic.

Consider the situation when for some particular value of ϕ the output in the t_0^{th} PRGA round of D-GRAIN(ϕ, r) i.e. $[\chi_{t_0}, \Upsilon_{t_0}, \Delta z_{t_0}]$ be such that

(i) $\Upsilon_{t_0} = \mathbf{0}$ and

(ii) χ_{t_0} has all but one element equal to 0, and this non-zero element is strictly greater than 1, i.e. $v_{t_0+l_w} > 1$ for some $w \leq c$ and all other $v_{t_0+l_k}, u_{t_0+i_k}$ equals 0.

Then following **(1) - (3)**, for all values of S, we must have $\theta_{t_0} = \theta^{\phi}_{t_0}$ and η_{t_0} and $\eta^{\phi}_{t_0}$ have all but their w^{th} element deterministically equal. Let us call the difference of the w^{th} elements of these vectors equal to δ. If $\mathcal{P}(\cdot)$ denotes the GF(2) sum of the elements of a vector, then we have

$$\delta_{t_0} = z_{t_0} \oplus z^{\phi}_{t_0} = \mathcal{P}(\eta_{t_0}) \oplus h(\theta_{t_0}) \oplus \mathcal{P}(\eta^{\phi}_{t_0}) \oplus h(\theta^{\phi}_{t_0}) = \delta \text{ (say)}.$$

Consider the output of D-Grain(ϕ, r) at the PRGA round $t_1 = t_0 - l_e + l_w$ for some $l_e < l_w$. Note that due to the evolution of the LFSR of Δ_ϕ-Grain the difference of the e^{th} element of χ_{t_1} must be equal to the w^{th} element of χ_{t_0}. Now if **(iii)** all the remaining elements of χ_{t_1} and the entire of Υ_{t_1} are all $0'$s, then following the previous argument we have

$$\delta_{t_1} = z_{t_1} \oplus z^{\phi}_{t_1} = \mathcal{P}(\eta_{t_1}) \oplus h(\theta_{t_1}) \oplus \mathcal{P}(\eta^{\phi}_{t_1}) \oplus h(\theta^{\phi}_{t_1}) = \delta.$$

Thus at PRGA rounds t_0, t_1 we have $\delta_{t_0} = \delta_{t_1} = \delta$, even though δ itself is not deterministic. Experimental results have shown that for all the three versions of Grain, taking $r = 2n$, there exist such pairs t_0, t_1 for many values of ϕ.

Similarly consider some other PRGA round $t_1 = t_0 - l_{e'} + l_w$ for some $l_{e'} < l_w$. Then the difference of the e'-th element of χ_{t_1} must be equal to the w^{th} element of χ_{t_0}. Now if **(iv)** $\Upsilon_{t_1} = \mathbf{0}$ and **(v)** there exists some w' such that $\chi_{t_1}[w'] = 1$ and all the remaining elements of of χ_{t_1} is 0, this implies that

(vi) all elements of θ_{t_1} and $\theta^{\phi}_{t_1}$ are deterministically equal,

(vii) the w'-th elements of η_{t_1} and $\eta^{\phi}_{t_1}$ are deterministically unequal,

(viii) the difference δ between the e'-th element of η_{t_1} and $\eta^{\phi}_{t_1}$ is probabilistic and

(ix) all other elements of η_{t_1} and $\eta^{\phi}_{t_1}$ are deterministically equal.

When this occurs,

$$\delta_{t_1} = z_{t_1} \oplus z^{\phi}_{t_1} = \mathcal{P}(\eta_{t_1}) \oplus h(\theta_{t_1}) \oplus \mathcal{P}(\eta^{\phi}_{t_1}) \oplus h(\theta^{\phi}_{t_1}) = 1 \oplus \delta.$$

Thus at PRGA rounds t_0, t_1 we have $\delta_{t_0} = 1 \oplus \delta_{t_1} = \delta$ even though δ is non-deterministic. As above, the existence of ϕ for which there exist PRGA rounds t_0, t_1 that satisfy **(i),(ii)**

and **(iv),(v)** for all the three versions of Grain, can be shown by construction, i.e., by executing the D-GRAIN (ϕ, r) routine for $r = 2n$ and for all $\phi \in [0, 2n-1]$.

Example 5.1. *Let S, $S_{\Delta_{38}}$ be two internal states in Grain v1, that differ only in the LFSR location 38 at the beginning of the PRGA. Then although $z_{79} = z_{79}^{38},\ z_{104} = z_{104}^{38} \oplus 1$ or $z_{79} = z_{79}^{38} \oplus 1,\ z_{104} = z_{104}^{38} \oplus 1$ does not hold for all values of S, $z_{79} \oplus z_{79}^{38} = z_{104} \oplus z_{104}^{38} \oplus 1$ always holds.*

Example 5.2. *Again, let S, S_{Δ_0} be two internal states in Grain v1 that differ only in the LFSR location 0 at the beginning of the PRGA. Then although $z_{41} = z_{41}^0,\ z_{66} = z_{66}^0$ or $z_{41} = z_{41}^0 \oplus 1,\ z_{66} = z_{66}^0 \oplus 1$ does not hold for all values of S, $z_{41} \oplus z_{41}^0 = z_{66} \oplus z_{66}^0$ always holds.*

Now by performing a differential trail analysis using D-GRAIN(ϕ, r) for all the register locations ϕ in the LFSR and the NFSR we will obtain a set of PRGA rounds for most of the register locations at which the difference between the faulty and faultless key-stream bits are thus related. This fact can be further utilized to improve the success probability of the identification algorithm. For example, suppose the given identification algorithm using the first two signature vectors, narrows down the set $\Psi_{1,\phi}$ to $\{0, 25\}$. Now suppose, we observe that $z_{41} \oplus z_{41}^{\phi} = 1 \oplus z_{66} \oplus z_{66}^{\phi}$. We can immediately conclude that $\phi \neq 0$, and hence $\phi = 25$ must be the actual fault location.

We will now formalize the above ideas by first defining the **Third and Fourth signature vectors** $\mathcal{Q}_{\phi}^3, \mathcal{Q}_{\phi}^4 \in \{0,1\}^{2n}$. As before, we index the n LFSR locations as $0, 1, \ldots, n-1$ and the n NFSR locations as $n, n+1, \ldots, 2n-1$. Then for every register location $\phi \in [0, 2n-1]$ define the set of tuples

$$\mathcal{C}_3^{\phi} = \{(i,j) : i \neq j \text{ and } z_i \oplus z_i^{\phi} = z_j \oplus z_j^{\phi} = \delta,\ \forall\ S_0,\ \text{but } 0 < \mathsf{Pr}(\delta = 0) < 1\}$$

$$\text{and } \mathcal{C}_4^{\phi} = \{(i,j) : i \neq j \text{ and } z_i \oplus z_i^{\phi} = z_j \oplus z_j^{\phi} \oplus 1 = \delta,\ \forall\ S_0,\ \text{but } 0 < \mathsf{Pr}(\delta = 0) < 1\}$$

Now we define $\mathcal{Q}_{\phi}^3, \mathcal{Q}_{\phi}^4$ as

$$\mathcal{Q}_{\phi}^3(i) = \mathcal{Q}_{\phi}^3(j) = \begin{cases} \max(i,j), & \text{if } (i,j) \in \mathcal{C}_3^{\phi} \\ 0, & \text{otherwise.} \end{cases}$$

$$\mathcal{Q}_{\phi}^4(i) = \mathcal{Q}_{\phi}^4(j) = \begin{cases} \max(i,j), & \text{if } (i,j) \in \mathcal{C}_4^{\phi} \\ 0, & \text{otherwise.} \end{cases}$$

The max() function has been chosen to ensure that for two distinct pairs $(i_0, j_0) \neq (i_1, j_1) \in C_t^{\phi}$, we have $\mathcal{Q}_{\phi}^t(i_0) \neq \mathcal{Q}_{\phi}^t(i_1)$. Now for ϕ to be the correct fault location for some differential vector E^{ϕ} we must have $E^{\phi}(i) = E^{\phi}(j)$ whenever $\mathcal{Q}_{\phi}^3(i) = \mathcal{Q}_{\phi}^3(j) \neq 0$

and $E^{\phi}(i) = 1 \oplus E^{\phi}(j)$ whenever $\mathcal{Q}^4_{\phi}(i) = \mathcal{Q}^4_{\phi}(j) \neq 0$. Let us denote these conditions by the notations $E^{\phi} \lhd \mathcal{Q}^3_{\phi}$ and $E^{\phi} \lhd \mathcal{Q}^4_{\phi}$. Now, our strategy would be to take the second candidate set $\Psi_{1,\phi}$ and examine all the $2n$ third Signature vectors and formulate the third candidate set defined as

$$\Psi_{2,\phi} = \{\psi : \psi \in \Psi_{1,\phi},\ E_{\phi} \lhd \mathcal{Q}^3_{\psi}\}.$$

If $|\Psi_{2,\phi}|$ is 1, then the single element in $\Psi_{2,\phi}$ will give us the fault location ϕ. If not, we then formulate the fourth candidate set

$$\Psi_{3,\phi} = \{\psi : \psi \in \Psi_{2,\phi},\ E_{\phi} \lhd \mathcal{Q}^4_{\psi}\}.$$

If $|\Psi_{3,\phi}|$ is 1, then the single element in $\Psi_{3,\phi}$ will give us the fault location ϕ. If $\Psi_{3,\phi}$ has more than one element, we will be unable to decide conclusively at this stage and the algorithm returns a failure message.

We now formally define our fault location identification algorithm $\mathsf{FLocl}(E^{\phi})$ in Algorithm 5.2.

Input: The vector E^{ϕ}

Output: The fault location $\phi \in [0, 2n-1]$ or a failure message $\perp$

Compute the vectors $\mathcal{Q}^1_{\phi}, \mathcal{Q}^2_{\phi}, \mathcal{Q}^3_{\phi}, \mathcal{Q}^4_{\phi}$

Formulate the set $\Psi_{0,\phi} = \{\psi : 0 \leq \psi \leq n-1,\ \mathcal{Q}^1_{\psi} \preceq E_{\phi}\}$.

if $|\Psi_{0,\phi}| = 1$ **then**
 | Return the single element in $\Psi_{0,\phi}$
end

Formulate the set $\Psi_{1,\phi} = \{\psi : \psi \in \Psi_{0,\phi},\ \mathcal{Q}^2_{\psi} \preceq \overline{E}_{\phi}\}$.

if $|\Psi_{1,\phi}| = 1$ **then**
 | Return the single element in $\Psi_{1,\phi}$
end

Formulate the set $\Psi_{2,\phi} = \{\psi : \psi \in \Psi_{1,\phi},\ E_{\phi} \lhd \mathcal{Q}^3_{\psi}\}$.

if $|\Psi_{2,\phi}| = 1$ **then**
 | Return the single element in $\Psi_{2,\phi}$
end

Formulate the set $\Psi_{3,\phi} = \{\psi : \psi \in \Psi_{2,\phi},\ E_{\phi} \lhd \mathcal{Q}^4_{\psi}\}$.

if $|\Psi_{3,\phi}| = 1$ **then**
 | Return the single element in $\Psi_{3,\phi}$
end

Return $\perp$.

Algorithm 5.2: The routine $\mathsf{FLocl}(E^{\phi})$

Experimental Results

We again performed computer experiments by simulating random single bit faults for 2^{20} randomly chosen Key-IVs. The probability that the routine $\mathsf{FLocl}(E^{\phi})$ identifies the correct fault location in the LFSR or the NFSR i.e. $\mathsf{Pr}(|\Psi_{3,\phi}| = 1)$ is around 1.00 for Grain v1, 1.00 for Grain-128 and 0.81 for Grain-128a.

5.3 DFA on Grain under relaxed assumptions

Now, once the attacker has deduced the location of an injected fault, he can proceed to use this information to mount an attack on the cipher. In this section we will describe an attack on the Grain family, under the most relaxed of assumptions, i.e., we assume that the attacker is well equipped and has the power to do the following:

- Inject multiple, time-synchronized, single bit-flipping faults in the same albeit random register location.
- Reset the device implementing the cipher and restart cipher operations afresh.

As already explained, there have been two other works presented in [35, 85], that cryptanalyze Grain-128 under the same fault model. However the algebraic structure of Grain-128 is relatively simple as the NFSR update g in Grain-128 is quadratic and the output function h is the sum of only one cubic monomial and a quadratic bent function. This is not the scenario in Grain v1, where the Boolean functions are of more complicated structure in their Algebraic Normal Form. Therefore, in this chapter, we will concentrate on Grain v1 as a case study to explain our novel approach. However, the attack is indeed generic and it will apparent that it works for any version of the Grain family.

The novel idea this fault attack is based on certain specific observations related to the output Boolean function h. For Grain v1, h is a 5-variable function with the differential property that

$$h(s_0, s_1, s_2, s_3, s_4) \oplus h(1 \oplus s_0, 1 \oplus s_1, s_2, s_3, 1 \oplus s_4) = s_2.$$

By using this differential property judicially, one can formulate several linear equations on the LFSR state variables in the beginning of the PRGA, and by solving them we get the complete LFSR state. Then we further note that h can be written as

$$s_4 \cdot u(s_0, s_1, s_2, s_3) \oplus v(s_0, s_1, s_2, s_3),$$

where

$$u(s_0, s_1, s_2, s_3) \oplus u(s_0, 1 \oplus s_1, s_2, 1 \oplus s_3) = 1.$$

This property helps us in determining the NFSR bits.

5.3.1 Determining the LFSR Internal State

Once the fault location ϕ has been identified we can proceed towards determining the LFSR internal state at the beginning of the PRGA. Depending on the value of ϕ we do one of the following.

- If $80 \leq \phi \leq 159$, i.e., the fault is injected in an NFSR location, we disregard the faulty keystream bits and reset the cipher and look to hit an LFSR location.
- If $0 \leq \phi \leq 37$, we disregard the faulty keystream bits, and reset the cipher and look to hit another LFSR location.
- If $38 \leq \phi \leq 41$, we look to apply another fault at the same location ϕ at the beginning of PRGA round 20 and record the faulty keystream bits at certain specific PRGA rounds. We then reset the cipher and look to hit another LFSR location.
- If $42 \leq \phi \leq 79$, we look to apply another fault at the same location ϕ at the beginning of PRGA round 20 and record the faulty keystream bits at certain specific PRGA rounds. We reset the cipher again and apply faults at the location ϕ at the beginning of PRGA rounds $204, 224$ and record the faulty keystream bits at certain other specific PRGA rounds. We then reset the cipher and look to hit another LFSR location
- We continue this process till all LFSR locations 38 to 79 have been hit.

Before describing the attack in detail let us state the following symbolic notations that we shall be using henceforth.

Some notations

1. $S_t = [x_0^t, x_1^t, \ldots, x_{79}^t \quad y_0^t, y_1^t, \ldots, y_{79}^t]$ is used to denote the internal state of the cipher at the beginning of round t of the PRGA. Thus x_i^t (y_i^t) denotes the i^{th} NFSR (LFSR) bit at the start of round t of the PRGA. When $t = 0$, we use $S_0 = [x_0, x_1, \ldots, x_{79} \quad y_0, y_1, \ldots, y_{79}]$ to denote the internal state for convenience.

2. $S_{t,\Delta_\phi}(t_1, t_2)$ is used to denote the internal state of the cipher at the beginning of round t of the PRGA, when a fault has been injected in LFSR location ϕ at the beginning of the t_1^{th} and the t_2^{th} PRGA round.

3. $z_i^{\phi}(t_1, t_2)$ denotes the keystream bit produced in the i^{th} PRGA round, after faults have been injected in LFSR location ϕ at the beginning of the t_1^{th} and the t_2^{th} PRGA round. z_i is the fault-free i^{th} keystream bit.

Beginning the attack

We start by making the following observation about the output Boolean function h in Grain v1: $h(s_0, s_1, s_2, s_3, s_4) \oplus h(1 \oplus s_0, 1 \oplus s_1, s_2, s_3, 1 \oplus s_4) = s_2$. Note that s_0, s_1, s_2, s_3 correspond to LFSR locations $3, 25, 46, 64$ respectively and s_4 corresponds to the NFSR location 63. This implies that if two internal states S and S_Δ be such that they differ in LFSR locations $3, 25$ and NFSR location 63 and in no other location that contributes inputs to the output keystream bit, then the difference of the keystream bit produced by them will be equal to the value in LFSR location 46. Getting differentials at exactly these 3 locations may be difficult by injecting a single fault, but may be achieved if we faulted the same LFSR location twice, as will be explained by the following lemma.

Lemma 5.1. *If a fault is injected in LFSR location* $38+r$ *(*$0 \leq r \leq 41$*), at the beginning of the PRGA rounds* λ *and* $\lambda + 20$ *(*$\lambda = 0, 1, \ldots$*), then in round number* $55 + \lambda + r$ *of the PRGA, the faulty internal state* $S_{55+\lambda+r,\Delta_{38+r}}(\lambda, \lambda + 20)$ *and the fault-free internal state* $S_{55+\lambda+r}$ *will differ in LFSR locations* $3, 25$ *and NFSR location* 63 *and in none of the other* 9 *tap locations that contributes to the output keystream bit.*

Proof. We will prove the Lemma for $\lambda = 0$. That the proof will hold for any $\lambda > 0$ is obvious from the statement. The proof requires the analysis of the differential trail of the successive PRGA rounds and thus one must execute the D-Grain$(38 + r, 160)$ routine (see Algorithm 5.1). However, this is slightly tricky as the D-Grain routine assumes that the fault is always injected at the start of PRGA and does not make provision for multiple fault injection at the same location. In order to circumvent this, we will modify the D-Grain routine as follows: we replace the comment in Line 1 of D-Grain, i.e., Algorithm 5.1 by the statement

$$\text{If } t = 20 \text{ then } u_\phi = 1$$

This will now take care of the double fault situation. After executing this routine for all $\phi = 38 + r$ $(0 \leq r \leq 41)$, one will observe that Υ_{55+r} will have 1 in the positions

corresponding to LFSR locations $3, 25$ and NFSR location 63. All other elements of Υ_{55+r} and χ_{55+r} will be zeroes, thus proving the Lemma. □

The above lemma implies that if $\lambda = 0$, i.e., if faults are injected at the beginning of the PRGA and round 20 at location $38 + r,\ 0 \leq r \leq 41$ of the LFSR, then in the PRGA round $55 + r$ we will have

$$z_{55+r} \oplus z_{55+r}^{38+r}(0, 20) = y_{46}^{55+r} \quad \forall r \in [0, 41].$$

Now since the NFSR does not influence the LFSR during the PRGA, y_{46}^{55+r} is a linear function of the initial LFSR bits $y_0, y_1, \ldots, y_{79}$ for all $0 \leq r \leq 41$. For example, by analyzing the LFSR we have

$$y_{46}^{55} = y_3 \oplus y_{16} \oplus y_{21} \oplus y_{26} \oplus y_{34} \oplus y_{41} \oplus y_{44} \oplus y_{54} \oplus y_{59} \oplus y_{65} \oplus y_{72}.$$

So in this process, we obtain 42 linear equations in the original LFSR bits $y_0, y_1, \ldots, y_{79}$. We need another 38 equations such that the resulting 80 equations are linearly independent. We have attempted to find the remaining 38 equations by resetting the cipher and then introducing faults later in the PRGA. If we let $\lambda = 204$, i.e., if double faults were introduced in LFSR locations $42 + r$ with $0 \leq r \leq 37$ at the beginning of the PRGA rounds 204 and 224, then by the previous analysis it may be deduced that

$$z_{263+r} \oplus z_{263+r}^{42+r}(204, 224) = y_{46}^{263+r} \quad \forall r \in [0, 37].$$

This provides us with another 38 equations. We have observed that these equations are linearly independent. Writing these equations in matrix notation, we have $LY = W$. The rows of the matrix L is defined by the linear functions $y_{46}^{55}, y_{46}^{56}, \ldots, y_{46}^{96}, y_{46}^{263}, \ldots, y_{46}^{300}$. Further, $Y = [y_0\ y_1\ \ldots\ y_{79}]^T$ and W is the column vector defined as follows

$$W(r) = z_{55+r} \oplus z_{55+r}^{38+r}(0, 20) \quad 0 \leq r \leq 41,$$
$$W(42 + r) = z_{263+r} \oplus z_{263+r}^{42+r}(204, 224) \quad 0 \leq r \leq 37.$$

Since the matrix L and its inverse can be pre-computed beforehand, the vector $Y = L^{-1}W$ can be calculated immediately after applying the faults and calculating W.

Note that for the second round of fault injections the choice of fault locations $42 \leq \phi \leq 79$ and PRGA rounds $204, 224$ is by no means unique. By searching over various values of λ, one may be able to obtain a set of linearly independent equations for other choices of fault locations and PRGA rounds.

Remark 5.2. The method works in a similar manner for Grain-128 and Grain-128a. For example, the output function in Grain-128 is $h(s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8) = s_0s_1 \oplus s_2s_3 \oplus s_4s_5 \oplus s_6s_7 \oplus s_0s_4s_8$, where s_0 and s_4 corresponds to NFSR variables. One can check that for any $\alpha \in \{001000000, 000100000, 000000100, 000000010\}$, $h(\mathbf{x}) \oplus h(\mathbf{x} \oplus \alpha)$ is a linear function of LFSR variables only.

5.3.2 Determining the NFSR Internal State

Once the LFSR internal state of the initial PRGA round is known, one can then proceed to determine the NFSR internal state. In [31] it was shown, that this could have been done efficiently for the initial version of the cipher i.e. Grain v0. After the attack in [31] was reported, the designers made the necessary changes to Grain v1, Grain-128 and Grain-128a so that for these new ciphers, determining the NFSR state form the knowledge of the LFSR state was no longer straightforward. In order to determine the NFSR bits, we look into the decomposition of the Boolean function h in more detail. The attack we will describe in this section can be mounted due to the following observations on the Grain output function h.

A. $h(\cdot)$ can be written in the form $s_j \cdot u(\cdot) \oplus v(\cdot)$ where s_j corresponds to a variable which takes input from an NFSR tap location;

B. There exists a differential β such that $u(\mathbf{s}) \oplus u(\mathbf{s} \oplus \beta) = 1$;

C. $v(\mathbf{s}) \oplus v(\mathbf{s} \oplus \beta) =$ a function of variables that takes input from LFSR locations only.

For Grain v1, $h(s_0, s_1, s_2, s_3, s_4) = s_4 \cdot u(s_0, s_1, s_2, s_3) \oplus v(s_0, s_1, s_2, s_3)$, where

$$u(s_0, s_1, s_2, s_3) = 1 \oplus s_3 \oplus s_0s_2 \oplus s_1s_2 \oplus s_2s_3,$$

$$v(s_0, s_1, s_2, s_3) = s_1 \oplus s_0s_3 \oplus s_2s_3 \oplus s_0s_1s_2 \oplus s_0s_2s_3$$

Thus we note that (i) u, v are functions on the LFSR bits only, (ii) $u(s_0, s_1, s_2, s_3) \oplus u(s_0, 1 \oplus s_1, s_2, 1 \oplus s_3) = 1$ and (iii) $v(s_0, s_1, s_2, s_3) \oplus v(s_0, 1 \oplus s_1, s_2, 1 \oplus s_3) = 1 \oplus s_0 \oplus s_2$. Hence h satisfies all the properties listed above.

The fault-free keystream bit at the t^{th} round can now be rewritten as $z_t = \bigoplus_{a \in A} x_a^t \oplus x_{63}^t \cdot u(y_3^t, y_{25}^t, y_{46}^t, y_{64}^t) \oplus v(y_3^t, y_{25}^t, y_{46}^t, y_{64}^t)$. Consider two internal states S_t and $S_{t,\Delta}$ which differ in the LFSR locations 25 and 64 and in no other location, that provides input to h. If z_t and $z_{t,\Delta}$ are the keystream bits produced by S_t and $S_{t,\Delta}$ in that round, then using the previous observation we can see that

$$z_t \oplus z_{t,\Delta} = x_{63}^t \oplus v(y_3^t, y_{25}^t, y_{46}^t, y_{64}^t) \oplus v(y_3^t, 1 \oplus y_{25}^t, y_{46}^t, 1 \oplus y_{64}^t).$$

Let $c_t = \left[v(y_3^t, y_{25}^t, y_{46}^t, y_{64}^t) \oplus v(y_3^t, 1 \oplus y_{25}^t, y_{46}^t, 1 \oplus y_{64}^t)\right]$. Since the LFSR internal state is already available, c_t can be computed immediately, and hence the difference of the two keystream bits plus the value of c_t gives us the value at the NFSR location 63 at round t of the PRGA. In the next Lemma, we shall investigate when this differential pattern in the internal state is obtained by employing the same fault injection strategy in the previous subsection.

Lemma 5.3. *Let $S_0, S_1, S_2, \ldots$ be the successive internal states of the PRGA for Grain v1. Then the faulty state $S_{t,\Delta_\phi}(0, 20)$ will differ from S_t at LFSR locations $25, 64$ and none of the other 10 tap locations that feed the output function for the following values of ϕ, t: (i) $\phi = 51 + r$, $t = 91 + r$ for $0 \leq r \leq 28$, (ii) $\phi = 62 + r$, $t = 55 + r$ for $0 \leq r \leq 17$, (iii) $\phi = 62 + r$, $t = 75 + r$ for $0 \leq r \leq 15$.*

Proof. The proof follows from an analysis of the differential trails of Grain v1 PRGA, and is similar to the proof for Lemma 5.1. □

The Lemma essentially implies that if faults are injected at the beginning of the PRGA and round 20 at location $51 + r$ of the LFSR ($0 \leq r \leq 28$), then in the PRGA round $91 + r$ we will have

$$z_{91+r} \oplus z_{91+r}^{51+r}(0, 20) \oplus c_{91+r} = x_{63}^{91+r} \quad \forall r \in [0, 28].$$

Also, the following equations hold:

$$z_{55+r} \oplus z_{55+r}^{62+r}(0, 20) \oplus c_{55+r} = x_{63}^{55+r} \quad \forall r \in [0, 17],$$

$$z_{75+r} \oplus z_{75+r}^{62+r}(0, 20) \oplus c_{75+r} = x_{63}^{75+r} \quad \forall r \in [0, 15].$$

Since the LHS of all the above equations are known, we can therefore calculate the value of the NFSR location 63 for all PRGA rounds 55, 56, ..., 72, 75, 76, ..., 119. Because of the shifting property of the NFSR, the equations $x_i^j = x_{i-1}^{j+1} \quad \forall i \in [1, 79]$ hold. Therefore knowing $x_{63}^{55}, x_{63}^{56}, \ldots, x_{63}^{72}, x_{63}^{75}, x_{63}^{76}, \ldots, x_{63}^{119}$ is equivalent to knowing $x_{15}^{103}, x_{16}^{103}, \ldots, x_{32}^{103}$, $x_{35}^{103}, x_{36}^{103}, \ldots, x_{79}^{103}$, i.e., we now know 63 out of the 80 NFSR state bits of S_{103}.

Finding the Remaining Bits

Any bits of the NFSR internal state not found out in the previous subsection could be obtained by performing an exhaustive search over them. However, if h is such that both u, v are functions on the LFSR bits only then the attack can be further simplified. Since the function h in Grain v1 satisfies this property, we proceed to determine the remaining

17 NFSR bits of S_{103}. These may be found by a combination of solving equations and guesswork. Since the 80 LFSR bits of S_0 have already been found in the previous section, one can efficiently calculate the 80 LFSR bits of S_{103} by running the Grain v1 PRGA routine. This is because the LFSR evolves independently during the PRGA. Then, by observing the fault-free output keystream bits we can write the following equations:

$$z_{102+\gamma} = x_{0+\gamma}^{103} \oplus x_{1+\gamma}^{103} \oplus x_{3+\gamma}^{103} \oplus x_{9+\gamma}^{103} \oplus x_{30+\gamma}^{103} \oplus x_{42+\gamma}^{103} \oplus x_{55+\gamma}^{103} \oplus \mathbf{u}_{102+\gamma} x_{62+\gamma}^{103} \oplus \mathbf{v}_{102+\gamma},$$

for $\gamma = 0, 1, \ldots, 14$, where $\mathbf{u}_i = u(y_3^i, y_{25}^i, y_{46}^i, y_{64}^i)$ and $\mathbf{v}_i = v(y_3^i, y_{25}^i, y_{46}^i, y_{64}^i)$. Since the LFSR initial state is known, $\mathbf{u}_i, \mathbf{v}_i$ are available. Consider the set of 15 equations given above. In the last equation it can be seen that x_{14}^{103} is the only unknown and hence its value may be easily calculated. Once x_{14}^{103} is known, x_{13}^{103} becomes the only unknown in the 14^{th} equation and its value too may be immediately calculated. Backtracking in this manner one can calculate upto x_5^{103} from the 6^{th} equation. At this point we have calculated the value of 73 NFSR bits of S_{103}. The 5^{th} equation is

$$z_{106} = x_4^{103} \oplus x_5^{103} \oplus x_7^{103} \oplus x_{13}^{103} \oplus x_{34}^{103} \oplus x_{46}^{103} \oplus x_{59}^{103} \oplus \mathbf{u}_{106} x_{66}^{103} \oplus \mathbf{v}_{106}.$$

This equation has two unknowns x_4^{103} and x_{34}^{103} and so the value of either unknown can not be calculated conclusively. Similarly the 4^{th} equation has two unknowns x_3^{103} and x_{33}^{103}. If we try out all the possibilities of $x_{34}^{103}, x_{33}^{103}$ then the value of the remaining 5 unknowns may be calculated uniquely. So we do an exhaustive search over the 2 bits (4 possible candidates) for S_{103}. The correct S_{103} may be found out by observing the keystream bits $z_{103}, z_{104}, \ldots$, as required. We eliminate any candidate S_{103} vector that does not produce the required keystream bit sequence. This routine thus gives us the entire S_{103} vector. Note that in order to recover the NFSR state one does not have to inject any additional faults other than those already injected to determine the LFSR state.

Remark 5.4. If the function h in Grain v1 were such that it could not be decomposed into u and v as above, then the attack would not have been as straightforward. The attack here is efficient as u and v are of certain nice structures and their inputs are from LFSR bits only. The LFSR bits are already known after the recovery of the LFSR bits and that helps in recovering the NFSR state easily. It can be checked that the output function of Grain-128 and Grain-128a also follows properties **(A), (B), (C)** given at the beginning of this section and thus renders them vulnerable to this attack.

5.3.3 Finding the Secret Key and Complexity of the Attack

It has already been explained in Section 4.3 that the KSA and PRGA routines in the Grain family are invertible. Once we have all the bits of S_{103}, by running the inverse PRGA routine described in Algorithm 4.2 103 times, we obtain the initial PRGA state S_0. Thereafter, by running the inverse KSA described in Algorithm 4.1 routine one can recover the secret key.

The attack complexity directly depends on the number of fault experiments to be performed such that all of locations in $[38, 79]$ of the LFSR are covered. To have this, the expected number of fault experiments is $160 \cdot \sum_{i=1}^{42} \frac{1}{i} \approx 688$.

Further depending on the LFSR location hit, during the attack phase, one needs to inject 2 or 4 extra faults for determining the internal state. Therefore, the expected number of faults that our attack needs is $688 + 4 \times 38 + 2 \times 4 \approx 2^{9.73}$.

To determine the internal state, we have to perform one matrix multiplication, and solve a set of 78 linear equations and then exhaustively search over 2 variables. After that, 103 invocations of the PRGA^{-1} routine and a single invocation of the KSA^{-1} routine are needed to determine the Secret Key.

Thus the dominant time/memory consuming process in our attack is the multiplication of $L^{-1}W$ which requires around 80×80 bits to store L^{-1} and $80^2 \approx O(2^{12.6})$ bit operations to calculate the product.

5.4 DFA on Grain under stricter assumptions

So far we had analyzed the situation when an all powerful adversary had the ability to inject multiple faults at a random but unknown register location. We will tighten the screws slightly in this Section and take away this power from the attacker. This makes for a far more realistic attack scenario as an adversary may not always possess the precision to inject multiple faults in the same register location. So in the attack that we are about to describe we assume that the attacker has the following liberties:

- He can inject a single, time-synchronized, single bit-flipping fault at some random register location.
- He can reset the device implementing the cipher and restart cipher operations afresh.

It can be argued that the attack described in the previous section was a first order differential attack on the Grain family, since, a linear first order derivative of the output function, i.e., $h(\mathbf{x}) \oplus h(\mathbf{x} \oplus \alpha)$ was used to determine linear equations to solve for the LFSR state. So far, we have ignored higher order differential properties of the function h. In this section we look into higher order differential properties of the output function h of the three ciphers of the Grain family. Since these output functions are of algebraic degree three there exist numerous higher order derivatives of these functions which are also linear. In this Section we shall utilize these properties of the output function to formulate linear equations which when solved together will reveal the internal state of the cipher.

Before we go into the mathematical details of the attack, let us define a few terms that we shall be using to describe the attack.

Definition 5.5. Consider a q-variable Boolean function F and any vector $\alpha \in \{0,1\}^q$. We refer to the function $F(\mathbf{x} \oplus \alpha)$ as a translation of F. The set of all possible translations of a given function F is denoted by the term 'Translation Set' and by the symbol $\mathcal{A}_F$. Since a q-variable function can have at most 2^q translations, the cardinality of $\mathcal{A}_F$ is atmost 2^q.

Definition 5.6. Consider a q-variable Boolean function F and its translation set $\mathcal{A}_F$. Any GF(2) linear combination $\hat{F}$ of the functions in $\mathcal{A}_F$, i.e.,

$$\hat{F}(\mathbf{x}) = c_1 F(\mathbf{x} \oplus \alpha_1) \oplus c_2 F(\mathbf{x} \oplus \alpha_2) \oplus \cdots \oplus c_i F(\mathbf{x} \oplus \alpha_i),$$

where $c_1, c_2, \ldots, c_i \in \{0,1\}$ is said to be a derivative of F. If $\hat{F}$ happens to be an affine Boolean function and $c_1 = c_2 = \cdots = c_i = 1$ then the set of vectors $\pi = [\alpha_1, \alpha_2, \ldots, \alpha_i]$ is said to be an affine differential tuple of F. If none of the vectors in π is $\mathbf{0}$ then π is said to be a weight i affine differential tuple of F otherwise π is said to be a weight $(i-1)$ affine differential tuple.

5.4.1 Beginning the attack

As in the previous Section, let us first describe some notations that we will henceforth use. Note that some notations are in common with the notations described in Section 5.3.1.

1. $S_t = [x_0^t, x_1^t, \ldots, x_{n-1}^t \quad y_0^t, y_1^t, \ldots, y_{n-1}^t]$ is used to denote the internal state of the cipher at the beginning of round t of the PRGA. Thus x_i^t (y_i^t) denotes the i^{th} NFSR (LFSR) bit at the start of round t of the PRGA. When $t = 0$, we use $S_0 = [x_0, x_1, \ldots, x_{n-1} \;\; y_0, y_1, \ldots, y_{n-1}]$ to denote the internal state for convenience.

2. S_{t,Δ_ϕ} is used to denote the internal state of the cipher at the beginning of round t of the PRGA, when a fault has been injected in LFSR location ϕ at the beginning of the PRGA round.

3. z_i^ϕ denotes the key-stream bit produced in the i^{th} PRGA round, after faults have been injected in LFSR location ϕ at the beginning of the PRGA round. z_i is the fault-free i^{th} key-stream bit.

4. $\eta_t = [x_{l_1}^t, x_{l_2}^t \ldots, x_{l_c}^t, y_{i_1}^t, y_{i_2}^t \ldots, y_{i_d}^t]$ is the set of elements in S_t which contribute to the output key-stream bit function linearly and correspondingly the vector $\theta_t = [y_{h_1}^t, y_{h_2}^t, \ldots, y_{h_e}^t, x_{j_1}^t, x_{j_2}^t, \ldots, x_{j_w}^t]$ is the subset of S_t which forms the input to the combining function h.

5. If $\mathbf{v}$ is an integer vector all elements of which are either 0 or 1, then we express $\mathbf{v}$ as a vector over GF(2) and denote it by the symbol $\widetilde{\mathbf{v}}$.

6. If $\mathbf{w}$ is a vector over GF(2) then $\mathcal{P}(\mathbf{w})$ denotes the GF(2) sum of the elements of $\mathbf{w}$.

Determining the LFSR.
During PRGA, the LFSR evolves linearly and independent of the NFSR. Hence, y_i^t for any $i \in [0, n-1]$ and $t \geq 0$ is a linear function of $y_0, y_1, \ldots, y_{n-1}$. Let S_0 and S_{0,Δ_ϕ} be two initial states of the Grain PRGA that differ in only the register location $\phi \in [0, 2n-1]$. Let $[\chi_{0,\phi}, \chi_{1,\phi}, \ldots, \chi_{2n-1,\phi}], [\Upsilon_{0,\phi}, \Upsilon_{1,\phi}, \ldots, \Upsilon_{2n-1,\phi}]$, $\Delta_\phi Z$ be the outputs of D-Grain$(\phi, 2n)$.

Let $[\mathbf{0}, \alpha_1]$ be a weight 1 affine differential tuple of h, such that $h(\mathbf{x}) \oplus h(\mathbf{x} \oplus \alpha_1) = h_{01}(\mathbf{x})$ is a function of variables that takes input from LFSR locations only. If, for some round t of the PRGA, we have $\chi_{t,\phi} \sqsubseteq 1$, $\Upsilon_{t,\phi} \sqsubseteq 1$ and $\widetilde{\Upsilon}_{t,\phi} = \alpha_1$, then we can conclude that the t^{th} round fault-free and faulty internal states S_t and S_{t,Δ_ϕ} differ deterministically in the bit locations that contribute to producing the output key-stream bit at round t. In such a scenario, the GF(2) sum of the fault-free and faulty key-stream bit at round t is given by

$$\begin{aligned} z_t \oplus z_t^\phi &= \mathcal{P}(\eta_t) \oplus h(\theta_t) \oplus \mathcal{P}(\eta_t \oplus \widetilde{\chi}_{t,\phi}) \oplus h(\theta_t \oplus \widetilde{\Upsilon}_{t,\phi}) \\ &= \mathcal{P}(\widetilde{\chi}_{t,\phi}) \oplus h(\theta_t) \oplus h(\theta_t \oplus \alpha_1) = \mathcal{P}(\widetilde{\chi}_{t,\phi}) \oplus h_{01}(\theta_t). \end{aligned}$$

Note that in the above equation $\mathcal{P}(\widetilde{\chi}_{t,\phi}) \oplus h_{01}(\theta_t)$ is an affine Boolean function in the LFSR state bits of $S_t = [y_0^t, y_1^t, \ldots, y_{n-1}^t]$ and hence $[y_0, y_1, \ldots, y_{n-1}]$. Since $z_t \oplus z_t^\phi$ is already known to us, this gives us one linear equation in $[y_0, y_1, \ldots, y_{n-1}]$. The trick is to get n such linear equations which are linearly independent by searching over all possible values of ϕ and affine differential tuples of h. Of course h may not have an

affine differential tuple $[\mathbf{0}, \alpha_1]$ of weight 1 or even if it does $\widetilde{\Upsilon}_{t,\phi} = \alpha_1$ and $\chi_{t,\phi} \sqsubseteq 1$ may not hold for any t or ϕ. In such situations, one can look at other higher weight affine differential tuples.

Exploring higher weight affine differential tuples. Consider λ fault locations $\phi_i \in [0, 2n-1]$. Let $[\chi_{0,\phi_i}, \ldots, \chi_{2n-1,\phi_i}]$, $[\Upsilon_{0,\phi_i}, \ldots, \Upsilon_{2n-1,\phi_i}]$, $\Delta_{\phi_i} Z$ be the λ outputs of D-GRAIN$(\phi_i, 2n)$ for $i \in [1, \lambda]$.

Case 1: λ is odd

Let $[\mathbf{0}, \alpha_1, \alpha_2, \ldots, \ldots, \alpha_\lambda]$ be a weight λ (where λ is an odd number) affine differential tuple of h, such that $h(\mathbf{x}) \oplus \bigoplus_{i=1}^{\lambda} h(\mathbf{x} \oplus \alpha_i) = H_1(\mathbf{x})$ is a function of variables that takes input from LFSR locations only. If for some round t of the PRGA, $\chi_{t,\phi_i} \sqsubseteq 1$, $\Upsilon_{t,\phi_i} \sqsubseteq 1$ and $\widetilde{\Upsilon}_{t,\phi_i} = \alpha_i$ for all $i \in [1, \lambda]$, then by the arguments outlined in the previous subsection, we conclude

$$z_t \oplus \bigoplus_{i=1}^{\lambda} z_t^{\phi_i} = \mathcal{P}(\eta_t) \oplus h(\theta_t) \oplus \bigoplus_{i=1}^{\lambda} \Big(\mathcal{P}(\eta_t \oplus \widetilde{\chi}_{t,\phi_i}) \oplus h(\theta_t \oplus \widetilde{\Upsilon}_{t,\phi_i}) \Big)$$
$$= \bigoplus_{i=1}^{\lambda} \mathcal{P}(\widetilde{\chi}_{t,\phi_i}) \oplus H_1(\theta_t).$$

If λ is odd then we can not exploit differential tuples of the form $[\alpha_1, \alpha_2, \ldots, \ldots, \alpha_\lambda]$ where all $\alpha_i \neq \mathbf{0}$ as an odd number of terms do not cancel out in GF(2).

Case 2: λ is even

Instead, if $[\alpha_1, \alpha_2, \ldots, \ldots, \alpha_\lambda]$ is a weight λ (λ is an even number) affine differential tuple of h, such that $\bigoplus_{i=1}^{\lambda} h(\mathbf{x} \oplus \alpha_i) = H_2(\mathbf{x})$ is a function of variables, that takes inputs from LFSR locations only, then by the previous arguments we have

$$\bigoplus_{i=1}^{\lambda} z_t^{\phi_i} = \bigoplus_{i=1}^{\lambda} \Big(\mathcal{P}(\eta_t \oplus \widetilde{\chi}_{t,\phi_i}) \oplus h(\theta_t \oplus \widetilde{\Upsilon}_{t,\phi_i}) \Big) = \bigoplus_{i=1}^{\lambda} \mathcal{P}(\widetilde{\chi}_{t,\phi_i}) \oplus H_2(\theta_t).$$

Note that each of the above cases gives us one linear equation in $[y_0, y_1, \ldots, y_{n-1}]$. We formally state the routine $\mathrm{FLE}_L(\lambda)$ in Algorithm 5.3 that attempts to find such linear equations by investigating weight λ affine differential tuples.

Solving the system

Ideally we need n linearly independent equations in $[y_0, y_1, \ldots, y_{n-1}]$ to solve the LFSR. If a call to $\mathrm{FLE}_L(1)$ does not give us the requisite number of equations then we must call $\mathrm{FLE}_L(2)$ and if required $\mathrm{FLE}_L(3)$ to obtain the required number of equations. Note that the number of iterations in the outer most **'for'** loop is of $\mathrm{FLE}_L(\lambda)$ is $\binom{n}{\lambda} \approx O(n^\lambda)$, so beyond a certain value of λ, it may not be practically feasible to call $\mathrm{FLE}_L(\lambda)$. Assuming

Input: λ: An integer > 0;
Output: Set of Rounds t, locations $[\phi_1, \phi_2, \ldots, \phi_\lambda]$, Affine expression in $[y_0, y_1, \ldots, y_{n-1}]$; Tuples $[\alpha_1, \ldots, \alpha_\lambda]$

---

for $\phi_1 = 0$ *to* $n-1$, $\phi_2 = 0$ *to* $n-1$, $\ldots$, $\phi_\lambda = 0$ *to* $n-1$ **do**
  if *All* ϕ_j*'s are pairwise unequal* **then**
    for $i = 1$ *to* λ **do**
      $([\chi_{0,\phi_i}, \ldots, \chi_{2n-1,\phi_i}],\ [\Upsilon_{0,\phi_i}, \ldots, \Upsilon_{2n-1,\phi_i}],\ \Delta_{\phi_i} Z) =$ D-Grain$(\phi_i, 2n)$
    end
    for $t = 0$ *to* $2n-1$ **do**
      if $\chi_{t,\phi_i} \sqsubseteq 1$ *AND* $\Upsilon_{t,\phi_i} \sqsubseteq 1$, $\forall i \in [1, \lambda]$ **then**
        if λ *is odd* **then**
          $H_1(\mathbf{x}) = h(\mathbf{x}) \oplus_{i=0}^{\lambda} h(\mathbf{x} \oplus \widetilde{\Upsilon}_{t,\phi_i})$;
          if H_1 *is a function only on LFSR bits* **then** **1**
            Output Round t, Locations $[\phi_1, \phi_2, \ldots, \phi_\lambda]$, Expression $\oplus_{i=1}^{\lambda} \mathcal{P}(\widetilde{\chi}_{t,\phi_i}) \oplus H_1(\theta_t)$;
            Output Tuple $[\mathbf{0}, \widetilde{\Upsilon}_{t,\phi_1}, \ldots, \widetilde{\Upsilon}_{t,\phi_\lambda}]$
          end
        end
        else
          $H_2(\mathbf{x}) = \oplus_{i=0}^{\lambda} h(\mathbf{x} \oplus \widetilde{\Upsilon}_{t,\phi_i})$;
          if H_2 *is a function only on LFSR bits* **then** **2**
            Output Round t, Locations $[\phi_1, \phi_2, \ldots, \phi_\lambda]$, Expression $\oplus_{i=1}^{\lambda} \mathcal{P}(\widetilde{\chi}_{t,\phi_i}) \oplus H_2(\theta_t)$;
            Output Tuple $[\widetilde{\Upsilon}_{t,\phi_1}, \ldots, \widetilde{\Upsilon}_{t,\phi_\lambda}]$
          end
        end
      end
    end
  end
end

Algorithm 5.3: FLE$_L(\lambda)$

that we have n outputs from the successive FLE$_L(\lambda)$ routines of the form

$$t_i,\ \ [\phi_{1,i}, \phi_{2,i}, \ldots, \phi_{\lambda_i,i}],\ \gamma_i \oplus \bigoplus_{j=0}^{n-1} c_{i,j} y_j,\ [\alpha_{1,i}, \alpha_{2,i}, \ldots, \alpha_{\lambda_i,i}],$$

$\forall i \in [0, n-1]$, if λ_i is even. Else the last output will be of the form $[\mathbf{0}, \alpha_{1,i}, \alpha_{2,i}, \ldots, \alpha_{\lambda_i,i}]$. Then we can write the equations so obtained in matrix form $LY = W$, where L is the $n \times n$ coefficient matrix $\{c_{i,j}\}$ over GF(2), Y is the column vector $[y_0, y_1, \ldots, y_{n-1}]^t$ and W is a column vector whose i^{th} element $W(i)$ is defined as follows:

$$W(i) \quad = \quad \begin{cases} \gamma_i \oplus z_{t_i} \oplus \bigoplus_{j=1}^{\lambda_i} z_{t_i}^{\phi_{j,i}}, & \text{if } \lambda_i \text{ is odd,} \\ \\ \gamma_i \oplus \bigoplus_{j=1}^{\lambda_i} z_{t_i}^{\phi_{j,i}}, & \text{if } \lambda_i \text{ is even.} \end{cases}$$

If the equations are linearly independent then L is invertible. Thus, the solution Y of the above system are obtained by computing $L^{-1}W$. Both L and its inverse may be precomputed and hence the solution can be obtained immediately after recording the faulty bits.

Determining the NFSR.

Once the LFSR state has been determined, we proceed to finding the NFSR state. Since the NFSR updates itself non-linearly, the method used to determine the NFSR initial state will be slightly different from the LFSR. If λ is odd, let $[\mathbf{0}, \alpha_1, \alpha_2, \ldots, \alpha_\lambda]$ be a weight λ (where λ is an odd number) tuple of h (not necessarily affine differential), such that $h(\mathbf{x}) \oplus \bigoplus_{i=1}^{\lambda} h(\mathbf{x} \oplus \alpha_i) = H_1(\mathbf{x}) = x' \oplus H_{11}(\mathbf{x})$ where x' is a variable that takes input from an NFSR location and $H_{11}(\mathbf{x})$ is a function only on the LFSR variables. If for some round t of the PRGA $\chi_{t,\phi_i} \sqsubseteq 1$ and $\Upsilon_{t,\phi_i} \sqsubseteq 1$ and $\widetilde{\Upsilon}_{t,\phi_i} = \alpha_i$ for all $i \in [1, \lambda]$, then by the arguments outlined in the previous subsection we conclude

$$\begin{aligned} z_t \oplus \bigoplus_{i=1}^{\lambda} z_t^{\phi_i} &= \mathcal{P}(\eta_t) \oplus h(\theta_t) \oplus \bigoplus_{i=1}^{\lambda} \left(\mathcal{P}(\eta_t \oplus \widetilde{\chi}_{t,\phi_i}) \oplus h(\theta_t \oplus \widetilde{\Upsilon}_{t,\phi_i}) \right) \\ &= \bigoplus_{i=1}^{\lambda} \mathcal{P}(\widetilde{\chi}_{t,\phi_i}) \oplus H_1(\theta_t) = \bigoplus_{i=1}^{\lambda} \mathcal{P}(\widetilde{\chi}_{t,\phi_i}) \oplus H_{11}(\theta_t) \oplus x_{j_r}^t, \end{aligned}$$

for some $r \in [1, w]$. Since, the LFSR is already known, $H_{11}(\theta_t)$ can be calculated and that leaves $x_{j_r}^t$ as the only unknown in the equation, whose value is also calculated immediately after recording the faulty bits and solving the LFSR.

The λ even case can be dealt with similarly. We can describe another routine $\text{FLE}_N(\lambda)$ which will help in determining the NFSR state. This routine is similar to the $\text{FLE}_L(\lambda)$ routine described in Algorithm 5.3. The only differences are that line 1 will change to

if $H_1(\mathbf{x}) = x' \oplus H_{11}(\mathbf{x})$ where x' is an NFSR term and $H_{11}(\mathbf{x})$ depends on LFSR variables only.

Line 2 of Algorithm 5.3 will also change accordingly. With the help of $\text{FLE}_N(\lambda)$ routine, we can obtain specific NFSR state bits at various rounds of operation of the PRGA.

Due to the shifting property of shift registers, the following equation holds $x_i^t = x_{i-1}^{t+1}$. For example, calculating x_{46}^{30} and x_{50}^{32} is the same as determining the two NFSR state bits of the internal state S_{30}: x_{46}^{30} and x_{52}^{30}.

Hence by using the $\text{FLE}_N(\lambda)$ for successive values of λ, one can obtain all the n NFSR state bits of S_t for some $t \geq 0$. Since the LFSR initial state of S_0 is already known and due to the fact that the LFSR operates independent of the NFSR in the PRGA, the attacker can compute the LFSR state bits of S_t by simply running the Grain PRGA forward for t rounds and thus compute the entire of S_t.

5.4.2 Finding the secret key and complexity of the attack

It is known that the KSA, PRGA routines in the Grain family are invertible (see Algorithms 4.1, 4.2). Once we have all the bits of S_t, by running the PRGA^{-1} (inverse PRGA) routine for t rounds one can recover S_0. Thereafter the KSA^{-1} (inverse KSA) routine can be used to find the secret key.

The attack complexity directly depends on the number of re-keyings to be performed such that approximately all of locations in $[0, n-1]$ of the LFSR are covered. Since each re-keying is followed by exactly one fault injection, the expected number of fault injection is $2n \cdot \sum_{i=1}^{n} \frac{1}{i} \approx 2n \cdot \ln\ n$ (this is approximately equal to $2^{9.45}$ for $n = 80$ and $2^{10.3}$ for $n = 128$). Thereafter, the attack requires one matrix multiplication between an $n \times n$ matrix and an $n \times 1$ vector to recover the LFSR, and solving a few equations to get the NFSR state. After this, t invocations of the PRGA^{-1} and a single invocation of the KSA^{-1} gives us the secret key.

Note that, construction of the matrix L and running the $\text{FLE}_L(\lambda)$ and $\text{FLE}_N(\lambda)$ can be done beforehand and thus do not add to the attack complexity. However, these routines are a part of the pre-processing phase, the exact runtime of which will depend on the nature of the functions g, h and also the choice of taps used in the cipher design.

5.4.3 Attacking the actual ciphers

Now we will provide the details of the actual attack on Grain v1, Grain-128 and Grain-128a.

Grain v1

In Grain v1 the non linear combining function is of the form

$$h(s_0, s_1, s_2, s_3, s_4) = s_1 \oplus s_4 \oplus s_0 s_3 \oplus s_2 s_3 \oplus s_3 s_4 \oplus s_0 s_1 s_2 \oplus s_0 s_2 s_3 \oplus s_0 s_2 s_4 \oplus s_1 s_2 s_4 \oplus s_2 s_3 s_4.$$

Here only s_4 corresponds to an NFSR variable. This function has 4 affine differential tuples of weight 1, only one of which ($[\mathbf{0}, \alpha = 11001]$) leads to a derivative which is a function of only LFSR variables. However, $\widetilde{\Upsilon}_{t,\phi} = \alpha$ and $\chi_{t,\phi} \sqsubseteq 1$ does not hold for any t or ϕ. Hence one needs to look at higher weight tuples.

A call to $\text{FLE}_L(3)$ returns 78 linearly independent equations. The result is given in Table 5.2.

t	ϕ_1	ϕ_2	ϕ_3	Range	Expression	ADT
$45+i$	$62+i$	$24+i$	$70+i$	$i \in [0,9]$	y^t_{46}	00000, 00100, 00110, 01000
$55+i$	$72+i$	$16+i$	$51+i$	$i \in [0,7]$		
$63+i$	$13+i$	$24+i$	$59+i$	$i \in [0,9]$		
$73+i$	$33+i$	$26+i$	$51+i$	$i \in [0,10]$		
$84+i$	$44+i$	$37+i$	$38+i$	$i \in [0,6]$		
$91+i$	$53+i$	$44+i$	$41+i$	$i \in [0,8]$		
$100+i$	$70+i$	$53+i$	$60+i$	$i \in [0,8]$		
109	79	71	69			
$77+i$	$45+i$	$51+i$	$38+i$	$i \in [0,5]$	$y^t_3 \oplus y^t_{25} \oplus y^t_{64}$	00000, 01100, 10000, 10110
$83+i$	$72+i$	$57+i$	$44+i$	$i \in [0,4]$		
94	62	79	55			
95	78	63	56		$y^t_3 \oplus y^t_{25} \oplus y^t_{46} \oplus y^t_{64}$	00000, 01001, 01100, 10110

TABLE 5.2: Output of $\text{FLE}_L(3)$ for Grain v1 (ADT implies Affine Differential Tuple)

t	ϕ_1	ϕ_2	Range	Expression	ADT
$110+i$	$64+i$	$77+i$	$i \in [0,1]$	y^t_{46}	00001, 11000

TABLE 5.3: Output of $\text{FLE}_L(2)$ for Grain v1

A call to $\text{FLE}_L(2)$ gives us the 2 other equations required to solve the system. The result is shown in Table 5.3. One can verify that the linear equations so obtained are linearly independent and thus LFSR can be solved readily. A call each to $\text{FLE}_N(1)$ and

t	ϕ_1	Range	Expression	ADT
$55+i$	$23+i$	$i \in [0,14]$	$1 \oplus y_3^t \oplus y_{46}^t \oplus x_{63}^t$	00000, 01010
$70+i$	$77+i$	$i \in [0,2]$		
$91+i$	$62+i$	$i \in [0,5]$		

TABLE 5.4: Output of $\text{FLE}_N(1)$ for Grain v1

t	ϕ_1	ϕ_2	ϕ_3	Range	Expression	ADT
$17+i$	i	$1+i$	$20+i$	$i \in [0,27]$	$1 \oplus y_3^t \oplus y_{46}^t \oplus x_{63}^t$	00000, 00001, 00010, 10000
$45+i$	$28+i$	$13+i$	$48+i$	$i \in [0,9]$		
$73+i$	$53+i$	$33+i$	$26+i$	$i \in [0,17]$	$1 \oplus y_3^t \oplus x_{63}^t$	00000, 00010, 00100, 00110

TABLE 5.5: Output of $\text{FLE}_N(3)$ for Grain v1

$\text{FLE}_N(3)$ gives us all the NFSR bits of S_{80}. The output of these routines are given as in Tables 5.4 and 5.5. A look at these tables shows that the attacker can calculate the values of x_{63}^t for all $t \in [17, 96]$. This is equivalent to calculating x_i^{80} for all $i \in [0, 79]$. Thereafter, S_0 and the secret key may be obtained as per the techniques outlined in Section 5.4.2.

Grain-128

In Grain-128 the non linear combining function is of the form $h(s_0, s_1, \ldots, s_8) = s_0 s_1 \oplus s_2 s_3 \oplus s_4 s_5 \oplus s_6 s_7 \oplus s_0 s_4 s_8$. Only s_0, s_4 correspond to the NFSR variables. This function has 4 affine differential tuples of weight 1 which produce derivatives on LFSR variables. A call to $\text{FLE}_L(1)$ produces all the 128 equations needed to solve the LFSR. The output of this routine is given in Table 5.6.

t	ϕ_1	Range	Expression	ADT
i	$20+i$	$i \in [0,107]$	y_{13}^t	000 000 000, 000 100 000
$61+i$	$50+i$	$i \in [0,19]$	y_{60}^t	000 000 000, 000 000 010

TABLE 5.6: Output of $\text{FLE}_L(1)$ for Grain-128

t	ϕ_1	Range	Expression	ADT
i	$8+i$	$i \in [0, 115]$	x_{12}^t	000 000 000, 010 000 000
$33+i$	$75+i$	$i \in [0, 11]$	x_{95}^t	000 000 000, 000 001 000

TABLE 5.7: Output of $\text{FLE}_N(1)$ for Grain-128

A call to $\text{FLE}_N(1)$ gives us all the NFSR bits of S_{12}. The output of this routine is in Table 5.7. Thus, $\text{FLE}_N(1)$ gives us x_{12}^t for all $t \in [0, 115]$, and x_{95}^t for all $t \in [0, 11]$. This is equivalent to all the NFSR state bits of S_{12}. Thereafter, S_0 and the secret key may be obtained as per the techniques outlined in Section 5.4.2.

Grain-128a

In Grain-128a, the first 64 key-stream bits and every alternate key-stream bit thereof are used to construct the message authentication code and therefore unavailable to the attacker. To resolve this problem, in Grain-128a every re-keying is followed by a fault injection at the beginning round 64 of the PRGA instead of round 0 and the goal of the attacker is to reconstruct the internal state at the 64^{th} instead of the 0^{th} PRGA round. Note that if a fault is applied at a random LFSR location ϕ at the 64^{th} PRGA round, then the t^{th} state of Δ_ϕ-Grain will align itself with the $(64+t)^{th}$ state of the actual cipher. Hence, in a slight departure from the notation introduced in the previous section we will call the 64^{th} PRGA state S_0 and all other notations are shifted with respect to t accordingly (e.g., S_t refers to the $(64+t)^{th}$ PRGA state etc).

The key-stream bit at every odd numbered round (after round 64 of the PRGA) is used for making the MAC and is unavailable to the attacker. Hence after calling $\text{FLE}_L(1)$ the attacker must reject all outputs with an odd value of t. Even then the attacker obtains all the equations required to solve the LFSR. The output is presented in Table 5.8. Similarly a call to $\text{FLE}_N(1)$ after rejecting outputs with odd values of t, gives us 112 NFSR bits of S_{62}. The output is given in Table 5.9.

t	ϕ_1	Range	Expression	ADT
$6+2i$	$26+2i$	$i \in [0, 50]$	y_{13}^t	000 000 000, 000 100 000
$108+2i$	$70+2i$	$i \in [0, 12]$		
$2i$	$13+2i$	$i \in [0, 33]$	y_{20}^t	000 000 000, 001 000 000
$28+2i$	$107+2i$	$i \in [0, 10]$	y_{60}^t	000 000 000, 000 000 010
$50+2i$	$1+2i$	$i \in [0, 18]$		

TABLE 5.8: Output of $\text{FLE}_L(1)$ for Grain-128a

t	ϕ_1	Range	Expression	ADT
$50+2i$	$58+2i$	$i \in [0,34]$	x_{12}^t	000 000 000, 010 000 000
$120+2i$	$96+2i$	$i \in [0,15]$		
$152+2i$	$102+2i$	$i \in [0,12]$		
$2i$	$42+2i$	$i \in [0,42]$	x_{95}^t	000 000 000, 000 001 000
$86+2i$	$38+2i$	$i \in [0,4]$		

TABLE 5.9: Output of $\text{FLE}_N(1)$ for Grain-128a

At this point, the attacker could simply guess the remaining 16 bits of S_{62} or give a call to $\text{FLE}_N(2)$ and thus increase the complexity of the preprocessing stage. As it turns out, the attacker can do even better without going for these two options. The 16 NFSR bits not determined at this point are x_{2i+1}^{62}, for $0 \leq i \leq 15$. Let us now look at the equations for the key-stream bits z_{62+2j} for $j \in [0,8]$,

$$z_{62+2j} = \bigoplus_{i \in \mathcal{B}} x_{i+2j}^{62} \oplus x_{15+2j}^{62} \oplus y_{93+2j}^{62} \oplus h(\theta_{62+2j}),$$

where $\mathcal{B} = \{2, 36, 45, 64, 73, 89\}$. Now, $x_{15+2j}^{62},\ j \in [0,8]$ is the only unknown in each of these equations and so its value can be calculated immediately. This leaves us with the 7 unknown bits $x_1^{62}, x_3^{62}, \ldots, x_{13}^{62}$. In addition to the entries in Table 5.9, $\text{FLE}_N(1)$ also gives the output

$$t = 96+2i, \phi_1 = 48+2i, x_{95}^t, [\mathbf{0}, 000\ 001\ 000],\ \forall i \in [0,6].$$

This gives us the bits x_{95}^{96+2i} or equivalently x_{127}^{64+2i} for $i \in [0,6]$. Let us write the NFSR update function g in the form $g(X) = x' \oplus g'(X)$, where x' corresponds to the variable that taps the 0^{th} NFSR location. Then looking at the NFSR update rule for Grain-128a, we have

$$x_{127}^{64+2i} = y_0^{63+2i} \oplus x_0^{63+2i} \oplus g'(X_{63+2i}) = y_{1+2i}^{62} \oplus x_{1+2i}^{62} \oplus g'(X_{63+2i}),$$

$\forall i \in [0,6]$. Again, $x_{1+2i}^{62},\ i \in [0,6]$ is the only unknown in these equations and so its value can be calculated immediately. This gives us all the NFSR bits of S_{62}. Using the techniques in Section 5.4.2, S_0 can be calculated. Since this state corresponds to the 64^{th} PRGA state, the PRGA^{-1} routine needs to be run 64 more times before invoking the KSA^{-1} routine which would then reveal the secret key.

5.5 DFA against Grain family with very few faults and minimal assumptions

In this Section, we describe our final attack on the Grain family of stream ciphers. The adversary, in this attack, is assumed to be equipped with the minimum powers, i.e.,

- He can inject an unsynchronized fault at some random register location, but there is no guarantee that it would flip the logic at a single register location.
- He can reset the device implementing the cipher and restart cipher operations afresh.

One of the striking features of the attack we will discuss here, is the drastic reduction in the number of faults required to mount the attack. Whereas the previous attacks in Sections 5.3, 5.4 required in excess of 2^9 faults, the attack we describe here will take less than 10 faults for all the three ciphers in the Grain family.

In the previous attacks that we have described, the technique of attack has been roughly the same:

1. Find the location of a randomly applied fault.
2. Use this information to formulate linear equations on the initial LFSR state.
3. If sufficient number of linear equations have been generated, the solve for the LFSR state.
4. Once the LFSR state is known, formulate some more linear equations to get the NFSR state.

Note that the attack protocol relies heavily on the formulation of linear equations to solve for the LFSR/NFSR states. Since a fault at a random register location does not always lead to a linear equation, our previous attack techniques relied on rekeying and reinjecting faults until sufficient number of linear equations are found. This is one of the reasons the fault count in the previous attacks is slightly on the higher side. On the other hand, every fault does lead to a new set of non-linear equations on the LFSR/NFSR state resulting from each faulty keystream bit. So, far we have not taken advantage of this enormous bank of non-linear equations. If we could utilize an equation solver that could solve these non-linear equations, we would the be able to think in terms of reducing the fault requirement.

For more than a decade, there has been seminal research in the area of algebraic cryptanalysis, the principal idea of which is to solve multivariate polynomial systems that describe a cipher. For a very brief introduction in this, one may refer [109, Section 5]. The DFA on Trivium [109] requires only 2 faults and this is far fewer than the fault requirements against the Grain family. This motivates us to see how this kind of algebraic cryptanalysis can be exploited towards DFA against Grain family.

SAT solvers have been used extensively in algebraic cryptanalysis [29]. These solvers are based on the Boolean satisfiability problem which is basically the problem of determining if there exists an interpretation that satisfies a given Boolean formula. In other words, it establishes if the variables of a given Boolean formula can be assigned in such a way as to make a given Boolean formula evaluate to TRUE. In this attack, we have made use of the SAT Solver Cryptominisat-2.9.5 [126] installed with SAGE 5.7 [129] to solve the bank of equations that we get from each faulty keystream bit. In our experience, given sufficient number of equations, the solver is able to come up with a solution in a time ranging from a few minutes to a few hours.

In order to explain the flow of the attack we will first assume that the attacker can indeed apply a time synchronized, single bit-flipping fault at a random register location. We will later (in Sections 5.6.1, 5.6.2) discuss the issues related to the following:

- We will first deal with the case when the attacker can apply a time synchronized fault that may disturb the logic in up to 3 continuous register locations. We will explain how an attacker can distinguish, with high probability, whether a faulty keystream segment has been produced due to a multiple bit or a single bit-flipping of the original faultless internal state. In the event that the attacker finds a faulty keystream has been produced due to a multiple bit-flip, he will simply discard the keystream, and inject fault afresh so that he obtains a faulty keystream produced due to a single bit-flip whose location he can conclusively identify.
- We will then investigate the case when the attacker cannot fully synchronize the timing of his fault with the start of the PRGA, and the best he can do is inject the fault at some PRGA round $\tau \leq \tau_{max}$. We will outline how the attacker can find the value of τ and proceed with the attack.

So initially, let us for the time being consider that the fault will be injected after the KSA, i.e., just before the PRGA starts. We will begin by explaining how the attacker builds up a bank of multivariate equations on the LFSR/NFSR state variables before feeding it to the SAT Solver.

5.5.1 Populating the bank of equations for Grain v1 and Grain-128

We will now explain the method of obtaining a large number of equations that will be used for algebraic cryptanalysis. For the time being we will consider the case that will work for Grain v1 or Grain-128. The case of Grain-128a will be little different that we will discuss next.

Equations from fault-free key-stream

Consider the equations from the ℓ-bit fault-free key-stream $z_0, \ldots, z_{\ell-1}$. As discussed, the LFSR state just after the KSA (at the beginning of the 0-th clock) is given by $Y_0 = [y_0, y_1, \ldots, y_{n-1}]$ and the NFSR state is $X_0 = [x_0, x_1, \ldots, x_{n-1}]$. In general, the value of ℓ required to complete the attack is more than 160 for all the three versions of Grain. But it is not feasible to compute the Algebraic Normal Form (ANF) of z_{159} on any standard PC, for any version of Grain. For example using a workstation with 1.83 GHz processor, 3 GHz RAM and 2 MB system cache, computing the ANF of any z_ℓ in Grain v1, for $\ell > 44$ is infeasible. The ANF of z_{44} itself has algebraic degree 17 and consists of 80643 monomials.

Also we have to keep in mind that we expect to solve these solutions using SAT solver. SAT solvers solve a polynomial equation system by converting the ANF′s to their equivalent Conjunctive Normal Forms (CNF′s). As we will see in Section 5.5.3, the representation of each degree d monomial requires $d + 1$ CNF clauses. Thus to enable the SAT solver to solve the system efficiently, the degrees of the expressions in the equation system must also be controlled.

In order to overcome both limitations, we use a technique popularly used in ANF-CNF conversions [29]. At each PRGA round $t > 0$, we introduce two new variables y_{t+n}, x_{t+n} to update the LFSR and NFSR state respectively. To illustrate the technique, let us denote the states at the beginning of the t-th ($t \geq 0$) PRGA round as

$$Y_t = [y_t, y_{t+1}, \ldots, y_{t+n-1}], \quad X_t = [x_t, x_{t+1}, \ldots, x_{t+n-1}].$$

Given these, we formulate the following equations.

1. LFSR equation: $y_{t+n} = f(Y_t)$.
2. NFSR equation: $x_{t+n} = y_t \oplus g(X_t)$.

3. Key-stream equation:

$$z_t = \bigoplus_{i=0}^{n-1} b_i y_{t+i} \oplus \bigoplus_{i=0}^{n-1} a_i x_{t+i} \oplus h(y_t, \ldots, y_{t+n-1}, x_t, \ldots, x_{t+n-1}).$$

In the Grain family, while the first equation is linear, the degrees of the other two equations are also not very high. We initially start with $2n$ variables, $y_0, y_1, \ldots, y_{n-1}$ and $x_0, x_1, \ldots, x_{n-1}$. Then corresponding to each key-stream bit z_t, we introduce two new variables y_{t+n}, x_{t+n} and obtain three more equations. Thus we have in total $2n+2\ell$ variables and 3ℓ equations. The advantages of using such a technique are as follows.

- First of all it allows us to formulate the expression for z_ℓ (via a series of equations) for values of $\ell \geq 159$. Instead, if at each round $t > 0$, the variables y_{t+n}, x_{t+n} were replaced by their equivalent algebraic expressions in $y_0, y_1, \ldots, y_{n-1}$ and $x_0, x_1, \ldots, x_{n-1}$, this would never have been possible on an ordinary PC.
- Since the expressions in the LFSR and NFSR cells always stay linear, this allows us to control the algebraic degree and the number of monomials in each of the 3ℓ equations so obtained.

Equations from faulty key-streams

We use a similar technique to extract equations from faulty key-streams. Let us assume that a fault is injected in the LFSR location ϕ at PRGA round 0. The same method will work if the fault is injected in the NFSR. Since we re-key the cipher with the same Key-IV before injecting a fault, after fault injection we obtain the state $y_0, y_1, \ldots, y_{\phi-1}, 1 \oplus y_\phi, y_{\phi+1} \ldots, y_{n-1}$ and $x_0, x_1, \ldots, x_{n-1}$ at the start of PRGA. Then corresponding to each key-stream bit z_t, we introduce two new variables $y^\phi_{t+n}, x^\phi_{t+n}$ and obtain three more equations. Thus we have additional 2ℓ variables and 3ℓ equations.

Total number of variables and equations

Given that we introduce ν faults after these re-keyings, the total number of variables is $2n + 2(\nu+1)\ell$ and the total number of equations is $3(\nu+1)\ell$.

5.5.2 Populating the bank of equations for Grain-128a

We will now explain the formation of equations for Grain-128a. Here the first 64 key-stream bits $z_0, \ldots, z_{63}$ and every other (alternating) key-stream bits thereafter are used to construct MAC. Hence these bits are unavailable to the attacker.

Equations from fault-free key-stream

Consider the equations from the ℓ-bit fault-free key-stream $z_{64}, z_{66}, \ldots, z_{64+2\ell-2}$ as only alternative key-stream bits are used for encryption. Hence, similar to the above, we have the following equations.

1. Two LFSR equations: $y_{t+n} = f(Y_t)$ and $y_{t+n+1} = f(Y_{t+1})$.
2. Two NFSR equations: $x_{t+n} = y_t \oplus g(X_t)$ and $x_{t+n+1} = y_{t+1} \oplus g(X_{t+1})$.
3. One Key-stream equation:

$$z_t = \bigoplus_{i=0}^{n-1} b_i y_{t+i} \oplus \bigoplus_{i=0}^{n-1} a_i x_{t+i} \oplus h(y_t, \ldots, y_{t+n-1}, x_t, \ldots, x_{t+n-1}).$$

We initially start with $2n$ variables, $y_0, y_1, \ldots, y_{n-1}$ and $x_0, x_1, \ldots, x_{n-1}$. Then corresponding to each key-stream bit z_t, we introduce four new variables y_{t+n}, y_{t+n+1}, x_{t+n}, x_{t+n+1} and obtain five more equations. Thus we have in total $2n + 4\ell$ variables and 5ℓ equations.

Equations from faulty key-streams

Let us consider that a fault is injected in the LFSR location ϕ at the beginning of the PRGA. Again, the method works similarly if the fault is injected in the NFSR. Since we will re-key the cipher with the same Key-IV, in such a case we will obtain the state $y_0, y_1, \ldots, y_{\phi-1}, 1 \oplus y_\phi, y_{\phi+1} \ldots, y_{n-1}$ and $x_0, x_1, \ldots, x_{n-1}$. Then corresponding to each key-stream bit z_t, we introduce four new variables $y^\phi_{t+n}, y^\phi_{t+n+1}, x^\phi_{t+n}, x^\phi_{t+n+1}$ and obtain five more equations. Thus we have additional 4ℓ variables and 5ℓ equations.

Total number of variables and equations

Given that we introduce ν faults after these re-keyings, the total number of variables is $2n + 4(\nu + 1)\ell$ and the total number of equations is $5(\nu + 1)\ell$. All these equations are used in the SAT solver to obtain $y_0, y_1, \ldots, y_{n-1}$ and $x_0, x_1, \ldots, x_{n-1}$. This completes the attack. As the ciphers in the Grain family are invertible both in KSA and PRGA, one can also get the secret key efficiently.

5.5.3 Using the SAT Solver

To solve polynomial systems of multivariate equations by SAT solvers, the attacker initially converts the system from Algebraic Normal Form (ANF) to Conjunctive Normal Form (CNF). We will show some standard techniques of how this can be done.

Conversion of monomials: Any monomial of the form $x_1x_2\cdots x_d$ is first equated to another variable β (say). The tautological equivalent of $\beta = x_1x_2\cdots x_d$ is $\beta \Leftrightarrow x_1x_2\cdots x_d$, which is same as the boolean expression β XNOR $x_1x_2\cdots x_d$. This therefore can be expressed as

$$(\overline{\beta} \vee x_1) \wedge (\overline{\beta} \vee x_2) \wedge \cdots \wedge (\overline{\beta} \vee x_d) \wedge (\beta \vee \overline{x_1} \vee \overline{x_2} \vee \cdots \vee \overline{x_d})$$

As can be seen this adds $d+1$ clauses to the system each of which need to be TRUE for the correct solution.

Conversion of linear expressions: A linear system of the form $x_1+x_2+\ldots+x_k = 1$ can be expressed equivalently as the boolean expression x_1 XOR x_2 XOR $\cdots x_k$. For example $x_1 + x_2 + x_3 + x_4 = 1$ can be expressed as

$$(x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee \overline{x_3} \vee \overline{x_4}) \wedge (x_1 \vee \overline{x_2} \vee x_3 \vee \overline{x_4}) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4) \wedge$$

$$(\overline{x_1} \vee \overline{x_2} \vee x_3 \vee x_4) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3} \vee x_4) \wedge (\overline{x_1} \vee x_2 \vee x_3 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$$

A linear system of the form $x_1 + x_2 + \ldots + x_k = 0$ can be expressed equivalently as the boolean expression $(x_1$ XOR x_2 XOR $\cdots x_k)'$.

Conversion of large XOR chains: In the above system, the number of clauses obtained depends on the value k. It can easily be shown that this number is 2^{k-1}. To prevent accumulation of such large chain of clauses, we introduce dummy variables γ_i at each stage. For example, $x_1 + x_2 + \ldots + x_k = 0$ is equivalent to:

$$x_1 + x_2 + x_3 + \gamma_1 = 0, \quad \gamma_1 + x_4 + x_5 + \gamma_2 = 0, \quad \cdots \gamma_M + x_{k-2} + x_{k-1} + x_k = 0.$$

This gives rise to $M = \frac{k}{2} - 1$ equations each having $2^{4-1} = 8$ clauses.

Example 5.3. *To solve the equation system* $x_1x_2+x_2x_3+x_4 = 0$, $x_2x_3+x_3+x_1 = 0$, *we translate the system into the following form* **[1]** $\beta_1 = x_1x_2$, **[2]** $\beta_2 = x_2x_3$, **[3]** $\beta_1 + \beta_2 + x_4 = 0$, **[4]** $\beta_2 + x_3 + x_1 = 0$.

[1] *gives us* $(\overline{\beta_1} \vee x_1) \wedge (\overline{\beta_1} \vee x_2) \wedge (\beta_1 \vee \overline{x_1} \vee \overline{x_2}) =$ *TRUE.*

[2] *gives us* $(\overline{\beta_2} \vee x_2) \wedge (\overline{\beta_2} \vee x_3) \wedge (\beta_2 \vee \overline{x_2} \vee \overline{x_3}) =$ *TRUE.*
[3] *gives us* $(\beta_1 \vee \beta_2 \vee \overline{x_4}) \wedge (\beta_1 \vee \overline{\beta_2} \vee x_4) \wedge (\overline{\beta_1} \vee \beta_2 \vee x_4) \wedge (\overline{\beta_1} \vee \overline{\beta_2} \vee \overline{x_4}) =$ *TRUE.*
[4] *gives us* $(x_1 \vee \beta_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{\beta_2} \vee x_3) \wedge (\overline{x_1} \vee \beta_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{\beta_2} \vee \overline{x_3}) =$ *TRUE.*

After the system of algebraic equations have been converted to their equivalent CNF, they are passed on to the SAT solver for extracting a solution.

5.6 Experimental Results

In this section we present the experimental results in detail. After the fault location and injection time of a particular faulty key-stream vector have been identified using the signature vectors, a system of equations are formulated using the steps outlined in Section 5.5.1, and the equations are then fed into a SAT solver. There are several issues that have been optimized.

- The number of faults is the most significant figure that we minimize using the SAT solvers. This implies that we also reduce the number of re-keyings of the cipher.
- The number of faulty key-stream bits required to solve the system is also important. In our experiments, We have used $2n$ key-stream bits corresponding to each fault, i.e., $2 \cdot 80 = 160$ for Grain v1 and $2 \cdot 128 = 256$ for Grain-128 and Grain-128a. In fact, for Grain-128a, we use even fewer key-stream bits as we obtain more equations per key-stream bit.

We have solved the equations using SAT solver Cryptominisat-2.9.5 [126] installed with SAGE 5.7 on Linux Ubuntu 2.6. The hardware platform is an HP Z800 workstation with 3GHz Intel(R) Xeon(R) CPU. We have considered three different cases: (i) the faults are introduced in LFSR only, (ii) the faults are introduced in NFSR only, and (iii) the faults are introduced in both LFSR and NFSR (here we consider that expected half of the faults are injected in LFSR and the other half in NFSR). The results have been presented in Table 5.10. We have presented the time required for the SAT solver part only as the time for identifying the location of the fault using signature vectors is negligible. For each row, we consider a set of ten (10) experiments. As it is not easy to count the exact number of computational steps required in the SAT solver, we have reported the amount of time required in seconds.

We note that with very little amount of key-stream, the attack takes longer time. The number of faults may be reduced further with more computational effort.

TABLE 5.10: Experimental Results

Faults in LFSR only					
Cipher	Number of faults	Amount of key-stream	Time (in sec.)		
			Minimum	Maximum	Average
Grain v1	10	160	16.48	49.23	27.40
	9	160	22.10	32.71	40.50
	8	160	18.62	92.34	48.40
Grain-128	5	256	5.21	9.43	7.10
	4	256	9.03	96.68	34.40
	3	256	24.52	361.53	163.70
Grain-128a	11	175	14.47	37.85	23.60
	10	175	26.82	253.15	52.74
Faults in NFSR only					
Cipher	Number of faults	Amount of key-stream	Time (in sec.)		
			Minimum	Maximum	Average
Grain v1	11	160	27.93	105.44	55.35
	10	160	21.14	89.50	43.64
	9	160	29.64	123.98	56.35
Grain-128	6	256	16.64	196.32	93.45
	5	256	22.87	380.01	147.70
Grain-128a	11	175	179.62	8453.14	1542.27
	10	175	175.07	8387.21	1495.54
Faults in both LFSR and NFSR					
Cipher	Number of faults	Amount of key-stream	Time (in sec.)		
			Minimum	Maximum	Average
Grain v1	11	160	54.96	1420.71	220.90
	10	160	19.17	452.30	352.20
Grain-128	6	256	6.48	14.32	10.41
	5	256	12.18	37.56	22.15
	4	256	27.63	4876.53	581.80
Grain-128a	11	175	46.45	259.34	101.10
	10	175	69.63	5144.56	1472.35

5.6.1 Identifying Multiple bit faults

So far we have discussed an attack scenario where an injected fault flips exactly one bit value at a random register location. We now relax the requirements of the attack, and assume a fault model that allows the attacker to inject a fault that affects more than one locations. Our strategy would be that, if it is found that the fault injection has flipped the logic at more than one register location, we will discard the keystreams and not use them for further processing.

We consider the case when at most three consecutive locations can be disturbed by a single fault injection. Thus, three cases are possible:

(a) exactly one bit is flipped (n cases each for the LFSR and NFSR and hence a total on $2n$ cases),

(b) 2 consecutive locations $i, i+1$ are flipped $(2(n-1)$ cases),

(c) 3 consecutive locations $i, i+1, i+2$ are flipped $(2(n-2)$ cases) .

Studying such a model makes sense if we attack an implementation of Grain where the register cells are physically positioned linearly one after the other.

It is clear that such a fault model allows a total of $2(n+n-1+n-2) = 6n-6$ types of faults out of which only n are single bit-flips. We assume that each of these $6n-6$ cases are equally probable. The success of our attack that we have described in Section 5.5 will depend on the ability of the attacker to deduce whether a given faulty key-stream vector has been produced as a result of a single bit toggling of any register location or a multiple-bit toggle. Thus, we need to design a fault location identification algorithm that analyzes a faulty key-stream and (i) if the faulty key-stream has been produced due to a single bit toggling of any location, the algorithm should output that particular position, and (ii) if the faulty key-stream has been produced due to multiple-bit toggling of locations, the algorithm should infer that the faulty key-stream could not have been produced due to a single bit toggle.

To solve the problem, will use the same fault location identification routine FLocI described in Algorithm 5.2. For the method to be a success, the routine would

- Return the fault location numbers for all possible cases when a single location is toggled (n out of $6n-6$ cases), i.e., when $|\Psi_{3,\phi}| = 1$
- Return the failure message $\perp$ in case $|\Psi_{3,\phi}| > 1$,
- Return the message $\emptyset$ if $|\Psi_{3,\phi}| = 0$, which is a conclusive identification of a multiple bit-flip.

After experimenting with randomly chosen single, double and triple bit faults for around 2^{20} Key-IV pairs, it was found that the probability that the algorithm successfully rejects a faulty stream produced due to a multiple bit fault i.e. $Pr(\Psi_{3,\phi} = \emptyset)$ is 0.94 for Grain v1, 0.99 Grain-128 and 0.86 for Grain-128a.

5.6.2 Identifying Fault Locations for Injections at random time

Until now we have assumed that the adversary is able to inject all faults at the beginning of a fixed PRGA round. This is usually practical as fault injections are usually synchronized with the power consumption curves of the device implementing the cryptosystem [52]. In this section we show that it is possible to attack Grain even if this

requirement is relaxed. We will show that if the adversary injects a fault at a PRGA round τ where $\tau \in [0, \tau_{max} - 1]$. In such an event, it is possible for the adversary, with high probability, determine the values of the fault location ϕ and the injection time τ. Before we get into further details, let us recap a few things and look at a definition that we will be using extensively.

The location identification algorithm that presented so far (call it $\mathsf{FLocl}(E^\phi)$ takes the difference vector $E^\phi = Z \oplus Z^\phi$, and returns the following

- The fault location ϕ if the set $\Psi_{3,\phi}$ has cardinality 1.
- The $\emptyset$ message if the set $\Psi_{3,\phi}$ has cardinality 0, which is indicative of the fact that Z^ϕ was generated due to multiple bit fault.
- A failure message $\perp$ if the set $\Psi_{3,\phi}$ has cardinality strictly greater than 1. This case may arise for both single and multiple bit faults.

Definition 5.7. Two distinct fault location and time injection pairs (ϕ, τ) and (ϕ', τ') are said to be equivalent if they produce the same faulty key-stream.

For example in Grain v1, faulting the NFSR location 70 at PRGA round 0 would produce the same faulty key-stream as faulting NFSR location 69 at PRGA round 1. This is because the difference that is induced in location 70 at PRGA round 0 shifts to location 69 in PRGA round 1 anyway. Thus $(70, 0)$ and $(69, 1)$ are equivalent pairs. However $(62, 0)$ and $(61, 1)$ are not equivalent since 62 is a tap for the update function of the NFSR for Grain v1. A difference induced in PRGA round 0 in location 62 travels to both locations 61 and 79 in the next round. Whereas a fault at location 61 in round 1 would affect only this location and not location 79.

Let us denote the elements of $E^\phi = [e_0, e_1, e_2, \ldots]$. Also define the vector $E_i^\phi = [e_i, e_{i+1}, \ldots]$. Let us assume that the vector E^ϕ has been produced due to fault injection at some LFSR or NFSR location ϕ at time τ where $0 \leq \tau \leq \tau_{max} - 1$. To identify (ϕ, τ) the adversary runs the routine $\mathsf{FLI}(E_i^\phi)$ for all $i \in [0, \tau_{max} - 1]$. As a result, the adversary could obtain

1. The output $\phi + i$ for all values of i. Note that since the pairs $(\phi + i, i)$ are equivalent, he can assume that $(\phi, 0)$ are the true values of (ϕ, τ).
2. The output $\phi + i$ for some values of i and a failure messages for some other values of i. The adversary then takes the minimum value of $i = i_{min}$ for which $\mathsf{FLI}(E_i^\phi)$ succeeds and assumes $(\phi + i_{min}, i_{min})$ to be the true values of (ϕ, τ).

3. The failure message for all values of i. In this event he rejects the key-stream. However, the probability of this outcome is quite low.

4. If he obtains $\emptyset$ for some value of i he deduces multiple-bit injection and rejects the key-stream.

5. If he obtains the outputs ϕ_1 for $i = i_1$ and ϕ_2 for $i = i_2$ such that (ϕ_1, i_1) and (ϕ_2, i_2) are not equivalent then he deduces that the algorithm has failed and rejects the keystream.

Experiments performed for around 2^{20} random Key-IVs the probability of Case 5 occurring is only about 0.089 for Grain v1 if we take $\tau_{max} = 10$. For for Grain-128, taking $\tau_{max} = 15$, the failure probability comes to 0.079. For higher values of τ_{max} the failure probability becomes non-negligible.

This approach, however, fails for Grain-128a. Recall, that every alternate key-stream bit in Grain-128a is used for the computation of MAC and is therefore not directly available to the attacker. It is easy to see that the given approach will fail in all cases when the injection time is an odd number.

5.7 Conclusion

In this chapter, the Differential Fault Analysis (DFA) against the Grain family of stream ciphers has been studied under various fault models – some more restrictive and some more relaxed. We have proposed three attacks on the Grain family of stream ciphers, each of which imposes increasing degrees of difficulty on the attacker, and outlined methods that enables the attacker to recover the Secret Key under each of these conditions. In the first attack, we assume that the attacker is able to inject time-synchronized, single bit-flipping faults in the same albeit random register location. The attacker uses a linear first order derivative of the output function h used in the Grain family to formulate linear equations and recover the internal state of the cipher at the beginning of the PRGA. In the second attack, the attacker is no longer allowed to inject multiple faults on the same register location. In this case, he uses higher order affine derivatives of the output function h to formulate linear equations and recover the internal state.

In third and final attack, we propose a DFA of the Grain family that requires the adversary to have the least control over fault injections, i.e., he exercises only partial control over the time of injection, and he can only guarantee that the fault he has injected affects no more than 3 continuous register locations. The algorithm we propose first finds the location and injection time of a randomly applied bit fault (it rejects

the faulty stream if it infers that it was produced due to multiple bit fault) and then populates a bank of equations in the internal state variables of the cipher at the start of the PRGA. The algorithm then tries to solve the equations using the Cryptominisat-2.9.5 SAT solver [126]. For all the three ciphers the solver is able to recover the entire internal state using equations generated by less than or equal to 10 random faults in a few minutes. This is, to the best of our knowledge, the best fault analysis that has been reported against the Grain family.

As we have pointed out, the number of faults may be reduced further with more computational effort. Dedicated hardware and parallel computation may be exploited in this direction. However, this is not in the scope of this work as we are interested in the proof-of-concept that can be achieved in a few minutes through a simple implementation. Estimating the minimum number of faults given some high-end hardware is an important open question for future research.

Chapter 6

Conditional Differential Cryptanalysis of Grain

As far as the Differential Cryptanalysis of reduced round Grain v1 is concerned, the best results were those published by Knellwolf et al. in Asiacrypt 2011. In an extended version of the paper, it was shown that it was possible to retrieve **(i)** 5 expressions in the Secret Key bits for a variant of Grain v1 that employs 97 rounds (in place of 160) in its Key Scheduling process using 2^{27} chosen IVs and **(ii)** 1 expression in Secret Key bits for a variant that employs 104 rounds in its Key Scheduling using 2^{35} chosen IVs. The authors had arrived at the values of these Secret Key expressions by observing certain biases in the keystream bits generated by the chosen IVs. These biases were observed purely experimentally and no theoretical justification was provided for the same. In this chapter, we will revisit Knellwolf's attacks on Grain v1 and try to provide a theoretical framework that will serve to prove the correctness of these attacks. We will also look at open problems which may possibly pave way for further research on Differential Cryptanalysis of Grain v1.

6.1 Introduction

Cube attacks was first introduced by Dinur and Shamir in [55] and have been used extensively to attack reduced round variants of the Grain family. In [56, 57], cube attacks have been used to successfully cryptanalyze reduced-round variants as well as full Grain 128. In [97], cube distinguishers were used to distinguish a variant of Grain-128a, that employs 189 out of the 256 rounds in the Key Scheduling process. However, due to the relative complex nature of the component functions used in the design of Grain v1, there have not been many advances in this direction against it. The best

published work on Grain v1 is by Knellwolf et al [94], an extended version of which appeared in [93, Chapter 3.4]. The attack, which can be best described as a dynamic cube attack over a single-dimensional cube, achieves the following objectives:

a) It retrieves 5 expressions in the Secret Key bits for a variant of Grain v1 that employs 97 rounds (in place of 160) in its Key Scheduling process using 2^{27} chosen IVs.

b) It retrieves 1 expression in Secret Key bits for a variant that employs 104 rounds in its Key Scheduling using 2^{35} chosen IVs.

The values of these Secret Key expressions were obtained by observing certain non-randomness in the keystream bits generated by the chosen IVs. More specifically, the authors could enumerate a set of IVs for which, the sum of the output bits over the single dimensional cube were biased towards 0. These biases were observed purely experimentally and no theoretical justification was provided for the same. Providing a theoretical explanation of these experimental observations has thus been an open problem in this domain.

In this chapter we will try to provide some answers to these questions which have thus far remained open. We will first briefly revisit the details of the attacks on Grain v1 described in [93, Chapter 3.4]. We will then describe a Differential Engine that will keep track of the differential trails in the Key Scheduling part of the cipher. Using this tool we will show that biases observed in the output cubes after 97 rounds respectively are due to unbalanced derivatives of the NFSR update function g and output function h used in the design of Grain v1, i.e., there exist differentials α, β for which the Boolean Functions $g(\mathbf{x}) \oplus g(\mathbf{x} \oplus \alpha)$ and $h(\mathbf{x}) \oplus h(\mathbf{x} \oplus \beta)$ are both unbalanced. For the attack on 104 rounds, the author of [93] observes that the bias is observed in only about 50% of the cases, and at this point it is not exactly clear what algebraic conditions the Secret Key needs to satisfy in order to observe the bias and perform the Key recovery.

6.2 Knellwolf's attack on Grain v1

The paper [94] by Knellwolf et al. at Asiacrypt 2011 remains the best published result in the field of cryptanalysis of Grain v1 in terms of the number of rounds attacked. We will describe a slightly modified version of the same attack that appeared in [93, Chapter 3.4]. As alluded to earlier, the attack can be described as a dynamic cube attack over a cube of dimension one. Grain v1 employs a 64 bit IV, and the 37^{th} IV bit was chosen as the cube variable. Algebraically, this is equivalent to analyzing two initializations of

the Grain v1 cipher, one with the initial state equal to

$$X_0 = [k_0, k_1, \ldots, k_{79}], \quad Y_0 = [\nu_0, \nu_1, \ldots, \nu_{37}, \ldots, \nu_{63}, 1, 1, \ldots, 1],$$

and the other with the initial state equal to

$$X_0' = [k_0, k_1, \ldots, k_{79}], \quad Y_0' = [\nu_0, \nu_1, \ldots, 1 \oplus \nu_{37}, \ldots, \nu_{63}, 1, 1, \ldots, 1].$$

where $K = [k_0, k_1, \ldots, k_{79}]$, $V = [\nu_0, \nu_1, \ldots, \nu_{37}, \ldots, \nu_{63}]$, and $V' = [\nu_0, \nu_1, \ldots, 1 \oplus \nu_{37}, \ldots, \nu_{63}]$ are the formal notations for the Secret Key and the two IVs that differ in the 37^{th} position. Let X_i, Y_i and X_i', Y_i' denote the NFSR, LFSR states at the i^{th} KSA round produced during the evolution of X_0, Y_0 and X_0', Y_0' respectively.

The two initializations by X_0, Y_0 and X_0', Y_0', thus, imply that at the beginning of the Key Scheduling Algorithm (KSA), a differential is introduced in the 37^{th} LFSR bit. It seems inevitable that as more and more KSA rounds are completed the difference would inevitably spread to the NFSR as well, i.e., there exists some i for which X_i and X_i' would no longer be algebraically equal. The strategy of the attackers, in [93], was to delay the inevitable and prevent the diffusion of the differential to the NFSR for as many KSA rounds as possible, by imposing certain algebraic conditions on the IV and Secret Key bits. As a result of this the attacker obtains several algebraic relations between the Secret Key bits and the IV bits that must be satisfied if the differential is to be contained in the LFSR for as long as possible. These relations may be of the following types :

Type 1: A relation of the form $F_1(V) = 0$, i.e., involving only the IV bits.

Type 2: A relation of the form $F_2(K, V) = 0$, i.e., involving both the Secret Key and the IV bits.

Now let the term z_t, z_t' respectively be used describe the output bit produced in the t^{th} KSA round by the Key-IV pair K, V and K, V' (note that when we try to cryptanalyze Grain v1 reduced to r KSA rounds, the values of the output bits z_t, z_t' for all $t < r$ are unavailable to the attacker). The attacker now analyzes the pair of simplified cipher initializations where the the differential originally introduced in the 37^{th} LFSR bit is prevented from propagating into the NFSR by imposing suitable **Type 1, 2** relations between the Key and IV bits. In such a simplified cipher, the attacker now tries to find some i for which the distribution of the sum $z_i \oplus z_i'$ shows some non-randomness. Based on this randomness the attacker tries to guess the values of one or several expressions in

the Secret Key bits. We will illustrate this attack paradigm with this concrete example as described below.

1. The attacker begins to analyze the two algebraic systems resulting from the initialization of Grain v1 by the Key-IV pairs K, V and K, V' respectively. Thus the attacker has to analyze the evolution of the difference between the states X_i, Y_i and X'_i, Y'_i for increasing values of i starting from 0, with $X_0 = K, Y_0 = V||\texttt{0x ffff}$ and $X'_0 = K, Y'_0 = V'||\texttt{0x ffff}$ as described above.

2. The attacker then looks at all KSA rounds t during which the differential could propagate to the NFSR. The first such instance occurs at round $t = 12$, when the difference originally introduced at LFSR bit 37 at $t = 0$, now sits in LFSR location 25 which feeds the output function h. Since during the KSA the NFSR is updated as $x_{t+n} = g(X_t) \oplus y_t \oplus z_t$, the difference generated between the updated NFSR bits x_{80+12} and x'_{80+12} is given by

$$\begin{aligned} x_{80+12} \oplus x'_{80+12} &= [\ g(X_{12}) \oplus y_{12} \oplus z_{12}\ ] \oplus [\ g(X'_{12}) \oplus y'_{12} \oplus z'_{12}\ ] \\ &= z_{12} \oplus z'_{12} = \nu_{15}\nu_{58} \oplus \nu_{58}k_{75} \oplus 1 \end{aligned}$$

   By algebraic calculation it can be verified that $X_{12} = X'_{12}$ and $y_{12} = y'_{12}$ and hence the above result follows. Now, the attacker must therefore set $x_{80+12} \oplus x'_{80+12} = \nu_{15}\nu_{58} \oplus \nu_{58}k_{75} \oplus 1 = 0$ to prevent the propagation of this differential. This can be achieved by setting $\nu_{58} \oplus 1 = 0$ and

$$C_1 :\ \nu_{15} \oplus K_1 = 0, \tag{6.1}$$

   where $K_1 = k_{75} \oplus 1$. Thus we obtain one **Type 1** relation and one **Type 2** relation.

3. The next instance of difference propagation occurs at KSA round $t = 34$. At this round, the difference generated between the updated NFSR bits x_{80+34} and x'_{80+34} is given by

$$\begin{aligned} x_{80+34} \oplus x'_{80+34} =& [\ g(X_{34}) \oplus y_{34} \oplus z_{34}\ ] \oplus [\ g(X'_{34}) \oplus y'_{34} \oplus z'_{34}\ ] = z_{34} \oplus z'_{34} \\ =& y_{98} \oplus y_{59}y_{80} \oplus y_{80}y_{98} \oplus y_{80}x_{97} \end{aligned}$$

   This difference can be nullified if we set $y_{98} = y_{80} = 0$. Now, both y_{98} and y_{80} are functions of $k_0, k_1, \ldots, k_{79}$ and $\nu_0, \nu_1, \ldots, \nu_{63}$ and hence $y_{98} = y_{80} = 0$ is satisfied if we impose the following conditions: $\nu_0 = 0, \nu_1 = 0, \nu_3 = 0, \nu_4 = 0, \nu_5 = 0, \nu_{21} = 0, \nu_{25} = 0, \nu_{26} = 0, \nu_{27} = 0, \nu_{43} = 0, \nu_{46} = 0, \nu_{47} = 0, \nu_{48} = 0$

$$C_2 : \ \nu_{13} \oplus \nu_{23} \oplus \nu_{38} \oplus \nu_{51} \oplus \nu_{62} \oplus K_2 = 0, \tag{6.2}$$

$$C_3 : \ \nu_2 \oplus \nu_{18} \oplus \nu_{31} \oplus \nu_{40} \oplus \nu_{41} \oplus \nu_{53} \oplus \nu_{56} \oplus K_3 = 0, \tag{6.3}$$

where

$$K_2 = k_1 \oplus k_2 \oplus k_4 \oplus k_{10} \oplus k_{31} \oplus k_{43} \oplus k_{56},$$

and K_3 is a polynomial expression of degree 7 with 39 monomials and 31 key variables.

4. The next instance is at $t = 40$. Again it can be verified that

$$\begin{aligned} x_{80+40} \oplus x'_{80+40} &= [g(X_{40}) \oplus y_{40} \oplus z_{40}] \oplus [g(X'_{40}) \oplus y'_{40} \oplus z'_{40}] = z_{40} \oplus z'_{40} \\ &= \nu_{43} y_{86} \oplus \nu_{43} \oplus y_{86} x_{103} \oplus y_{86} \oplus x_{103} \end{aligned}$$

The difference is nullified if we set $\nu_{43} = 0, y_{86} = 0$, and $x_{103} = 0$ for which the following conditions are imposed: $\nu_8 = 0, \nu_9 = 0, \nu_{10} = 0, \nu_{19} = 0, \nu_{28} = 0, \nu_{29} = 0, \nu_{31} = 0, \nu_{44} = 0, \nu_{49} = 0, \nu_{51} = 0, \nu_{52} = 0, \nu_{53} = 0, \nu_{57} = 0$

$$C_4 : \ \nu_6 \oplus K_4 = 0, \tag{6.4}$$

$$C_5 : \ \nu_7 \oplus \nu_{20} \oplus \nu_{23} \oplus \nu_{32} \oplus \nu_{45} \oplus K_5 = 0, \tag{6.5}$$

$$K_4 = k_7 \oplus k_8 \oplus k_{10} \oplus k_{16} \oplus k_{37} \oplus k_{49} \oplus k_{62} \oplus 1$$

and K_5 is a polynomial expression of degree 15 with 2365 monomials in 57 key variables.

The five **Type 2** relations $C_1, C_2, \ldots, C_5$ obtained in Equations (6.1)-(6.5) are crucial to the Key recovery attack. First note that due to the several **Type 1** relations that assign 27 of the IV bits to 0 or 1, the effective IV space is reduced to $\{0,1\}^{37}$. We will partition this space into 2^5 disjoint sets $T_i,\ 0 \leq i < 32$ as follows. Let $\{\nu_2, \nu_6, \nu_7, \nu_{13}, \nu_{15}\}$ be the set of dynamic cube variables. Let $K_1, K_2, \ldots, K_5$ be the unknown key expressions as described above and write $\mathbf{U} = [K_1, K_2, K_3, K_4, K_5]$. Then, for each $\mathbf{U} \in \{0,1\}^5$ the set $T_{\mathbf{U}}$ can be generated as follows:

1. Define the Set

$$\begin{aligned} T_{\mathbf{U}} \leftarrow \{V \in \{0,1\}^{64} \mid \nu_{58} = 1, \nu_0 = 0, \nu_1 = 0, \nu_3 = 0, \nu_4 = 0, \nu_5 = 0, \nu_{21} = 0, \\ \nu_{25} = 0, \nu_{26} = 0, \nu_{27} = 0, \nu_{43} = 0, \nu_{46} = 0, \nu_{47} = 0, \\ \nu_{48} = 0, \nu_8 = 0, \nu_9 = 0, \nu_{10} = 0, \nu_{19} = 0, \nu_{28} = 0, \\ \nu_{29} = 0, \nu_{31} = 0, \nu_{44} = 0, \nu_{49} = 0, \nu_{51} = 0, \nu_{52} = 0, \\ \nu_{53} = 0, \nu_{57} = 0\} \end{aligned}$$

2. For all $V \in T_{\mathbf{U}}$, adjust $\nu_2, \nu_6, \nu_7, \nu_{13}, \nu_{15}$ according to $\mathbf{U}$:

$$\begin{aligned} \nu_{15} &\leftarrow K_1, \ \nu_{13} \leftarrow \nu_{23} \oplus \nu_{38} \oplus \nu_{51} \oplus \nu_{62} \oplus K_2, \\ \nu_2 &\leftarrow \nu_{18} \oplus \nu_{31} \oplus \nu_{40} \oplus \nu_{41} \oplus \nu_{53} \oplus \nu_{56} \oplus K_3 \\ \nu_6 &\leftarrow K_4, \ \nu_7 \leftarrow \nu_{20} \oplus \nu_{23} \oplus \nu_{32} \oplus \nu_{45} \oplus K_5 \end{aligned}$$

The attacker observes that if the conditions C_1 to C_5 are all satisfied then the distributions of $z_{97} \oplus z'_{97}$ and $z_{104} \oplus z'_{104}$ exhibit non-random behavior. More specifically, it was experimentally observed that

$$\mathsf{Pr}\left[z_{97} \oplus z'_{97} = 0 \mid C_i \text{ is satisfied } \forall i \in [1,5]\right] = \frac{1}{2} + \epsilon_1, \tag{6.6}$$

$$\mathsf{Pr}\left[z_{104} \oplus z'_{104} = 0 \mid C_i \text{ is satisfied } \forall i \in [1,5]\right] = \frac{1}{2} + \epsilon_2, \tag{6.7}$$

where ϵ_1, ϵ_2 are some positive biases. Note that these biases were observed experimentally and no theoretical proof was provided for them. In this chapter, we shall attempt to provide a theoretical framework to prove the bias at round 97.

To mount the attack, the attacker tries to compute the distribution of $z_{97} \oplus z'_{97}$ and $z_{104} \oplus z'_{104}$ in each of the 32 sets $T_{\mathbf{U}}$. Observe that all the conditions $C_1, C_2, \ldots, C_5$ are satisfied in only one of these sets $T_{\mathbf{U_0}}$ where $\mathbf{U_0}$ is the correct value of $\mathbf{U}$. The attacker will therefore be able to observe the bias in the set $T_{\mathbf{U_0}}$, and by standard randomness assumptions, fail to observe any bias in the other sets, thereby determining the values of the five expressions $K_1, K_2, \ldots, K_5$. As it turns out, it the attacker may observe bias in three other sets $T_{\mathbf{U}'}$, where the values of $\mathbf{U}'$ are different from the correct $\mathbf{U_0}$. In fact the conditions C_2, C_3 need not be satisfied and thus the bias will be observed in all the other 3 sets where C_1, C_4, C_5 are satisfied but C_2, C_3 are not, and we shall provide a framework to prove this.

6.3 The Differential Engine $\Delta\mathsf{Grain}_{\mathsf{KSA}}$

In order to prove the biases observed in the distribution of $z_{97} \oplus z'_{97}$ and $z_{104} \oplus z'_{104}$ we will define a tool $\Delta\mathsf{Grain}_{\mathsf{KSA}}$ that will keep track of the differential trails of any cipher in the Grain family during the Key Scheduling process. The tool is a modification of the engine D-Grain that appeared in Algorithm 5.1. Note that while D-Grain computed the differential trails during the PRGA, our engine will do so during the KSA.

6.3.1 Generalized Grain cipher

To begin, let us rewrite the definition of the generalized Grain stream cipher that was introduced in Section 5.2.1. The cipher covers the descriptions of Grain v1, Grain-128 and Grain-128a. We already know that any cipher in the Grain family consists of an n-bit LFSR and an n-bit NFSR (see Figure 4.1). The update function of the LFSR is given by the equation

$$y_{t+n} = f(Y_t) = y_t \oplus y_{t+f_1} \oplus y_{t+f_2} \oplus \cdots \oplus y_{t+f_a},$$

where $Y_t = [y_t, y_{t+1}, \ldots, y_{t+n-1}]$ is an n-bit vector that denotes the LFSR state at the t^{th} clock interval and f is a linear function on the LFSR state bits obtained from a primitive polynomial in $GF(2)$ of degree n. The NFSR state is updated as

$$\begin{aligned} x_{t+n} &= y_t \oplus g(X_t) = y_t \oplus g(x_t, x_{t+g_1}, x_{t+g_2}, \ldots, x_{t+g_b}) \\ &= y_t \oplus x_t \oplus x_{t+g_1} \oplus \cdots \oplus x_{t+g_{b_0}} \oplus g'(x_{t+g_{b_0+1}}, x_{t+g_{b_0+2}}, \ldots, x_{t+g_b}) \end{aligned}$$

Here, $X_t = [x_t, x_{t+1}, \ldots, x_{t+n-1}]$ is an n-bit vector that denotes the NFSR state at the t^{th} clock interval and g is a non-linear function of the NFSR state bits in which the NFSR locations $0, g_1, g_2, \ldots, g_{b_0}$ only contribute linearly. The output key-stream is produced by combining the LFSR and NFSR bits as

$$\begin{aligned} z_t = x_{t+l_1} \oplus x_{t+l_2} \oplus \cdots \oplus x_{t+l_c} \oplus y_{t+i_1} \oplus y_{t+i_2} \oplus \cdots \oplus y_{t+i_d} \oplus \\ h(y_{t+h_1}, y_{t+h_2}, \ldots, y_{t+h_e}, x_{t+j_1}, x_{t+j_2}, \ldots, x_{t+j_w}). \end{aligned}$$

Here h is another non-linear combining Boolean function. So it is clear that Grain v1, Grain-128 and Grain-128a are particular instances of the generalized Grain cipher.

6.3.2 $\Delta\mathsf{Grain}_{\mathsf{KSA}}$

As defined earlier, let $S_0 = [X_0||Y_0] \in \{0,1\}^{2n}$ be the initial state of the generalized Grain KSA and $S_0^{\phi} = [X_0^{\phi}||Y_0^{\phi}]$ be the initial state which differs from S_0 in some LFSR

location $\phi \in [0, m-1]$, where m is the length of the IV. Note that, in the particular case of Grain v1, where we introduce the difference in the 37^{th} IV bit, the notation X_0', Y_0' actually implies X_0^{37}, Y_0^{37} in this context.

The task is to ascertain how the corresponding internal states in the t^{th} round S_t and S_t^{ϕ} will differ from each other, for some integer $t > 0$. We present the following algorithm which we will call $\Delta\mathsf{Grain}_{\mathsf{KSA}}$ that takes as input the difference location $\phi \in [0, m-1]$ and the value r of the number of rounds, and returns the following: **(i)** a set of r integer arrays χ_t, for $0 \leq t < r$, each of length $c + d$, **(ii)** a set of r integer arrays Υ_t, for $0 \leq t < r$, each of length $e + w$ and **(iii)** an integer array ΔZ of length r.

Note that as already defined in the description of generalized Grain, d, c are the number of LFSR, NFSR bits which are linearly added to the output function h. And e, w are the number of LFSR, NFSR bits that are input to the function h.

Now consider the corresponding generalized differential engine Δ_{ϕ}-$\mathsf{Grain}_{\mathsf{KSA}}$ with an n-cell LFSR ΔL and an n-cell NFSR ΔN. All the elements of ΔL and ΔN are integers. We will denote the t^{th} round state of ΔL as $\Delta L_t = [u_t, u_{t+1}, \ldots, u_{t+n-1}]$ and that of ΔN as $\Delta N_t = [v_t, v_{t+1}, \ldots, v_{t+n-1}]$. Initially all the elements of $\Delta N, \Delta L$ are set to 0, with the only exception that – The cell numbered ϕ of ΔL is set to 1.

The initial states $\Delta N_0, \Delta L_0$ are indicative of the difference between S_0 and S_0^{ϕ} and we will show that the t^{th} states $\Delta N_t, \Delta L_t$ are indicative of the difference between S_t and S_t^{ϕ}. Define the function $\mathsf{lin} : \cup_{i=1}^{\infty} \mathbb{Z}_+^i \rightarrow \{0, 1, 2\}$ (where $\mathbb{Z}_+$ is the set of non negative integers)

$$\mathsf{lin}(q_1, q_2, \ldots, q_i) \quad = \quad \begin{cases} q_1 + q_2 + \cdots + q_i \bmod 2 & \text{if } \max(q_1, q_2, \ldots, q_i) \leq 1, \\ 2, & \text{otherwise.} \end{cases}$$

Define the intermediate variables ℓ_t, r_t, Ω_t as follows:

$$\ell_t = \mathsf{lin}\ (u_t, u_{t+f_1}, \ldots, u_{t+f_a}),\ r_t = \mathsf{lin}\ (u_t, v_t, v_{t+g_1}, \cdots, v_{t+g_{b_0}})$$

$$\Omega_t = 2 \cdot \mathsf{OR}(v_{t+g_{b_0+1}}, v_{t+g_{b_0+2}}, \ldots, v_{t+g_b}).$$

Here OR is a map from $\cup_{i=1}^{\infty} \mathbb{Z}_+^i \rightarrow \{0, 1\}$ which roughly represents the logical 'or' operation and is defined as

$$\mathsf{OR}(q_0, q_1, \ldots, q_i) \quad = \quad \begin{cases} 0, & \text{if } q_0 = q_1 = q_2 = \cdots = q_i = 0, \\ 1, & \text{otherwise.} \end{cases}$$

Let $\chi_t = [v_{t+l_1}, v_{t+l_2}, \ldots, v_{t+l_c}, u_{t+i_1}, u_{t+i_2}, \ldots, u_{t+i_d}]$, and also define the vector $\Upsilon_t = [u_{t+h_1}, u_{t+h_2}, \ldots, u_{t+h_e}, v_{t+j_1}, v_{t+j_2}, \ldots, v_{t+j_w}]$. Note that $\chi_t(\Upsilon_t)$ is the set of cells in Δ_ϕ-$\mathsf{Grain_{KSA}}$ which corresponds to the bits which are linearly added to the output function h (input to h) in the t^{th} KSA stage of the actual cipher. The t^{th} key-stream element π_t produced by this engine is given as

$$\pi_t = \mathsf{lin}\left(\mathsf{lin}(\chi_t), 2 \cdot \mathsf{OR}(\Upsilon_t)\right)$$

Here $\mathbf{0}$ denotes the all zero vector. Now ΔL updates itself as $u_{t+n} = \mathsf{lin}(\ell_t, \pi_t)$. And similarly, ΔN updates itself as $v_{t+n} = \mathsf{lin}(r_t, \Omega_t, \pi_t)$. We will now explain the rationale behind choosing the internal variables and then explain clearly the working of the engine:

1. The Keystream element π_t: We will begin with the working hypothesis that if any element in the differential engine is :

   $\rightarrow 0$**,** the difference of the corresponding elements in S_t and S_t^ϕ is always 0.

   $\rightarrow 1$**,** the difference of the corresponding elements in S_t and S_t^ϕ is always 1.

   $\rightarrow 2$**,** the difference of the corresponding elements in S_t and S_t^ϕ is probabilistically either 0 or 1 and the exact value would depend on the exact value of the initial vector S_0 and actual update functions.

   For example if some element u_{t+n} is 0 we can assume that the corresponding LFSR bits y_{t+n} and y_{t+n}^ϕ are always equal, if π_t is 1 for some t, then we can assume that difference of the keystream bits z_t and z_t^ϕ is always unequal etc. We will show that this hypotheses is correct as we go along trying to explain the rationale behind the various elements of the engine.

   The function $\mathsf{lin}(\boldsymbol{\Delta})$ computes the modulo 2 sum of the elements of the vector $\boldsymbol{\Delta}$ only if all its elements are 0 or 1, otherwise it returns 2. This captures the notion of difference propagation rules over ordinary GF(2) addition. Let $\mathbf{x}$ and $\mathbf{x}^\phi$ be vectors in the original cipher initializations S_0 and S_0^ϕ respectively, whose contents need to be summed for some intermediate cipher operation. Let $\delta = \mathbf{x} \oplus \mathbf{x}^\phi$, then the difference of sums of the bits of $\mathbf{x}$ and $\mathbf{x}^\phi$ is equal to the sum of the contents of δ. Now if the elements of δ are always 0 or 1 (this corresponds to all elements of $\boldsymbol{\Delta}$ being either 0 or 1 in the differential engine), it implies that the corresponding elements of $\mathbf{x}$ and $\mathbf{x}^\phi$ are respectively always equal or different. Then, the difference of sums of the bits of $\mathbf{x}$ and $\mathbf{x}^\phi$ will either be always 0 or 1 and is given by the sum of elements of δ. In such an event, $\mathsf{lin}(\boldsymbol{\Delta})$ computes the modulo 2 sum of the elements of $\boldsymbol{\Delta}$ which is either 0 or 1. If, however, some corresponding elements $\mathbf{x}$ and $\mathbf{x}^\phi$ are

only probabilistically equal (this corresponds to some elements of $\mathbf{\Delta}$ being equal to 2), then the difference between the sums of their contents is also probabilistically 0 or 1. In such an event, $\mathsf{lin}(\mathbf{\Delta})$ returns 2.

The function $2 \cdot \mathsf{OR}(\mathbf{\Delta})$ returns 0 only if all elements of the vector $\mathbf{\Delta}$ is 0 and returns 2 otherwise. This captures the notion of difference propagation rules over non-linear Boolean functions. Again, let $\mathbf{x}$ and $\mathbf{x}^\phi$ be vectors in the original cipher initializations S_0 and S_0^ϕ respectively, which are fed to some non-linear function F during some intermediate cipher operation. As above, let $\delta = \mathbf{x} \oplus \mathbf{x}^\phi$. Then difference between $F(\mathbf{x})$ and $F(\mathbf{x}^\phi)$ is deterministically 0 only if all elements of δ are also deterministically 0 (this corresponds to all elements of $\mathbf{\Delta}$ being 0). If even one element of δ is not deterministically 0 then the difference between $F(\mathbf{x})$ and $F(\mathbf{x}^\phi)$ becomes probabilistic and depends on the nature of the Boolean Function $F(\mathbf{x}) \oplus F(\mathbf{x}^\phi)$. In such an event, $2 \cdot \mathsf{OR}(\mathbf{\Delta})$ returns 2.

Now observe the equation defining π_t. Note that χ_t consists of tap locations that add linearly to the output function and Υ_t consists of the locations that feed the non-linear h function in the original generalized Grain cipher. Thus the $\mathsf{lin}()$ of $\mathsf{lin}(\chi_t)$ and $2 \cdot \mathsf{OR}(\Upsilon_t)$ will effectively capture the difference between actual t^{th} round keystream bits z_t and z_t^ϕ in the two initializations of the generalized cipher.

2. Update rule of ΔL: In the original cipher, the update to the LFSR is the GF(2) sum of 2 parts: the keystream bit z_t and the linear update function f over the LFSR bits $y_t, y_{t+f_1}, y_{t+f_2}, \ldots, y_{t+f_a}$. The function $\ell_t = \mathsf{lin}(u_t, u_{t+f_1}, \ldots, u_{t+f_a})$ captures the difference propagation over the linear sum. So the definition of u_{t+n} which is $\mathsf{lin}(\ell_t, \pi_t)$ captures the difference between y_{t+n} and y_{t+n}^ϕ.

3. Update rule of ΔN: In the original cipher, the update to the NFSR is the GF(2) sum of 4 parts: the keystream bit z_t, the LFSR bit y_t, the linear function over the NFSR bits $x_t, x_{t+g_1}, \ldots, x_{t+g_{b_0}}$ and the non-linear update function g' over the bits $x_{t+g_{b_0+1}}, \ldots, x_{t+g_b}$. The function $r_t = \mathsf{lin}\ (u_t, v_t, v_{t+g_1}\ , \cdots, v_{t+g_{b_0}})$ captures the difference propagation over the linear parts, and $\Omega_t = 2 \cdot \mathsf{OR}(v_{t+g_{b_0+1}}, v_{t+g_{b_0+2}}, \ldots, v_{t+g_b})$ captures the difference over the non-linear function g'. And thus the definition of v_{t+n} which is $\mathsf{lin}(r_t, \Omega_t, \pi_t)$ captures the difference between x_{t+n} and x_{t+n}^ϕ.

4. An exception to the rule: Our definition of π_t some times fails to capture the exact difference between z_t and z_t^{ϕ}. We will demonstrate this with an example: We go back to our original system in Grain v1 where the differential is introduced via the 37^{th} IV bit and therefore we run the engine Δ_{37}-Grain$_{\mathsf{KSA}}$. At round 30 the values of χ_t and Υ_t are as follows:

$$t = 30: \quad \chi_t = \mathbf{0}, \quad \Upsilon_t = [u_{t+3} = 0, u_{t+25} = 0, u_{t+46} = 0, u_{t+64} = 1, v_{t+63} = 0]$$

Here $\mathbf{0}$ is the all zero vector. This implies that if we introduce an IV differential at location 37 then at KSA round 30 all state bits in S_{30} and S_{30}^{37} involved in the computation of their respective keystream bits are equal except the bits y_{t+64} and y_{t+64}^{37}, which are deterministically unequal, i.e., $y_{t+64} = 1 \oplus y_{t+64}^{37}$ always holds. Then, it follows that

$$\begin{aligned} z_{30} \oplus z_{30}^{37} &= h(y_{33}, y_{55}, y_{76}, y_{94}, x_{93}) \oplus h(y_{33}, y_{55}, y_{76}, 1 \oplus y_{94}, x_{93}) \\ &= y_{33}y_{76} \oplus y_{33} \oplus y_{76}x_{93} \oplus y_{76} \oplus x_{93} = 1. \end{aligned}$$

The above follows because y_{76} is initialized to 1 as it is a part of the `0x ffff` padding that is used in Grain v1. Thus, z_{30} and z_{30}^{37} are deterministically unequal. But according to the definition of π_t, the value of π_{30} would be computed as 2. To prevent a situation like this one must always check if for some t, the values of χ_t and Υ_t throw up an exception. If it does we must assign the value 1 to π_t. Thus the definition of π_t can be rewritten thus:

$$\pi_t = \begin{cases} 1 & \text{if } \chi_t, \Upsilon_t \text{ throws up an exception} \\ \mathsf{lin}\left(\mathsf{lin}(\chi_t), 2 \cdot \mathsf{OR}(\Upsilon_t)\right) & \text{otherwise.} \end{cases}$$

We present an algorithmic description of Δ_{ϕ}-Grain$_{\mathsf{KSA}}$ in Algorithm 6.1.

6.4 Proving the biases

We will now provide a theoretical frame work to prove the biases reported in Equations (6.6), (6.7) using the differential engine Δ_{ϕ}-Grain$_{\mathsf{KSA}}$ that was described in the previous Section. Note that the probability values we shall work out are computed over the randomness due to the Key bits and the those IV bits not assigned by the **Type 1, 2** relations in Section 6.2. However, these results also hold, even if the Key is fixed, and the randomness comes only from the IV bits. Before we do that let us look at the following Lemma that we will use. As the lemma is quite straightforward, we state it here without proof.

Input: ϕ: An LFSR location $\in [0, m-1]$, an integer $r(>0)$;

Output: An integer array ΔZ of r elements;

Output: Two integer arrays χ_t, Υ_t for $0 \leq t < r$;

$[u_0, u_1, \ldots, u_{n-1}] \leftarrow \mathbf{0}, [v_0, v_1, \ldots, v_{n-1}] \leftarrow \mathbf{0}$;

$t \leftarrow 0$;

$u_\phi = 1$;

while $t < r$ **do**

$\Upsilon_t \leftarrow [u_{t+h_1}, u_{t+h_2}, \ldots, u_{t+h_e}, v_{t+j_1}, v_{t+j_2}, \ldots, v_{t+j_w}]$;

$\chi_t \leftarrow [v_{t+l_1}, v_{t+l_2}, \ldots, v_{t+l_c}, u_{t+i_1}, u_{t+i_2}, \ldots, u_{t+i_d}]$;

$\ell_t \leftarrow \mathsf{lin}(u_t, u_{t+f_1}, u_{t+f_2}, \ldots, u_{t+f_a})$;

$r_t \leftarrow \mathsf{lin}(u_t, v_t, v_{t+g_1}, \cdots, v_{t+g_{b_0}})$;

$\Omega_t \leftarrow 2 \cdot \mathsf{OR}(v_{t+g_{b_0+1}}, v_{t+g_{b_0+2}}, \ldots, v_{t+g_b})$;

if χ_t, Υ_t *throws up an* exception **then**

$\pi_t \leftarrow 1$

end

else

$\pi_t \leftarrow \mathsf{lin}\,(\mathsf{lin}(\chi_t), 2 \cdot \mathsf{OR}(\Upsilon_t))$

end

$u_{t+n} \leftarrow \mathsf{lin}(\pi_t, \ell_t)$;

$v_{t+n} \leftarrow \mathsf{lin}(\pi_t, r_t, \Omega_t)$;

1 `/*Any modification goes here */`;

$t = t + 1$;

end

$\Delta Z = [\Delta z_0, \Delta z_1, \ldots, \Delta z_{r-1}]$;

Return $[\chi_0, \chi_1, \ldots, \chi_{r-1}]$, $[\Upsilon_0, \Upsilon_1, \ldots, \Upsilon_{r-1}]$, ΔZ

Algorithm 6.1: Δ_ϕ-$\mathsf{Grain}_{\mathsf{KSA}}$

Lemma 6.1. *Let F be an i-variable Boolean function, with $wt(F) = \mathsf{w}$. If the vector X is chosen uniformly from $\{0,1\}^i$ then $\mathsf{Pr}[F(X) = 0] = 1 - \frac{\mathsf{w}}{2^i}$.*

6.4.1 Δ_ϕ-$\mathsf{Grain}_{\mathsf{KSA}}$ with overrides

The system Δ_ϕ-$\mathsf{Grain}_{\mathsf{KSA}}$ works fine to track differential trails produced due to difference introduced in the ϕ^{th} IV bit. But notice that, Knellwolf's attack imposes several algebraic conditions among the Secret Key and IV bits in order to prevent the propagation of any difference to the NFSR. So, in order to replicate the difference propagation in Knellwolf's system by using the engine Δ_ϕ-$\mathsf{Grain}_{\mathsf{KSA}}$ certain modifications need to be made to it.

Since Knellwolf's system introduces difference at the 37^{th} IV bit, we run Δ_{37}-Grain$_{\mathsf{KSA}}$. Thereafter the propagation of the differential is stopped at $t = 12, 34, 40$. Hence at these rounds u_{t+n}, v_{t+n} need to be manually assigned to 0. This corresponds to inserting the following code snippet at line 1 of Algorithm 6.1.

$$\textbf{if} \;\; t \;\in\; \{12, 34, 40\} : u_{t+n} \leftarrow 0, \;\; v_{t+n} \leftarrow 0$$

Thereafter we look at the output produced by such a system at KSA round 97. The values of χ_{97}, Υ_{97} are as follows:

$$\chi_{97} : [v_{98} = 0, v_{99} = 0, v_{101} = 0, v_{107} = 0, v_{128} = 2, v_{140} = 0, v_{153} = 2]$$

$$\Upsilon_{97} : [u_{100} = 0, u_{122} = 1, u_{143} = 2, u_{161} = 2, v_{160} = 2]$$

This implies that of all the bits S_{97}, S'_{97} involved in the computation of z_{97} and z'_{97} respectively, the relations between only i) x_{128}, x'_{128} ii) x_{153}, x'_{153} iii) y_{143}, y'_{143} iv) y_{161}, y'_{161} v) x_{160}, x'_{160} is probabilistic. Therefore we have

$$\begin{aligned} z_{97} \oplus z'_{97} = {} & [x_{128} \oplus x'_{128}] \oplus [x_{153} \oplus x'_{153}] \oplus \\ & [h(y_{100}, y_{122}, y_{143}, y_{161}, x_{160}) \oplus h(y_{100}, 1 \oplus y_{122}, y'_{143}, y'_{161}, x'_{160})] \end{aligned} \tag{6.8}$$

We begin with the assumption that the random variables $x_{128} \oplus x'_{128}$, $x_{153} \oplus x'_{153}$, $y_{143} \oplus y'_{143}$, $y_{161} \oplus y'_{161}$ and $x_{160} \oplus x'_{160}$ are statistically mutually independent of one another. It is difficult to prove this assumption theoretically but extensive computer simulations have shown that one can make this assumption. We must therefore attempt to find the distributions of these variables, to prove the bias.

A. $x_{128} \oplus x'_{128}$ **:** To find this distribution we need to look at the state of our modified Δ_{37}-Grain$_{\mathsf{KSA}}$ at $t = 128 - 80 = 48$. At this round the vectors χ_t, Υ_t are as follows:

$$\chi_{48} : \mathbf{0}, \quad \Upsilon_{48} : [u_{51} = 0, u_{73} = 0, u_{94} = 1, u_{112} = 1, v_{111} = 0]$$

Among, the other state bits used in the computation of v_{128} only $v_{110} = 1$ and the rest are 0.Thus we have

$$\begin{aligned} x_{128} \oplus x'_{128} &= [g(X_{48}) \oplus y_{48} \oplus z_{48}] \oplus [g(X'_{48}) \oplus y'_{48} \oplus z'_{48}] \\ &= [g(x_{48}, x_{57}, \ldots, x_{110}, x_{111}) \oplus g(x_{48}, x_{57}, \ldots, 1 \oplus x_{110}, x_{111})] \oplus \\ &\quad [h(y_{51}, y_{73}, y_{94}, y_{112}, x_{111}) \oplus h(y_{51}, y_{73}, 1 \oplus y_{94}, 1 \oplus y_{112}, x_{111})] \\ &= x_{111} \oplus y_{94}x_{111} \oplus y_{94} \oplus y_{112}x_{111} \oplus y_{112} \end{aligned}$$

The above equations follow because $y_{73} = 1$ as required by the padding rule of Grain v1, and $y_{51} = 0$ as this is one of the **Type 1** conditions imposed on the IV bits. Assuming that the variables y_{94}, x_{111}, y_{112} are uniformly and independently distributed, and since $x_{111} \oplus y_{94}x_{111} \oplus y_{94} \oplus y_{112}x_{111} \oplus y_{112}$ is a Boolean Function of weight 6 we can use Lemma 6.1 to say:

$$\mathsf{Pr}[x_{128} \oplus x'_{128} = 0] \;=\; 1 - \frac{6}{8} = \frac{1}{4}$$

B. $x_{153} \oplus x'_{153}$ **:** To find this distribution we need to look at the state of our modified Δ_{37}-$\mathsf{Grain}_{\mathsf{KSA}}$ at $t = 153 - 80 = 73$. At this round, it turns out that $\chi_t = \Upsilon_t = \mathbf{0}$. Among the other elements involved in the computation of v_{153} only $v_{110} = v_{135} = 1$ and $v_{133} = 2$ and the rest are zero. Since $v_{133} = 2$, the difference between x_{133} and x'_{133} is still probabilistic. We would need to compute the distribution of $x_{133} \oplus x'_{133}$ before we can compute the distribution of $x_{153} \oplus x'_{153}$.

To find this distribution we look at Δ_{37}-$\mathsf{Grain}_{\mathsf{KSA}}$ at $t = 133 - 80 = 53$. At this round among all the elements involved in the computation of v_{153} only $u_{117} = 1$ and the rest are 0. So we have,

$$\begin{aligned} x_{133} \oplus x'_{133} &= [\; g(X_{53}) \oplus y_{53} \oplus z_{53} \;] \oplus [\; g(X'_{53}) \oplus y'_{53} \oplus z'_{53} \;] \\ &= h(y_{56}, y_{78}, y_{99}, y_{117}, x_{116}) \oplus h(y_{56}, y_{78}, y_{99}, 1 \oplus y_{117}, x_{116}) \\ &= y_{56}y_{99} \oplus y_{56} \oplus y_{99}x_{116} \oplus y_{99} \oplus x_{116} \end{aligned}$$

Again, assuming independent and uniform distribution of the inputs, and since $y_{56}y_{99} \oplus y_{56} \oplus y_{99}x_{116} \oplus y_{99} \oplus x_{116}$ is Boolean Function of weight 6, we have

$$\mathsf{Pr}[x_{133} \oplus x'_{133} = 0] \;=\; 1 - \frac{6}{8} = \frac{1}{4}$$

Now going back to the original problem, we have

$$\begin{aligned} &x_{153} \oplus x'_{153} = [\; g(X_{73}) \oplus y_{73} \oplus z_{73} \;] \oplus [\; g(X'_{73}) \oplus y'_{73} \oplus z'_{73} \;] \\ &= g(\ldots, x_{110}, \ldots, x_{133}, x_{135}, \ldots) \;\oplus g(\ldots, 1 \oplus x_{110}, \ldots, x'_{133}, 1 \oplus x_{135}, \ldots) \end{aligned}$$

Define

$$G_1 = g(\ldots, x_{110}, \ldots, x_{133}, x_{135}, \ldots) \oplus g(\ldots, 1 \oplus x_{110}, \ldots, x_{133}, 1 \oplus x_{135}, \ldots)$$

$$G_2 = g(\ldots, x_{110}, \ldots, x_{133}, x_{135}, \ldots) \oplus g(\ldots, 1 \oplus x_{110}, \ldots, 1 \oplus x_{133}, 1 \oplus x_{135}, \ldots)$$

We have $x_{153} \oplus x'_{153}$ equal to G_1 if $x_{133} \oplus x'_{133} = 0$ and equal to G_2 otherwise. Since, G_1 is a Boolean Function of weight 3456 and weight of G_2 is 3840, under

standard assumptions of independence we have

$$\begin{aligned}\mathsf{Pr}[x_{153} \oplus x'_{153} = 0] &= \sum_{i=0}^{1} \mathsf{Pr}[x_{133} \oplus x'_{133} = i]\mathsf{Pr}[G_1 = i] \\ &= \frac{1}{4}\left[1 - \frac{3456}{2^{13}}\right] + \frac{3}{4}\left[1 - \frac{3840}{2^{13}}\right] = \frac{139}{256}\end{aligned}$$

C. $y_{143} \oplus y'_{143}$ **:** As before we look at the output of Δ_{37}-$\mathsf{Grain}_{\mathsf{KSA}}$ at $t = 143 - 80 = 63$. At this round we have $\chi_{63} = \mathbf{0}$ and

$$\Upsilon_{63} : [u_{66} = 0, u_{88} = 0, u_{109} = 0, u_{127} = 2, v_{126} = 2]$$

All other elements involved in the computation of u_{143} are zero. We therefore need to compute the distributions of $y_{127} \oplus y'_{127}$ and $x_{126} \oplus x'_{126}$.

To compute the distribution of $y_{127} \oplus y'_{127}$ we look at $t = 47$. All elements involved in the computation of u_{127} is 0 except $v_{110} = 1$. So we have

$$\begin{aligned}y_{127} \oplus y'_{127} &= [\ f(Y_{47}) \oplus z_{47}\ ] \oplus [\ f(Y'_{47}) \oplus z'_{47}\ ] \\ &= h(y_{50}, y_{72}, y_{93}, y_{111}, x_{110}) \oplus h(y_{50}, y_{72}, y_{93}, y_{111}, 1 \oplus x_{110}) \\ &= y_{50}y_{93} \oplus y_{93} \oplus y_{93}y_{111} \oplus y_{111} \oplus 1\end{aligned}$$

The above expression represents a balanced Boolean Function and hence we have $\mathsf{Pr}[y_{127} \oplus y'_{127} = 0] = \frac{1}{2}$. To compute the distribution of $x_{126} \oplus x'_{126}$ we look at $t = 46$. At this round all the elements involved in the computation of v_{126} are zero except $u_{110} = 1$. So we have

$$\begin{aligned}x_{126} \oplus x'_{126} &= [\ g(X_{46}) \oplus y_{46} \oplus z_{46}\ ] \oplus [\ g(X'_{46}) \oplus y'_{46} \oplus z'_{46}\ ] \\ &= h(y_{49}, y_{71}, y_{92}, y_{110}, x_{109}) \oplus h(y_{49}, y_{71}, y_{92}, 1 \oplus y_{110}, x_{109}) \\ &= y_{92}x_{109} \oplus y_{92} \oplus x_{109}\end{aligned}$$

This is an Boolean Function of weight 3 and so we have $\mathsf{Pr}[x_{126} \oplus x'_{126} = 0] = 1 - \frac{3}{4} = \frac{1}{4}$. Now we have

$$\begin{aligned}y_{143} \oplus y'_{143} &= [\ f(Y_{63}) \oplus z_{63}\ ] \oplus [\ f(Y'_{63}) \oplus z'_{63}\ ] \\ &= h(y_{66}, y_{88}, y_{109}, y_{127}, x_{126}) \oplus h(y_{66}, y_{88}, y_{109}, y'_{127}, x'_{126}) \\ &= h(1, y_{88}, y_{109}, y_{127}, x_{126}) \oplus h(1, y_{88}, y_{109}, y'_{127}, x'_{126})\end{aligned}$$

The above follows since $y_{66} = 1$ is a part of the padding used in Grain v1. For $i, j = 0, 1$, define

$$h_{ij} = h(1, \ldots, y_{127}, x_{126}) \oplus h(1, \ldots, i \oplus y_{127}, j \oplus x_{126})$$

h_{01}, h_{11} are balanced functions and $\mathsf{Pr}[h_{10} = 0] = \frac{1}{4}$. Assuming independence, $\mathsf{Pr}[y_{143} \oplus y'_{143} = 0]$ is given by the expression

$$\sum_{i=0}^{1}\sum_{j=0}^{1} \mathsf{Pr}[y_{127} \oplus y'_{127} = i]\mathsf{Pr}[x_{126} \oplus x'_{126} = j] \cdot \mathsf{Pr}[h_{ij} = 0] = \frac{17}{32}$$

D. $y_{161} \oplus y'_{161}$ **and** $x_{160} \oplus x'_{160}$ **:** To compute the distribution of $y_{161} \oplus y'_{161}$ we need to look at round $t = 81$. At this round both χ_t and Υ_t have many elements equal to 2 and hence at this point we have to delve into several lower KSA rounds, and frankly this exercise becomes a little tedious. Due to lack of space we do not include extensive analysis of these two distributions and simply state the results.

$$\mathsf{Pr}[y_{161} \oplus y'_{161} = 0] \;=\; 0.5, \quad \mathsf{Pr}[x_{160} \oplus x'_{160} = 0] \;=\; 0.4977$$

E. $h(y_{100}, y_{122}, y_{143}, y_{161}, x_{160}) \oplus h(y_{100}, 1 \oplus y_{122}, y'_{143}, y'_{161}, x'_{160})$ **:** For the sake of conciseness, let this expression be denoted by the symbol H and again for $i, j, k = 0, 1$, let us define the functions

$$H_{ijk} = h(y_{100}, y_{122}, y_{143}, y_{161}, x_{160}) \oplus h(y_{100}, 1 \oplus y_{122}, y_{143} \oplus i, y_{161} \oplus j, x_{160} \oplus k)$$

It turns out that all H_{ijk} are balanced except for H_{000} for which $\mathsf{Pr}[H_{000} = 0] = \frac{1}{4}$. Assuming independence, $\mathsf{Pr}[H = 0]$ is given by the expression:

$$\sum_{i,j,k=0}^{1} \mathsf{Pr}[y_{143} \oplus y'_{143} = i]\ \mathsf{Pr}[y_{161} \oplus y'_{161} = j]\ \mathsf{Pr}[x_{160} \oplus x'_{160} = k]\ \mathsf{Pr}[H_{ijk} = 0]$$
$$= 0.467$$

6.4.2 Computing $\mathsf{Pr}[z_{97} \oplus z'_{97} = 0]$

Now we know from Equation (6.8), that $z_{97} \oplus z'_{97}$ is the GF(2) sum of the three expressions $x_{128} \oplus x'_{128}$, $x_{153} \oplus x'_{153}$ and H whose distributions we have just computed. Thus we have

$$\begin{aligned}\mathsf{Pr}[z_{97} \oplus z'_{97} = 0] &= \sum_{i \oplus j \oplus k = 0} \mathsf{Pr}[x_{128} \oplus x'_{128} = i] \cdot \mathsf{Pr}[x_{153} \oplus x'_{153} = j] \cdot \mathsf{Pr}[H = k] \\ &= \; 0.5014\end{aligned}$$

The above bias has been verified by experiments with over 2^{20} randomly chosen Secret Keys.

6.4.3 Biases in the other Sets

In [93], it was observed that bias can be observed in 3 other sets $T_{\mathbf{U}}$ other than the one indexed by the 5 correct Key expressions $\mathbf{U}_0$. These sets are those indexed by sets three $T_{\mathbf{U}}$ where **a**) C_2 is not satisfied but C_1 is, **b**) C_1 is not satisfied but C_2 is and **c**) None of C_1 or C_2 is satisfied. This can be proven in a similar manner by performing the above analysis with Δ_{37}-$\mathsf{Grain}_{\mathsf{KSA}}$ with a different set of overrides than the ones used in the previous proof. Note that for all the cases **a-c**, it implies that the differential at KSA round $t = 34$ is not eliminated. So as before we analyze a modified Δ_{37}-$\mathsf{Grain}_{\mathsf{KSA}}$, i.e., a modified Algorithm 6.1 in which Line 1 is replaced by

$$\textbf{if} \;\; t \;\in\; \{12, 40\} : u_{t+n} \leftarrow 0, \quad v_{t+n} \leftarrow 0$$

6.5 Conclusion and Open Problems

In this chapter, we revisited Knellwolf's attacks [93, 94], on Grain v1. The attacks, which were the best published on Grain v1, in terms of the number of rounds attacked, were based on certain biases that were observed experimentally in the distribution of the keystream bits. There were however no theoretical proof of these biases. In this work, we have tried to provide a theoretical framework to prove the biases and thus prove correctness of these attacks.

One open problem in this area is, of course, to use the engine Δ_{ϕ}-$\mathsf{Grain}_{\mathsf{KSA}}$ to attack a higher number rounds of the KSA of Grain v1. Another important open problem in this domain is to prove the bias at round 104. The author of [93] observes that at round 104, a bias is observed in one of the Sets in only about 50 % of the cases. It would be a worthwhile exercise, to determine explicitly, the algebraic conditions the Secret Key bits need to satisfy for the bias to be observed.

Chapter 7

Differential Fault Analysis of MICKEY 2.0

In this chapter we describe the ideas leading to a Differential Fault Attack (DFA) on MICKEY 2.0, a stream cipher from eStream hardware profile. Using standard assumptions for fault attacks, we first show that if the adversary can induce random single bit faults in the internal state of the cipher, then by injecting around $2^{16.7}$ faults and performing $2^{32.5}$ computations on an average, it is possible to recover the entire internal state of MICKEY at the beginning of the key-stream generation phase. We further consider the scenario where the fault may affect more than one (at most three) neighboring bits and in that case we require around $2^{18.4}$ faults on an average to mount the DFA. We further show that if the attacker can solve multivariate equations (say, using SAT solvers) then the attack can be carried out using around $2^{14.7}$ faults in the single-bit fault model and $2^{16.06}$ faults for the multiple-bit scenario.

7.1 Introduction

The stream cipher MICKEY 2.0 [16] was designed by Steve Babbage and Matthew Dodd as a submission to the eStream project. The cipher has been selected as a part of eStream's final hardware portfolio. MICKEY is a synchronous, bit-oriented stream cipher designed for low hardware complexity and high speed. After a TMD tradeoff attack [80] against the initial version of MICKEY (version 1), the designers responded with a tweak to the design by increasing the state size from 160 to 200 bits and altering the values of some control bit tap locations. These changes were incorporated in MICKEY 2.0 and these are the only differences between MICKEY version 1 and MICKEY 2.0. While MICKEY 2.0 uses an 80-bit key and a variable length IV, a modified version

of the cipher, MICKEY-128 2.0 that uses a 128-bit key [17] was also proposed by the designers.

The name MICKEY is derived from "Mutual Irregular Clocking KEY-stream generator" which describes the behavior of the cipher. The state consists of two 100-bit shift registers named R and S, each of which is irregularly clocked and controlled by the other. The cipher specification underlines that each key can be used with up to 2^{40} different IVs of the same length, and that 2^{40} key-stream bits can be generated from each Key-IV pair. Very little cryptanalysis of MICKEY 2.0 is available in literature. In fact it has been noted in [47, Section 3.2] that other than the observation related to time or power analysis attacks [66] on straightforward implementations of the MICKEY family, there have been no known cryptanalytic advances on these ciphers. Although, in [130], non-smooth cryptanalysis of MICKEY 2.0 was presented. The attack, however, had time complexity more than exhaustive search. Apart from these, no other published results on MICKEY 2.0 are available. The work in this chapter presents cryptanalytic result of MICKEY 2.0 in terms of differential fault attack.

Due to the extremely complex algebraic structure of MICKEY 2.0, it is not possible to attack the cipher under the same attack model described in Section 5.3. That is to say, against MICKEY 2.0, the attacker must be equipped with certain additional powers. In the attack that we will describe, we assume that the attacker is able to do the following:

1. We assume that the adversary can re-key the cipher with the original Key-IV and restart cipher operations multiple times.

2. He has precise control over the timing of the fault injection, i.e., the faults he injects are time-synchronized.

3. He is able to flip exactly one bit lying in some random register location which he can not choose. He is however able to register he intends to injects faults in, i.e., he is able to target either the register R or the register S to inject faults.

4. Later, in Section 7.5, we will explore the situation when he can inject a fault that may affect more than one value in contiguous register locations. We present explicit results considering the events when upto three contiguous register locations may be affected in R or S.

7.2 Structure of MICKEY 2.0

MICKEY 2.0 uses an 80-bit key and a variable length IV, the length of which may be between 0 and 80 bits. The physical structure of the cipher consists of two 100 bit

registers R and S that exercise mutual control over each other's evolution. Both the registers are initialized to the all-zero state, and the three stages of register update (i) IV loading, (ii) Key Loading, and (iii) Pre-Clock are executed sequentially before the production of the first key-stream bit. Thereafter, during the PRGA (Pseudo Random bitstream Generation Algorithm), key-stream bits are produced. Let $r_0, r_1, r_2, \ldots, r_{99}$ denote the contents of the register R and $s_0, s_1, s_2, \ldots, s_{99}$ denote the contents of the register S. In order to describe the structure of the cipher and its working let us first define the following routines.

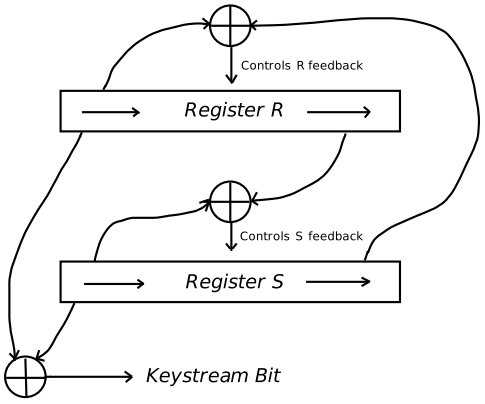


FIGURE 7.1: The variable clocking architecture of MICKEY

Clocking register R

Let $r_0, r_1, \ldots, r_{99}$ be the state of the register R before clocking, and let $r'_0, r'_1, \ldots, r'_{99}$ be the state of the register R after clocking. Define the integer array $RTAPS$ as follows

$$\begin{aligned} RTAPS = \{\ & 0, 1, 3, 4, 5, 6, 9, 12, 13, 16, 19, 20, 21, 22, 25, 28, 37, 38, 41, 42, \\ & 45, 46, 50, 52, 54, 56, 58, 60, 61, 63, 64, 65, 66, 67, 71, 72, 79, 80, \\ & 81, 82, 87, 88, 89, 90, 91, 92, 94, 95, 96, 97\} \end{aligned}$$

Now define an operation

$$\mathsf{CLOCK\_R}(R,\ \mathsf{INPUT\_BIT\_R}, \mathsf{CONTROL\_BIT\_R})$$

1. Define $\mathsf{FEEDBACK\_BIT} = r_{99} \oplus \mathsf{INPUT\_BIT\_R}$
2. For $1 \leq i \leq 99$: $r'_i = r_{i-1}$. $r'_0 = 0$.
3. For $0 \leq i \leq 99$: if $i \in RTAPS$, $r'_i = r'_i \oplus \mathsf{FEEDBACK\_BIT}$.
4. If $\mathsf{CONTROL\_BIT\_R} = 1$:

   For $0 \leq i \leq 99$: $r'_i = r'_i \oplus r_i$

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\mathsf{COMP0}_i$		0	0	0	1	1	0	0	0	1	0	1	1	1	1	0	1
$\mathsf{COMP1}_i$		1	0	1	1	0	0	1	0	1	1	1	1	0	0	1	0
$\mathsf{FB0}_i$	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	0	0
$\mathsf{FB1}_i$	1	1	1	0	1	1	1	0	0	0	0	1	1	1	0	1	0
i	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
$\mathsf{COMP0}_i$	0	0	1	0	1	0	1	0	1	0	1	0	1	1	0	1	0
$\mathsf{COMP1}_i$	1	0	0	0	1	1	0	1	0	1	1	1	0	1	1	1	1
$\mathsf{FB0}_i$	1	0	1	1	1	1	1	1	1	1	1	1	0	0	1	1	0
$\mathsf{FB1}_i$	0	1	1	0	0	0	1	0	0	1	1	0	0	1	0	1	1
i	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
$\mathsf{COMP0}_i$	0	1	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0
$\mathsf{COMP1}_i$	0	0	0	1	1	0	1	0	1	1	1	0	0	0	0	1	0
$\mathsf{FB0}_i$	0	0	0	0	0	1	1	1	0	0	1	0	0	1	0	1	0
$\mathsf{FB1}_i$	0	0	0	1	1	0	0	0	0	0	1	1	0	1	1	0	0
i	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67
$\mathsf{COMP0}_i$	0	0	0	1	0	1	0	0	1	1	1	1	0	0	1	0	1
$\mathsf{COMP1}_i$	0	0	1	0	1	1	1	0	0	0	1	1	1	1	1	1	0
$\mathsf{FB0}_i$	1	0	0	1	0	1	1	1	1	0	1	0	1	0	1	0	0
$\mathsf{FB1}_i$	0	1	0	0	0	1	0	0	1	0	0	1	0	1	1	0	1
i	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84
$\mathsf{COMP0}_i$	0	1	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1
$\mathsf{COMP1}_i$	1	0	1	1	1	0	1	1	1	1	0	0	0	1	0	0	0
$\mathsf{FB0}_i$	0	0	0	0	0	0	0	1	1	0	1	0	0	0	1	1	0
$\mathsf{FB1}_i$	0	1	0	0	1	0	1	0	0	0	1	1	1	1	0	1	1
i	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99		
$\mathsf{COMP0}_i$	1	1	0	1	0	1	0	0	0	0	0	0	1	1			
$\mathsf{COMP1}_i$	0	1	1	1	0	0	0	1	0	0	1	1	0	0			
$\mathsf{FB0}_i$	1	1	1	0	0	1	1	1	0	0	1	1	0	0	0		
$\mathsf{FB1}_i$	1	1	1	0	0	0	0	0	0	1	0	0	0	0	1		

TABLE 7.1: The sequences $\mathsf{COMP0}$, $\mathsf{COMP1}$, $\mathsf{FB0}$, $\mathsf{FB1}$

Clocking register S

Let $s_0, s_1, \ldots, s_{99}$ be the state of the register S before clocking, and let $s'_0, s'_1, \ldots, s'_{99}$ be the state of the register S after clocking. Let $\hat{s}_0, \hat{s}_1, \ldots, \hat{s}_{99}$ be intermediate variables. Define the four sequences $\mathsf{COMP0}_i$, $1 \leq i \leq 98$; $\mathsf{COMP1}_i$, $1 \leq i \leq 98$; $\mathsf{FB0}_i$, $0 \leq i \leq 99$ and $\mathsf{FB1}_i$, $0 \leq i \leq 99$ over GF(2) as in Table 7.1: Now define an operation

$$\mathsf{CLOCK\_S}(S,\ \mathsf{INPUT\_BIT\_S},\mathsf{CONTROL\_BIT\_S})$$

1. Define $\mathsf{FEEDBACK\_BIT} = s_{99} \oplus \mathsf{INPUT\_BIT\_S}$
2. For $1 \leq i \leq 98$: $\hat{s}_i = s_{i-1} \oplus \big((s_i \oplus \mathsf{COMP0}_i) \cdot (s_{i\oplus 1} \oplus \mathsf{COMP1}_i)\big)$. $\hat{s}_0 = 0$, $\hat{s}_{99} = s_{98}$.
3. If $\mathsf{CONTROL\_BIT\_S} = 0$:

   For $0 \leq i \leq 99$: $s'_i = \hat{s}_i \oplus (\ \mathsf{FB0}_i \cdot\ \mathsf{FEEDBACK\_BIT})$

Else If $\mathsf{CONTROL\_BIT\_S} = 1$:

For $0 \leq i \leq 99$: $s'_i = \hat{s}_i \oplus (\ \mathsf{FB1}_i \cdot\ \mathsf{FEEDBACK\_BIT})$

The CLOCK_KG routine

We define another operation

$$\mathsf{CLOCK\_KG}\ (R, S,\ \mathsf{MIXING}\ ,\ \mathsf{INPUT\_BIT}\ )$$

1. $\mathsf{CONTROL\_BIT\_R} = s_{34} \oplus r_{67},\ \ \mathsf{CONTROL\_BIT\_S} = s_{67} \oplus r_{33}$
2. If $\mathsf{MIXING} = 1$:

   $\mathsf{INPUT\_BIT\_R} =\ \mathsf{INPUT\_BIT}\ \oplus s_{50}$

   Else If $\mathsf{MIXING} = 0$:

   $\mathsf{INPUT\_BIT\_R} =\ \mathsf{INPUT\_BIT}$
3. $\mathsf{INPUT\_BIT\_S} =\ \mathsf{INPUT\_BIT}$
4. $\mathsf{CLOCK\_R}(R,\ \mathsf{INPUT\_BIT\_R}, \mathsf{CONTROL\_BIT\_R})$
5. $\mathsf{CLOCK\_S}(S,\ \mathsf{INPUT\_BIT\_S}, \mathsf{CONTROL\_BIT\_S})$

Working of the Cipher

We will now describe the algorithm governing the functioning of the cipher. Let $K = k_0, k_1, \ldots, k_{79}$ be the 80 bit key used by the cipher. Let $IV = iv_0, iv_1, \ldots, iv_{v-1}$ be the v-bit IV ($0 \leq v \leq 80$). Then the cipher operates in the 4 stages as described below.

STAGE 1. IV loading

Initialize both R and S to the all-zero state.

For $0 \leq i \leq v-1$: $\mathsf{CLOCK\_KG}(R, S, 1, iv_i)$

STAGE 2. Key loading

For $0 \leq i \leq 79$: $\mathsf{CLOCK\_KG}(R, S, 1, k_i)$

STAGE 3. Preclock Stage

For $0 \leq i \leq 99$: $\mathsf{CLOCK\_KG}(R, S, 1, 0)$

STAGE 4. PRGA(Pseudo-Random stream generation algorithm)

$i \leftarrow 0$

While key-stream is required

$z_i = r_0 \oplus s_0$

$\mathsf{CLOCK\_KG}(R, S, 0, 0)$

$i \leftarrow i + 1$

7.3 An alternate description of MICKEY 2.0 PRGA and a summary of results

We will now provide an alternate description of this stage of operation (PRGA) in MICKEY 2.0. Consider the binary variables a_0, a_1, a_2, a_3. Let a_0 be defined as

$$a_0 = \begin{cases} a_2, & \text{if } a_1 = 0 \\ a_3, & \text{if } a_1 = 1. \end{cases}$$

Then it is straightforward to see that a_0 can be expressed as a multivariate polynomial over GF(2), i.e.,

$$a_0 = (1 \oplus a_1) \cdot a_2 \oplus a_1 \cdot a_3.$$

The state registers R and S, during the PRGA, are updated by a call to the $\mathsf{CLOCK\_KG}$ routine, which in turn calls the $\mathsf{CLOCK\_R}$ and the $\mathsf{CLOCK\_S}$ routine. In both these routines, the state is updated via a number of If-Else constructs. As a result of this, the state update may be equivalently expressed as a series of multi-variate polynomials over GF(2).

Let $r_0, r_1, \ldots, r_{99}, s_0, s_1, \ldots, s_{99}$ denote the internal state at a certain round during the MICKEY PRGA and let $r'_0, r'_1, \ldots, r'_{99}, s'_0, s'_1, \ldots, s'_{99}$ denote the internal state at the next round. Then it is possible to write

$$r'_i = \rho_i(r_0, r_1, \ldots, r_{99}, s_0, s_1, \ldots, s_{99}),$$

$$s'_i = \beta_i(r_0, r_1, \ldots, r_{99}, s_0, s_1, \ldots, s_{99}),$$

$\forall i \in [0, 99]$, where ρ_i, β_i are polynomial functions over GF(2). The exact forms of ρ_i, β_i are described in the following tables.

TABLE 7.2: The update functions ρ, β for MICKEY 2.0

i	ρ_i	β_i
0	$r_0 \cdot r_{67} \oplus r_0 \cdot s_{34} \oplus r_{99}$	s_{99}
1	$r_0 \oplus r_1 \cdot r_{67} \oplus r_1 \cdot s_{34} \oplus r_{99}$	$s_0 \oplus s_1 \cdot s_2 \oplus s_1 \oplus s_{99}$
2	$r_1 \oplus r_2 \cdot r_{67} \oplus r_2 \cdot s_{34}$	$s_1 \oplus s_2 \cdot s_3 \oplus s_{99}$
3	$r_2 \oplus r_3 \cdot r_{67} \oplus r_3 \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_2 \oplus s_3 \cdot s_4 \oplus s_3 \oplus s_{67} \cdot s_{99} \oplus s_{99}$
4	$r_3 \oplus r_4 \cdot r_{67} \oplus r_4 \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_3 \oplus s_4 \cdot s_5 \oplus s_4 \oplus s_5 \oplus s_{67} \cdot s_{99} \oplus 1$
5	$r_4 \oplus r_5 \cdot r_{67} \oplus r_5 \cdot s_{34} \oplus r_{99}$	$s_4 \oplus s_5 \cdot s_6 \oplus s_6 \oplus s_{99}$
6	$r_5 \oplus r_6 \cdot r_{67} \oplus r_6 \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_5 \oplus s_6 \cdot s_7 \oplus s_{67} \cdot s_{99}$
7	$r_6 \oplus r_7 \cdot r_{67} \oplus r_7 \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_6 \oplus s_7 \cdot s_8 \oplus s_7 \oplus s_{67} \cdot s_{99} \oplus s_{99}$
8	$r_7 \oplus r_8 \cdot r_{67} \oplus r_8 \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_7 \oplus s_8 \cdot s_9 \oplus s_{67} \cdot s_{99} \oplus s_{99}$
9	$r_8 \oplus r_9 \cdot r_{67} \oplus r_9 \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_8 \oplus s_9 \cdot s_{10} \oplus s_9 \oplus s_{10} \oplus s_{67} \cdot s_{99} \oplus s_{99} \oplus 1$
10	$r_9 \oplus r_{10} \cdot r_{67} \oplus r_{10} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_9 \oplus s_{10} \cdot s_{11} \oplus s_{10} \oplus s_{67} \cdot s_{99} \oplus s_{99}$
11	$r_{10} \oplus r_{11} \cdot r_{67} \oplus r_{11} \cdot s_{34}$	$s_{10} \oplus s_{11} \cdot s_{12} \oplus s_{11} \oplus s_{12} \oplus s_{99} \oplus 1$
12	$r_{11} \oplus r_{12} \cdot r_{67} \oplus r_{12} \cdot s_{34} \oplus r_{99}$	$s_{11} \oplus s_{12} \cdot s_{13} \oplus s_{12} \oplus s_{13} \oplus s_{99} \oplus 1$
13	$r_{12} \oplus r_{13} \cdot r_{67} \oplus r_{13} \cdot s_{34} \oplus r_{99}$	$s_{12} \oplus s_{13} \cdot s_{14} \oplus s_{14} \oplus s_{99}$
14	$r_{13} \oplus r_{14} \cdot r_{67} \oplus r_{14} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{13} \oplus s_{14} \cdot s_{15} \oplus s_{15} \oplus s_{67} \cdot s_{99} \oplus s_{99}$
15	$r_{14} \oplus r_{15} \cdot r_{67} \oplus r_{15} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{14} \oplus s_{15} \cdot s_{16} \oplus s_{15} \oplus s_{67} \cdot s_{99}$
16	$r_{15} \oplus r_{16} \cdot r_{67} \oplus r_{16} \cdot s_{34} \oplus r_{99}$	$s_{15} \oplus s_{16} \cdot s_{17} \oplus s_{17}$
17	$r_{16} \oplus r_{17} \cdot r_{67} \oplus r_{17} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{16} \oplus s_{17} \cdot s_{18} \oplus s_{17} \oplus s_{67} \cdot s_{99} \oplus s_{99}$
18	$r_{17} \oplus r_{18} \cdot r_{67} \oplus r_{18} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{17} \oplus s_{18} \cdot s_{19} \oplus s_{67} \cdot s_{99}$
19	$r_{18} \oplus r_{19} \cdot r_{67} \oplus r_{19} \cdot s_{34} \oplus r_{99}$	$s_{18} \oplus s_{19} \cdot s_{20} \oplus s_{20} \oplus s_{99}$
20	$r_{19} \oplus r_{20} \cdot r_{67} \oplus r_{20} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{19} \oplus s_{20} \cdot s_{21} \oplus s_{67} \cdot s_{99} \oplus s_{99}$
21	$r_{20} \oplus r_{21} \cdot r_{67} \oplus r_{21} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{20} \oplus s_{21} \cdot s_{22} \oplus s_{21} \oplus s_{22} \oplus s_{67} \cdot s_{99} \oplus s_{99} \oplus 1$
22	$r_{21} \oplus r_{22} \cdot r_{67} \oplus r_{22} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{21} \oplus s_{22} \cdot s_{23} \oplus s_{22} \oplus s_{67} \cdot s_{99} \oplus s_{99}$
23	$r_{22} \oplus r_{23} \cdot r_{67} \oplus r_{23} \cdot s_{34}$	$s_{22} \oplus s_{23} \cdot s_{24} \oplus s_{24} \oplus s_{99}$
24	$r_{23} \oplus r_{24} \cdot r_{67} \oplus r_{24} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{23} \oplus s_{24} \cdot s_{25} \oplus s_{24} \oplus s_{67} \cdot s_{99} \oplus s_{99}$
25	$r_{24} \oplus r_{25} \cdot r_{67} \oplus r_{25} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{24} \oplus s_{25} \cdot s_{26} \oplus s_{26} \oplus s_{67} \cdot s_{99} \oplus s_{99}$
26	$r_{25} \oplus r_{26} \cdot r_{67} \oplus r_{26} \cdot s_{34}$	$s_{25} \oplus s_{26} \cdot s_{27} \oplus s_{26} \oplus s_{99}$
27	$r_{26} \oplus r_{27} \cdot r_{67} \oplus r_{27} \cdot s_{34}$	$s_{26} \oplus s_{27} \cdot s_{28} \oplus s_{27} \oplus s_{28} \oplus s_{99} \oplus 1$
28	$r_{27} \oplus r_{28} \cdot r_{67} \oplus r_{28} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{27} \oplus s_{28} \cdot s_{29} \oplus s_{28} \oplus s_{67} \cdot s_{99} \oplus s_{99}$
29	$r_{28} \oplus r_{29} \cdot r_{67} \oplus r_{29} \cdot s_{34}$	$s_{28} \oplus s_{29} \cdot s_{30} \oplus s_{30}$
30	$r_{29} \oplus r_{30} \cdot r_{67} \oplus r_{30} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{29} \oplus s_{30} \cdot s_{31} \oplus s_{30} \oplus s_{31} \oplus s_{67} \cdot s_{99} \oplus 1$
31	$r_{30} \oplus r_{31} \cdot r_{67} \oplus r_{31} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{30} \oplus s_{31} \cdot s_{32} \oplus s_{31} \oplus s_{67} \cdot s_{99} \oplus s_{99}$
32	$r_{31} \oplus r_{32} \cdot r_{67} \oplus r_{32} \cdot s_{34}$	$s_{31} \oplus s_{32} \cdot s_{33} \oplus s_{32} \oplus s_{33} \oplus s_{99} \oplus 1$
33	$r_{32} \oplus r_{33} \cdot r_{67} \oplus r_{33} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{32} \oplus s_{33} \cdot s_{34} \oplus s_{33} \oplus s_{67} \cdot s_{99}$
34	$r_{33} \oplus r_{34} \cdot r_{67} \oplus r_{34} \cdot s_{34}$	$s_{33} \oplus s_{34} \cdot s_{35}$
35	$r_{34} \oplus r_{35} \cdot r_{67} \oplus r_{35} \cdot s_{34}$	$s_{34} \oplus s_{35} \cdot s_{36} \oplus s_{36}$
36	$r_{35} \oplus r_{36} \cdot r_{67} \oplus r_{36} \cdot s_{34}$	$s_{35} \oplus s_{36} \cdot s_{37}$
37	$r_{36} \oplus r_{37} \cdot r_{67} \oplus r_{37} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{36} \oplus s_{37} \cdot s_{38} \oplus s_{37} \oplus s_{67} \cdot s_{99}$
38	$r_{37} \oplus r_{38} \cdot r_{67} \oplus r_{38} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{37} \oplus s_{38} \cdot s_{39} \oplus s_{38} \oplus s_{67} \cdot s_{99}$
39	$r_{38} \oplus r_{39} \cdot r_{67} \oplus r_{39} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{38} \oplus s_{39} \cdot s_{40} \oplus s_{67} \cdot s_{99} \oplus s_{99}$
40	$r_{39} \oplus r_{40} \cdot r_{67} \oplus r_{40} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{39} \oplus s_{40} \cdot s_{41} \oplus s_{40} \oplus s_{67} \cdot s_{99} \oplus s_{99}$
41	$r_{40} \oplus r_{41} \cdot r_{67} \oplus r_{41} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{40} \oplus s_{41} \cdot s_{42} \oplus s_{67} \cdot s_{99} \oplus s_{99}$
42	$r_{41} \oplus r_{42} \cdot r_{67} \oplus r_{42} \cdot s_{34} \oplus r_{99}$	$s_{41} \oplus s_{42} \cdot s_{43} \oplus s_{42}$
43	$r_{42} \oplus r_{43} \cdot r_{67} \oplus r_{43} \cdot s_{34}$	$s_{42} \oplus s_{43} \cdot s_{44} \oplus s_{43} \oplus s_{44} \oplus 1$
44	$r_{43} \oplus r_{44} \cdot r_{67} \oplus r_{44} \cdot s_{34}$	$s_{43} \oplus s_{44} \cdot s_{45} \oplus s_{44} \oplus s_{99}$

i	ρ_i	β_i
45	$r_{44} \oplus r_{45} \cdot r_{67} \oplus r_{45} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{44} \oplus s_{45} \cdot s_{46} \oplus s_{46} \oplus s_{67} \cdot s_{99}$
46	$r_{45} \oplus r_{46} \cdot r_{67} \oplus r_{46} \cdot s_{34} \oplus r_{99}$	$s_{45} \oplus s_{46} \cdot s_{47}$
47	$r_{46} \oplus r_{47} \cdot r_{67} \oplus r_{47} \cdot s_{34}$	$s_{46} \oplus s_{47} \cdot s_{48} \oplus s_{48} \oplus s_{99}$
48	$r_{47} \oplus r_{48} \cdot r_{67} \oplus r_{48} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{47} \oplus s_{48} \cdot s_{49} \oplus s_{67} \cdot s_{99}$
49	$r_{48} \oplus r_{49} \cdot r_{67} \oplus r_{49} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{48} \oplus s_{49} \cdot s_{50} \oplus s_{49} \oplus s_{50} \oplus s_{67} \cdot s_{99} \oplus s_{99} \oplus 1$
50	$r_{49} \oplus r_{50} \cdot r_{67} \oplus r_{50} \cdot s_{34} \oplus r_{99}$	$s_{49} \oplus s_{50} \cdot s_{51}$
51	$r_{50} \oplus r_{51} \cdot r_{67} \oplus r_{51} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{50} \oplus s_{51} \cdot s_{52} \oplus s_{67} \cdot s_{99} \oplus s_{99}$
52	$r_{51} \oplus r_{52} \cdot r_{67} \oplus r_{52} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{51} \oplus s_{52} \cdot s_{53} \oplus s_{67} \cdot s_{99}$
53	$r_{52} \oplus r_{53} \cdot r_{67} \oplus r_{53} \cdot s_{34}$	$s_{52} \oplus s_{53} \cdot s_{54} \oplus s_{53}$
54	$r_{53} \oplus r_{54} \cdot r_{67} \oplus r_{54} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{53} \oplus s_{54} \cdot s_{55} \oplus s_{55} \oplus s_{67} \cdot s_{99} \oplus s_{99}$
55	$r_{54} \oplus r_{55} \cdot r_{67} \oplus r_{55} \cdot s_{34}$	$s_{54} \oplus s_{55} \cdot s_{56} \oplus s_{55}$
56	$r_{55} \oplus r_{56} \cdot r_{67} \oplus r_{56} \cdot s_{34} \oplus r_{99}$	$s_{55} \oplus s_{56} \cdot s_{57} \oplus s_{56} \oplus s_{57} \oplus s_{99} \oplus 1$
57	$r_{56} \oplus r_{57} \cdot r_{67} \oplus r_{57} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{56} \oplus s_{57} \cdot s_{58} \oplus s_{57} \oplus s_{67} \cdot s_{99} \oplus s_{99}$
58	$r_{57} \oplus r_{58} \cdot r_{67} \oplus r_{58} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{57} \oplus s_{58} \cdot s_{59} \oplus s_{67} \cdot s_{99} \oplus s_{99}$
59	$r_{58} \oplus r_{59} \cdot r_{67} \oplus r_{59} \cdot s_{34}$	$s_{58} \oplus s_{59} \cdot s_{60} \oplus s_{60} \oplus s_{99}$
60	$r_{59} \oplus r_{60} \cdot r_{67} \oplus r_{60} \cdot s_{34} \oplus r_{99}$	$s_{59} \oplus s_{60} \cdot s_{61} \oplus s_{61}$
61	$r_{60} \oplus r_{61} \cdot r_{67} \oplus r_{61} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{60} \oplus s_{61} \cdot s_{62} \oplus s_{61} \oplus s_{62} \oplus s_{67} \cdot s_{99} \oplus s_{99} \oplus 1$
62	$r_{61} \oplus r_{62} \cdot r_{67} \oplus r_{62} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{61} \oplus s_{62} \cdot s_{63} \oplus s_{62} \oplus s_{63} \oplus s_{67} \cdot s_{99} \oplus 1$
63	$r_{62} \oplus r_{63} \cdot r_{67} \oplus r_{63} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{62} \oplus s_{63} \cdot s_{64} \oplus s_{63} \oplus s_{67} \cdot s_{99} \oplus s_{99}$
64	$r_{63} \oplus r_{64} \cdot r_{67} \oplus r_{64} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{63} \oplus s_{64} \cdot s_{65} \oplus s_{64} \oplus s_{67} \cdot s_{99}$
65	$r_{64} \oplus r_{65} \cdot r_{67} \oplus r_{65} \cdot s_{34} \oplus r_{99}$	$s_{64} \oplus s_{65} \cdot s_{66} \oplus s_{65} \oplus s_{66} \oplus s_{99} \oplus 1$
66	$r_{65} \oplus r_{66} \cdot r_{67} \oplus r_{66} \cdot s_{34} \oplus r_{99}$	$s_{65} \oplus s_{66} \cdot s_{67} \oplus s_{66}$
67	$r_{66} \oplus r_{67} \cdot s_{34} \oplus r_{67} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{66} \oplus s_{67} \cdot s_{68} \oplus s_{67} \cdot s_{99} \oplus s_{68}$
68	$r_{67} \cdot r_{68} \oplus r_{67} \oplus r_{68} \cdot s_{34}$	$s_{67} \oplus s_{68} \cdot s_{69} \oplus s_{68}$
69	$r_{67} \cdot r_{69} \oplus r_{68} \oplus r_{69} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{67} \cdot s_{99} \oplus s_{68} \oplus s_{69} \cdot s_{70} \oplus s_{70}$
70	$r_{67} \cdot r_{70} \oplus r_{69} \oplus r_{70} \cdot s_{34}$	$s_{69} \oplus s_{70} \cdot s_{71} \oplus s_{70} \oplus s_{71} \oplus 1$
71	$r_{67} \cdot r_{71} \oplus r_{70} \oplus r_{71} \cdot s_{34} \oplus r_{99}$	$s_{70} \oplus s_{71} \cdot s_{72} \oplus s_{71} \oplus s_{72} \oplus 1$
72	$r_{67} \cdot r_{72} \oplus r_{71} \oplus r_{72} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{67} \cdot s_{99} \oplus s_{71} \oplus s_{72} \cdot s_{73} \oplus s_{72} \oplus s_{73} \oplus 1$
73	$r_{67} \cdot r_{73} \oplus r_{72} \oplus r_{73} \cdot s_{34}$	$s_{72} \oplus s_{73} \cdot s_{74} \oplus s_{74}$
74	$r_{67} \cdot r_{74} \oplus r_{73} \oplus r_{74} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{67} \cdot s_{99} \oplus s_{73} \oplus s_{74} \cdot s_{75} \oplus s_{74} \oplus s_{75} \oplus 1$
75	$r_{67} \cdot r_{75} \oplus r_{74} \oplus r_{75} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{67} \cdot s_{99} \oplus s_{74} \oplus s_{75} \cdot s_{76} \oplus s_{75} \oplus s_{76} \oplus s_{99} \oplus 1$
76	$r_{67} \cdot r_{76} \oplus r_{75} \oplus r_{76} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{67} \cdot s_{99} \oplus s_{75} \oplus s_{76} \cdot s_{77} \oplus s_{76} \oplus s_{77} \oplus s_{99} \oplus 1$
77	$r_{67} \cdot r_{77} \oplus r_{76} \oplus r_{77} \cdot s_{34}$	$s_{76} \oplus s_{77} \cdot s_{78} \oplus s_{77} \oplus s_{78} \oplus 1$
78	$r_{67} \cdot r_{78} \oplus r_{77} \oplus r_{78} \cdot s_{34}$	$s_{77} \oplus s_{78} \cdot s_{79} \oplus s_{99}$
79	$r_{67} \cdot r_{79} \oplus r_{78} \oplus r_{79} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{67} \cdot s_{99} \oplus s_{78} \oplus s_{79} \cdot s_{80} \oplus s_{80}$
80	$r_{67} \cdot r_{80} \oplus r_{79} \oplus r_{80} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{67} \cdot s_{99} \oplus s_{79} \oplus s_{80} \cdot s_{81}$
81	$r_{67} \cdot r_{81} \oplus r_{80} \oplus r_{81} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{67} \cdot s_{99} \oplus s_{80} \oplus s_{81} \cdot s_{82} \oplus s_{81} \oplus s_{82} \oplus 1$
82	$r_{67} \cdot r_{82} \oplus r_{81} \oplus r_{82} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{67} \cdot s_{99} \oplus s_{81} \oplus s_{82} \cdot s_{83} \oplus s_{83} \oplus s_{99}$
83	$r_{67} \cdot r_{83} \oplus r_{82} \oplus r_{83} \cdot s_{34}$	$s_{82} \oplus s_{83} \cdot s_{84} \oplus s_{84} \oplus s_{99}$
84	$r_{67} \cdot r_{84} \oplus r_{83} \oplus r_{84} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{67} \cdot s_{99} \oplus s_{83} \oplus s_{84} \cdot s_{85} \oplus s_{85}$
85	$r_{67} \cdot r_{85} \oplus r_{84} \oplus r_{85} \cdot s_{34}$	$s_{84} \oplus s_{85} \cdot s_{86} \oplus s_{86} \oplus s_{99}$
86	$r_{67} \cdot r_{86} \oplus r_{85} \oplus r_{86} \cdot s_{34}$	$s_{85} \oplus s_{86} \cdot s_{87} \oplus s_{86} \oplus s_{87} \oplus s_{99} \oplus 1$
87	$r_{67} \cdot r_{87} \oplus r_{86} \oplus r_{87} \cdot s_{34} \oplus r_{99}$	$s_{86} \oplus s_{87} \cdot s_{88} \oplus s_{87} \oplus s_{99}$
88	$r_{67} \cdot r_{88} \oplus r_{87} \oplus r_{88} \cdot s_{34} \oplus r_{99}$	$s_{87} \oplus s_{88} \cdot s_{89} \oplus s_{88} \oplus s_{89} \oplus 1$
89	$r_{67} \cdot r_{89} \oplus r_{88} \oplus r_{89} \cdot s_{34} \oplus r_{99}$	$s_{88} \oplus s_{89} \cdot s_{90}$
90	$r_{67} \cdot r_{90} \oplus r_{89} \oplus r_{90} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{67} \cdot s_{99} \oplus s_{89} \oplus s_{90} \cdot s_{91} \oplus s_{91} \oplus s_{99}$
91	$r_{67} \cdot r_{91} \oplus r_{90} \oplus r_{91} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{67} \cdot s_{99} \oplus s_{90} \oplus s_{91} \cdot s_{92} \oplus s_{99}$
92	$r_{67} \cdot r_{92} \oplus r_{91} \oplus r_{92} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{67} \cdot s_{99} \oplus s_{91} \oplus s_{92} \cdot s_{93} \oplus s_{92} \oplus s_{99}$
93	$r_{67} \cdot r_{93} \oplus r_{92} \oplus r_{93} \cdot s_{34}$	$s_{92} \oplus s_{93} \cdot s_{94}$
94	$r_{67} \cdot r_{94} \oplus r_{93} \oplus r_{94} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{67} \cdot s_{99} \oplus s_{93} \oplus s_{94} \cdot s_{95}$
95	$r_{67} \cdot r_{95} \oplus r_{94} \oplus r_{95} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{67} \cdot s_{99} \oplus s_{94} \oplus s_{95} \cdot s_{96} \oplus s_{95} \oplus s_{99}$
96	$r_{67} \cdot r_{96} \oplus r_{95} \oplus r_{96} \cdot s_{34} \oplus r_{99}$	$r_{33} \cdot s_{99} \oplus s_{67} \cdot s_{99} \oplus s_{95} \oplus s_{96} \cdot s_{97} \oplus s_{96} \oplus s_{99}$
97	$r_{67} \cdot r_{97} \oplus r_{96} \oplus r_{97} \cdot s_{34} \oplus r_{99}$	$s_{96} \oplus s_{97} \cdot s_{98} \oplus s_{98}$
98	$r_{67} \cdot r_{98} \oplus r_{97} \oplus r_{98} \cdot s_{34}$	$s_{97} \oplus s_{98} \cdot s_{99} \oplus s_{99}$
99	$r_{67} \cdot r_{99} \oplus r_{98} \oplus r_{99} \cdot s_{34}$	$r_{33} \cdot s_{99} \oplus s_{67} \cdot s_{99} \oplus s_{98}$

Before describing the attack, let us fix certain notations that will be used henceforth.

1. $R_t = [r_0^t, r_1^t, \ldots, r_{99}^t], S_t = [s_0^t, s_1^t, \ldots, s_{99}^t]$ is used to denote the internal states of the R, S registers at the beginning of the round t of the PRGA. That is, r_i^t, s_i^t respectively denotes the i^{th} bit of the registers R, S at the beginning of round t of the PRGA. Note that $r_i^{t+1} = \rho_i(R_t, S_t)$ and $s_i^{t+1} = \beta_i(R_t, S_t)$.

2. The value of the variables CONTROL_BIT_R and CONTROL_BIT_S, at the PRGA round t, are denoted by the variables CR_t, CS_t respectively. These bits are used by the R, S registers to exercise mutual self control over each other. Note that $CR_t = r_{67}^t \oplus s_{34}^t$ and $CS_t = r_{33}^t \oplus s_{67}^t$.

3. $R_{t,\Delta r_\phi}(t_0), S_{t,\Delta r_\phi}(t_0)$ (resp. $R_{t,\Delta s_\phi}(t_0), S_{t,\Delta s_\phi}(t_0)$) are used to denote the internal states of the cipher at the beginning of round t of the PRGA, when a fault has been injected in location ϕ of R(resp. S) at the beginning of round t_0 of the PRGA.

4. $z_{i,\Delta r_\phi}(t_0)$ or $z_{i,\Delta s_\phi}(t_0)$ denotes the key-stream bit produced in the i^{th} PRGA round, after a fault has been injected in location ϕ of R or S at the beginning of round t_0 of the PRGA. By z_i, we refer to the fault-free key-stream bit produced in the i^{th} PRGA round.

With this background in hand, we present a summary of the results described in this chapter. The complete attack, assuming that the adversary is able to induce single bit faults in random register locations, is described in Section 7.4. In Section 7.5 we explore the case when the adversary is able to induce a fault that affects the bit values of (random) consecutive (upto 3) register locations. In Section 7.6 we propose improvements of the attack using SAT Solvers, in both the single bit-flip and the multiple bit-flip scenario. Section 7.7 concludes the chapter with a comparison of the state-of-the-art fault attacks currently reported against the stream ciphers in the hardware portfolio of eStream.

7.4 Complete description of the Attack

We start with some technical results that will be used later.

Lemma 7.1. *Consider the first* 100 *internal states of the MICKEY 2.0 PRGA. If* r_{99}^t *and* CR_t *are known* $\forall t \in [0, 99]$*, then the initial state* R_0 *can be calculated efficiently.*

Proof. Let the values of r_{99}^t and CR_t be known $\forall t \in [0, 99]$. We notice that the functions ρ_i for all values of $i \in [1, 99]$ are of the form $\rho_i(\cdot) = r_{i-1} \oplus (s_{34} \oplus r_{67}) \cdot r_i \oplus \alpha_i \cdot r_{99}$, where

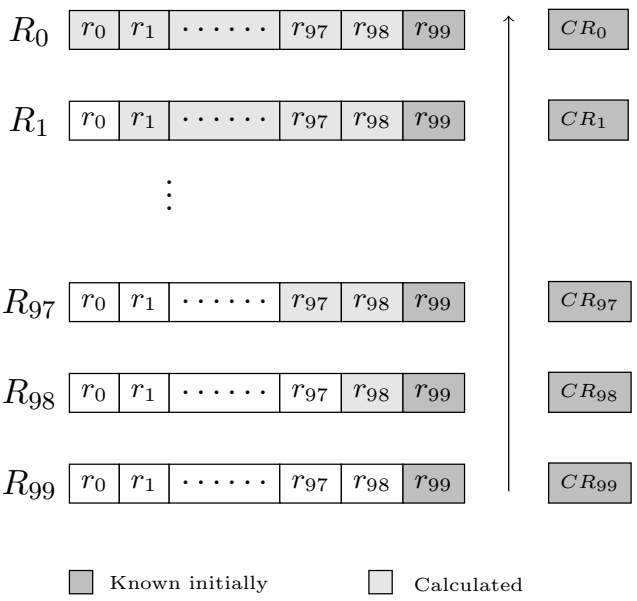


FIGURE 7.2: Constructing the state R_0. Starting from PRGA round 99, any bit calculated at PRGA round i is used to determine state bits of round $i-1$.

$s_{34} \oplus r_{67}$ is the value of CONTROL_BIT_R. It can be easily deduced that $\alpha_i = 1$, if $i \in RTAPS$ and is 0 otherwise. Now consider the following equation governing r_{99}^{99} :

$$r_{99}^{99} = \rho_{99}(R_{98}, S_{98}) = r_{98}^{98} \oplus CR_{98} \cdot r_{99}^{98} \oplus \alpha_{99} \cdot r_{99}^{98}.$$

In the above equation, r_{98}^{98} is the only unknown and it appears as a linear term, and so its value can be calculated immediately. We therefore know the values of 2 state bits of R_{98}: r_{99}^{98}, r_{98}^{98}. Similarly look at the equations governing r_{99}^{98}, r_{98}^{98}:

$$r_{99}^{98} = r_{98}^{97} \oplus CR_{97} \cdot r_{99}^{97} \oplus \alpha_{99} \cdot r_{99}^{97},$$

$$r_{98}^{98} = r_{97}^{97} \oplus CR_{97} \cdot r_{98}^{97} \oplus \alpha_{98} \cdot r_{99}^{97}.$$

As before, r_{98}^{97} is the lone unknown term in the first equation whose value is determined immediately. After this, r_{97}^{97} becomes the only unknown linear term in the next equation whose value too is determined easily. Thus we know 3 bits of R_{97}: r_{97+i}^{97}, $i = 0, 1, 2$. Continuing in such a bottom-up manner we can successively determine 4 bits of R_{96}, 5 bits of R_{95} and eventually all the 100 bits of R_0. (The process is explained pictorially in Figure 7.2.) □

Lemma 7.2. *Consider the first* 100 *internal states of the MICKEY 2.0 PRGA. If R_0 is known and s_{99}^t, CS_t, CR_t are known $\forall t \in [0, 99]$, then the initial state S_0 of the register S can be determined efficiently.*

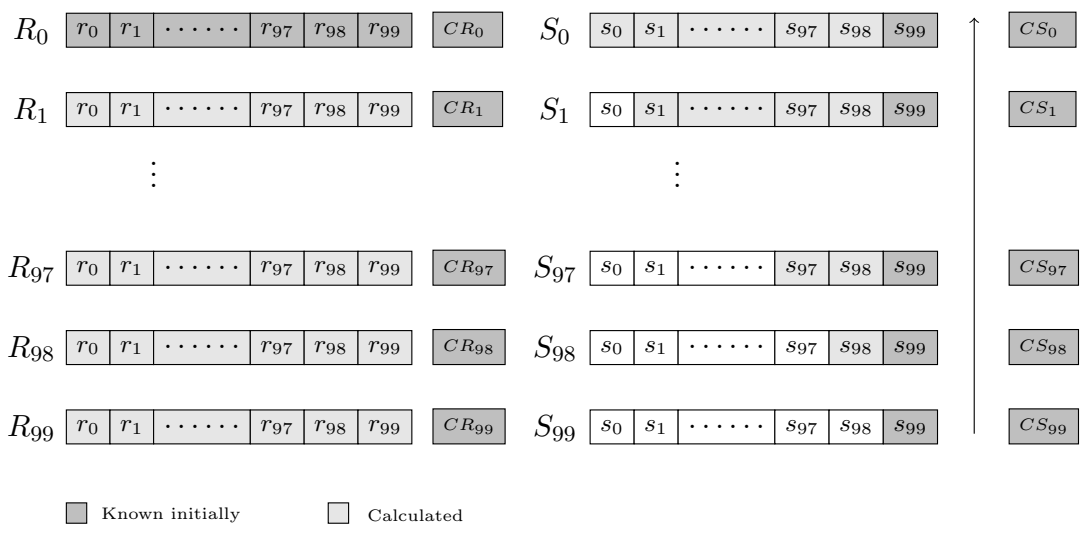


FIGURE 7.3: Constructing the state S_0. Starting from PRGA round 99, any bit calculated at PRGA round i is used to determine state bits of round $i-1$.

Proof. Since R_0 is known and so is CR_t for each $t \in [0, 99]$, we can construct all the bits of R_1 by calculating

$$r_i^1 = r_{i-1}^0 \oplus CR_0 \cdot r_i^0 \oplus \alpha_i \cdot r_{99}^0, \ \forall i \in [1, 99],$$

and r_0^1 is given by $r_0^0 \cdot CR_0 \oplus r_{99}^0$. Once all the bits of R_1 are known, all the bits of R_2 can be determined by calculating

$$r_i^2 = r_{i-1}^1 \oplus CR_1 \cdot r_i^1 \oplus \alpha_i \cdot r_{99}^1, \ \forall i \in [1, 99],$$

and $r_0^2 = r_0^1 \cdot CR_1 \oplus r_{99}^1$. Similarly all the bits of the states $R_3, R_4, \ldots, R_{99}$ can be calculated successively. As before, we begin by observing that the functions β_i for all values of $i \in [1, 99]$ are of the form

$$\beta_i(\cdot) = s_{i-1} \oplus \lambda_i \cdot (s_{67} \oplus r_{33}) \cdot s_{99} \oplus \hat{\beta}_i(s_i, s_{i+1}, \ldots, s_{99}),$$

where $s_{67} \oplus r_{33}$ is the value of CONTROL_BIT_S and $\hat{\beta}_i$ is a function that depends on $s_i, s_{i+1}, \ldots, s_{99}$ but not any of $s_0, s_1, \ldots, s_{i-1}$. It can be easily deduced that $\lambda_i = 1$ if $\mathsf{FB0}_i \neq \mathsf{FB1}_i$ and is 0 otherwise.

Now consider the following equation governing s_{99}^{99}:

$$s_{99}^{99} = \beta_{99}(R_{98}, S_{98}) = s_{98}^{98} \oplus \lambda_{99} \cdot CS_{98} \cdot s_{99}^{98} \oplus \hat{\beta}_{99}(s_{99}^{98}).$$

In the above equation s_{98}^{98} is the only unknown and it appears as a linear term, and so its value can be calculated immediately. We therefore know the values of the 2 state

i	$\theta_i(\cdot)$
0	$r_0 \oplus s_0$
1	$r_0 \cdot r_{67} \oplus r_0 \cdot s_{34} \oplus r_{99} \oplus s_{99}$
2	$r_0 \cdot r_{66} \cdot r_{67} \oplus r_0 \cdot r_{66} \cdot s_{34} \oplus r_0 \cdot r_{67} \cdot r_{99} \oplus$ $r_0 \cdot r_{67} \cdot s_{33} \oplus r_0 \cdot r_{67} \cdot s_{34} \cdot s_{35} \oplus r_0 \cdot r_{67} \cdot s_{34} \oplus$ $r_0 \cdot r_{67} \oplus r_0 \cdot r_{99} \cdot s_{34} \oplus r_0 \cdot s_{33} \cdot s_{34} \oplus r_0 \cdot s_{34} \cdot s_{35} \oplus$ $r_{33} \cdot s_{99} \oplus r_{66} \cdot r_{99} \oplus r_{67} \cdot r_{99} \cdot s_{34} \oplus r_{98} \oplus r_{99} \cdot s_{33} \oplus$ $r_{99} \cdot s_{34} \cdot s_{35} \oplus r_{99} \cdot s_{34} \oplus r_{99} \oplus s_{67} \cdot s_{99} \oplus s_{98}$

TABLE 7.3: The functions θ_i

bits of S_{98}: s_{99}^{98}, s_{98}^{98}. Similarly consider the equations involving s_{99}^{98}, s_{98}^{98} :

$$s_{99}^{98} = s_{98}^{97} \oplus \lambda_{99} \cdot CS_{97} \cdot s_{99}^{97} \oplus \hat{\beta}_{99}(s_{99}^{97}),$$

$$s_{98}^{98} = s_{97}^{97} \oplus \lambda_{98} \cdot CS_{97} \cdot s_{99}^{97} \oplus \hat{\beta}_{98}(s_{98}^{97}, s_{99}^{97}).$$

As before, s_{98}^{97} is the lone unknown term in the first equation whose value can be determined immediately. After this, s_{97}^{97} becomes the only unknown linear term in the next equation whose value can also be obtained easily. Thus we know 3 bits of S_{97}: s_{97+i}^{97}, $i = 0, 1, 2$. Continuing in such a bottom-up manner, we can successively determine 4 bits of S_{96}, 5 bits of S_{95} and eventually all the 100 bits of S_0. (The process is explained pictorially in Figure 7.3.) □

7.4.1 Faulting specific bits of R, S

The output key-stream bits $z_t, z_{t+1}, \ldots$ can also be expressed as polynomial functions over R_t, S_t. We have

$$\begin{aligned} z_t &= r_0^t \oplus s_0^t = \theta_0(R_t, S_t), \\ z_{t+1} &= r_0^{t+1} \oplus s_0^{t+1} \\ &= \rho_0(R_t, S_t) \oplus \beta_0(R_t, S_t) = \theta_1(R_t, S_t), \\ z_{t+2} &= r_0^{t+2} \oplus s_0^{t+2} \\ &= \rho_0(R_{t+1}, S_{t+1}) \oplus \beta_0(R_{t+1}, S_{t+1}) = \theta_2(R_t, S_t). \end{aligned}$$

The exact forms of $\theta_0, \theta_1, \theta_2$ are given in Table 7.3.

In the rest of this section we will assume that the adversary can (a) re-key the device containing the cipher with the original Key-IV, (b) apply faults to specific bit locations in the R, S registers and (c) exercise control over the timing of fault injection. Note that (b) is a stronger assumption, but we do not need it in our attack. We are using this

assumption here to build a sub-routine. In the next sub-section we shall demonstrate how the adversary can partially identify the location of any fault injected at a random position by comparing the faulty and fault-free key-streams.

We observe the following differential properties of the functions $\theta_0, \theta_1, \theta_2$.

- $\theta_1(\ldots, r_{67}, \ldots) \oplus \theta_1(\ldots, 1 \oplus r_{67}, \ldots) = r_0$,
- $\theta_1(r_0, \ldots) \oplus \theta_1(1 \oplus r_0, \ldots) = s_{34} \oplus r_{67}$,
- $\theta_2(\ldots, s_{99}) \oplus \theta_2(\ldots, 1 \oplus s_{99}) = s_{67} \oplus r_{33}$.

These differential properties have the following immediate implications.

$$z_{t+1} \oplus z_{t+1,\Delta r_{67}}(t) = r_0^t \tag{7.1}$$

$$z_{t+1} \oplus z_{t+1,\Delta r_0}(t) = CR_t \tag{7.2}$$

$$z_{t+2} \oplus z_{t+2,\Delta s_{99}}(t) = CS_t \tag{7.3}$$

The above equations hold for all the values of $t = 0, 1, 2, \ldots$. This implies that if the adversary is able to re-key the device with the original Key-IV pair multiple times and apply faults at the PRGA rounds $t = 0, 1, 2, 3, \ldots, 100$ at precisely[1] the R register locations $0, 67$ and the S register location 99, then by observing the difference between the fault-free and faulty key-stream bits, he would be able to recover the values of r_0^t, CR_t, CS_t for all values of $t = 0, 1, 2, \ldots, 100$. The fault at each register location must be preceded by re-keying.

Determining the other bits

Hereafter, the values s_0^t for all $t = 0, 1, 2, \ldots, 100$ may be found by solving: $s_0^t = z_t \oplus r_0^t$. Since $\beta_0(\cdot) = s_{99}$, this implies that $s_0^{t+1} = s_{99}^t,\ \forall t = 0, 1, 2, \ldots$. Therefore, calculating the values of $s_0^t,\ \forall t \in [1, 100]$ is the same as calculating $s_{99}^t,\ \forall t \in [0, 99]$. The values of $r_{99}^t,\ \forall t \in [0, 99]$ are obtained as follows. Consider the equation for z_{t+1}:

$$\begin{aligned} z_{t+1} = \theta_1(R_t, S_t) &= r_0^t \cdot r_{67}^t \oplus r_0^t \cdot s_{34}^t \oplus r_{99}^t \oplus s_{99}^t \\ &= CR_t \cdot r_0^t \oplus r_{99}^t \oplus s_{99}^t,\ \forall t \in [0, 99]. \end{aligned}$$

[1] We would again like to point out that our actual attack does not need precise fault injection at all locations of R, S. This will be explained in the next sub-section.

Here, r_{99}^t is the only unknown linear term in these equations and hence its value too can be determined immediately. At this point, we have the following state bits with us:

$$[r_0^t,\ r_{99}^t,\ CR_t,\ s_0^t,\ s_{99}^t,\ CS_t],\quad \forall t \in [0, 99].$$

Now by using the techniques presented in Lemma 7.1, we can determine all the bits of the state R_0. Thereafter using Lemma 7.2, one can determine all the bits of S_0. Thus we have recovered the entire internal state at the beginning of the PRGA.

7.4.2 How to identify the random locations where faults are injected

In this subsection we will show how the adversary can identify the locations of randomly applied faults to the registers R and S. Although it will not be possible to conclusively determine the location of faults applied to each and every location of R and the S registers, we will show that the adversary can, with some probability, identify faulty streams corresponding to locations $0, 67$ of R and 99 of S. The adversary will then use the techniques described in Subsection 7.4.1 to complete the attack.

To help with the process of fault location identification, as described for the Grain family in Section 5.2.2, we define the first and second Signature Vectors for the location ϕ of R as

$$\Psi^1_{r_\phi}[i] = \begin{cases} 1, & \text{if } z_{t+i} = z_{t+i,\Delta r_\phi}(t) \text{ for all } R_t, S_t, \\ 0, & \text{otherwise.} \end{cases}$$
$$\Psi^2_{r_\phi}[i] = \begin{cases} 1, & \text{if } z_{t+i} \neq z_{t+i,\Delta r_\phi}(t) \text{ for all } R_t, S_t, \\ 0, & \text{otherwise.} \end{cases}$$

for $i = 0, 1, 2, \ldots, l-1$. Here $l \approx 40$ is a suitably chosen constant.

Remark 7.3. The value of l should be large enough so that one can differentiate, with probability almost 1, 100 randomly generated bit sequences over GF(2) by comparing the first l bits of each sequence. This requires the value of l to be at least $2 \cdot \log_2 100 \approx 14$. We take $l = 40$, as computer simulations show that this value of l is sufficient to make a successful distinction with high probability.

Similarly one can define Signature Vectors for any location ϕ the register S.

$$\Psi^1_{s_\phi}[i] = \begin{cases} 1, & \text{if } z_{t+i} = z_{t+i,\Delta s_\phi}(t) \text{ for all } R_t, S_t, \\ 0, & \text{otherwise.} \end{cases}$$
$$\Psi^2_{s_\phi}[i] = \begin{cases} 1, & \text{if } z_{t+i} \neq z_{t+i,\Delta s_\phi}(t) \text{ for all } R_t, S_t, \\ 0, & \text{otherwise.} \end{cases}$$

The task for the fault location identification routine is to determine the fault location ϕ of R (or S) by analyzing the difference between the sequences $z_t, z_{t+1}, \ldots$ and $z_{t,\Delta r_\phi}(t), z_{t+1,\Delta r_\phi}(t), \ldots$ (or $z_{t,\Delta s_\phi}(t), \ldots$) by using the Signature Vectors $\Psi^1_{r_\phi}, \Psi^2_{r_\phi}$ (or $\Psi^1_{s_\phi}, \Psi^2_{s_\phi}$). Note that the i^{th} bit of $\Psi^1_{r_\phi}$ is 1 if and only if the $(t+i)^{th}$ key-stream bits produced by R_t, S_t and $R_{t,\Delta r_\phi}(t), S_{t,\Delta r_\phi}(t)$ are the same for all choices of the internal state R_t, S_t and that i^{th} bit of $\Psi^2_{r_\phi}$ is 1 if the above key-stream bits are different for all choices of the internal state.

The concept of Signature Vectors to deduce the location of a randomly applied fault was introduced in Section 5.2.2 for the Grain family. However the analysis of Section 5.2.2 cannot be reproduced for MICKEY 2.0, since a lot of different register locations have the same Signature Vector. However one can observe the following which are important to mount the attack.

Theorem 7.4. *The following statements hold for Signature Vectors* $\Psi^1_{r_\phi}, \Psi^2_{r_\phi}, \Psi^1_{s_\phi}, \Psi^2_{s_\phi}$ *of MICKEY 2.0.*

A. $\Psi^1_{r_\phi}[0] = 1, \forall \phi \in [1, 99]$ *and* $\Psi^2_{r_0}[0] = 1$.

B. $\Psi^1_{r_\phi}[0] = \Psi^1_{r_\phi}[1] = 1, \forall \phi \in [1, 99] \setminus \{67, 99\}$.

C. $\Psi^2_{r_{99}}[1] = 1$, *and* $\Psi^2_{r_{67}}[1] = 0$.

D. $\Psi^1_{s_\phi}[0] = 1, \forall \phi \in [1, 99]$ *and* $\Psi^2_{s_0}[0] = 1$.

E. $\Psi^1_{s_\phi}[0] = \Psi^1_{s_\phi}[1] = 1, \forall \phi \in [1, 99] \setminus \{34, 99\}$.

F. $\Psi^2_{s_{99}}[1] = 1$, *and* $\Psi^2_{s_{34}}[1] = 0$.

Proof. **A.** We have

$$\begin{aligned} z_t \oplus z_{t,\Delta r_0}(t) &= \theta_0(R_t, S_t) \oplus \theta_0(R_{t,\Delta r_0}(t), S_{t,\Delta r_0}(t)) \\ &= (r^t_0 \oplus s^t_0) \oplus (1 \oplus r^t_0 \oplus s^t_0) \\ &= 1, \ \forall R_t, S_t \in \{0,1\}^{100}. \end{aligned}$$

So, $\Psi^2_{r_0}[0] = 1$. Also θ_0 is not a function of any r_i, s_i for $i \in [1, 99]$ and so

$$\theta_0(R_{t,\Delta r_\phi}(t), S_{t,\Delta r_\phi}(t)) = \theta_0(R_t, S_t) \ \forall \phi \in [1, 99]$$

and so we have

$$\begin{aligned} z_t \oplus z_{t,\Delta r_\phi}(t) &= \theta_0(R_t, S_t) \oplus \theta_0(R_{t,\Delta r_\phi}(t), S_{t,\Delta r_\phi}(t)) \\ &= 0, \ \forall \phi \in [1, 99], \ \forall R_t, S_t \in \{0,1\}^{100}. \end{aligned}$$

So, $\Psi^1_{r_\phi}[0] = 1$ for all $\phi \in [1, 99]$.

B. Since θ_1 is a function of $r_0, r_{67}, s_{34}, r_{99}, s_{99}$ only, for any $\phi \in [1, 99] \setminus \{67, 99\}$ we have

$$\theta_1(R_{t,\Delta r_\phi}(t), S_{t,\Delta r_\phi}(t)) = \theta_1(R_t, S_t).$$

Therefore $z_{t+1} + z_{t+1,\Delta r_\phi}(t)$ equals

$$\begin{aligned}&\theta_1(R_t, S_t) + \theta_1(R_{t,\Delta r_\phi}(t), S_{t,\Delta r_\phi}(t))\\ &= 0,\ \forall \phi \in [1, 99] \setminus \{67, 99\},\ \forall R_t, S_t \in \{0, 1\}^{100}.\end{aligned}$$

So, $\Psi^1_{r_\phi}[1] = 1$ for all $\phi \in [1, 99] \setminus \{67, 99\}$.

C. We have $z_{t+1} + z_{t+1,\Delta r_{99}}(t)$ equals

$$\begin{aligned}&\theta_1(R_t, S_t) + \theta_1(R_{t,\Delta r_{99}}(t), S_{t,\Delta r_{99}}(t))\\ &= (r^t_0 \cdot r^t_{67} + r^t_0 \cdot s^t_{34} + r^t_{99} + s^t_{99}) +\\ &\quad (r^t_0 \cdot r^t_{67} + r^t_0 \cdot s^t_{34} + 1 + r^t_{99} + s^t_{99})\\ &= 1,\ \forall R_t, S_t \in \{0, 1\}^{100}.\end{aligned}$$

So, $\Psi^2_{r_{99}}[1] = 1$. Also $z_{t+1} + z_{t+1,\Delta r_{67}}(t)$ equals

$$\begin{aligned}&\theta_1(R_t, S_t) + \theta_1(R_{t,\Delta r_{67}}(t), S_{t,\Delta r_{67}}(t))\\ &= (r^t_0 \cdot r^t_{67} + r^t_0 \cdot s^t_{34} + r^t_{99} + s^t_{99}) +\\ &\quad (r^t_0 \cdot (1 + r^t_{67}) + r^t_0 \cdot s^t_{34} + r^t_{99} + s^t_{99})\\ &= r^t_0 \neq 0 \text{ or } 1,\ \forall R_t, S_t \in \{0, 1\}^{100}.\end{aligned}$$

So, $\Psi^2_{r_{67}}[1] = 0$.

D. We have

$$\begin{aligned}z_t + z_{t,\Delta s_0}(t) &= \theta_0(R_t, S_t) + \theta_0(R_{t,\Delta s_0}(t), S_{t,\Delta s_0}(t))\\ &= (r^t_0 + s^t_0) + (r^t_0 + 1 + s^t_0)\\ &= 1,\ \forall R_t, S_t \in \{0, 1\}^{100}.\end{aligned}$$

So, $\Psi^2_{s_0}[0] = 1$. Also θ_0 is not a function of any r_i, s_i for $i \in [1, 99]$ and so

$$\theta_0(R_{t,\Delta s_\phi}(t), S_{t,\Delta s_\phi}(t)) = \theta_0(R_t, S_t)$$

for all $\phi \in [1, 99]$ and so we have

$$\begin{aligned} z_t + z_{t,\Delta s_\phi}(t) &= \theta_0(R_t, S_t) + \theta_0(R_{t,\Delta s_\phi}(t), S_{t,\Delta s_\phi}(t)) \\ &= 0,\ \forall \phi \in [1, 99],\ \forall R_t, S_t \in \{0,1\}^{100}. \end{aligned}$$

So, $\Psi^1_{s_\phi}[0] = 1$ for all $\phi \in [1, 99]$.

E. Since θ_1 is a function of $r_0, r_{67}, s_{34}, r_{99}, s_{99}$ only, for any $\phi \in [1, 99] \setminus \{34, 99\}$ we have

$$\theta_1(R_{t,\Delta s_\phi}(t), S_{t,\Delta s_\phi}(t)) = \theta_1(R_t, S_t).$$

Therefore $z_{t+1} + z_{t+1,\Delta s_\phi}(t)$ equals

$$\begin{aligned} &\theta_1(R_t, S_t) + \theta_1(R_{t,\Delta s_\phi}(t), S_{t,\Delta s_\phi}(t)) \\ &= 0,\ \forall \phi \in [1, 99] \setminus \{34, 99\},\ \forall R_t, S_t \in \{0,1\}^{100}. \end{aligned}$$

So, $\Psi^1_{s_\phi}[1] = 1$ for all $\phi \in [1, 99] \setminus \{34, 99\}$.

F. We have $z_{t+1} + z_{t+1,\Delta s_{99}}(t)$ equals

$$\begin{aligned} &\theta_1(R_t, S_t) + \theta_1(R_{t,\Delta s_{99}}(t), S_{t,\Delta s_{99}}(t)) \\ &= (r^t_0 \cdot r^t_{67} + r^t_0 \cdot s^t_{34} + r^t_{99} + s^t_{99}) + \\ &\quad (r^t_0 \cdot r^t_{67} + r^t_0 \cdot s^t_{34} + r^t_{99} + 1 + s^t_{99}) \\ &= 1,\ \forall R_t, S_t \in \{0,1\}^{100}. \end{aligned}$$

So, $\Psi^2_{s_{99}}[1] = 1$. Also $z_{t+1} + z_{t+1,\Delta s_{34}}(t)$ equals

$$\begin{aligned} &\theta_1(R_t, S_t) + \theta_1(R_{t,\Delta s_{34}}(t), S_{t,\Delta s_{34}}(t)) \\ &= (r^t_0 \cdot r^t_{67} + r^t_0 \cdot s^t_{34} + r^t_{99} + s^t_{99}) + \\ &\quad (r^t_0 \cdot r^t_{67} + r^t_0 \cdot (1 + s^t_{34}) + r^t_{99} + s^t_{99}) \\ &= r^t_0 \neq 0 \text{ or } 1,\ \forall R_t, S_t \in \{0,1\}^{100}. \end{aligned}$$

So, $\Psi^2_{s_{34}}[1] = 0$.

This concludes the proof. □

Now, consider the attack scenario in which the adversary is able to re-key the device with the same Key-IV multiple number of times and inject a single fault at a random location of register R at the beginning of any particular PRGA round $t \in [0, 100]$ and obtain faulty key-streams. He continues the process until he obtains 100 different faulty

key-streams corresponding to 100 different fault locations in R and for each $t \in [0, 100]$ (as mentioned earlier this is done by comparing the first l bits of each faulty key-stream sequence). Assuming that every location has equal probability of getting injected by fault, the above process on an average takes around $100 \cdot \sum_{i=1}^{100} \frac{1}{i} \approx 2^{9.02}$ faults [59] and hence re-keyings for each value of $t \in [0, 100]$ and hence a total of $101 \cdot 2^{9.02} \approx 2^{15.68}$ faults. The process has to be repeated for the S register, and so the expected number of faults is $2 \cdot 2^{15.68} = 2^{16.68}$.

If we define the vectors

$$Z_t = [z_t, z_{t+1}, \ldots, z_{t+l-1}]$$

and correspondingly

$$\Delta_{r_\phi} Z_t = [z_{t,\Delta r_\phi}(t), z_{t+1,\Delta r_\phi}(t), \ldots, z_{t+l-1,\Delta r_\phi}(t)],$$

then the adversary at this point has knowledge of the 100 differential key-streams $\eta_{t,r_\phi} = Z_t \oplus \Delta_{r_\phi} Z_t$ for each value of $t \in [0, 100]$. The adversary, however, does not know the exact fault location corresponding to any differential stream, i.e., he has been unable to assign fault location labels to any of the differential streams. With this information in hand, we shall study the implications of the observations **A** to **F**.

Implication of A: For any $t \in [0, 100]$, $\Psi^2_{r_0}[0] = 1$ guarantees that there is at least one differential stream with $\eta_{t,r_\phi}[0] = 1$ whereas $\Psi^1_{r_\phi}[0] = 1, \forall \phi \in [1, 99]$ guarantees that that there is exactly one differential stream with this property. This implies that out of the 100 differential streams for any PRGA round t the one and only differential stream with this property must have been produced due to a fault on the 0^{th} location in R. Labelling of this stream helps us determine the values of CR_t for all $t \in [0, 100]$ from Equation (7.2).

Implication of B, C: Once the differential stream corresponding to the 0^{th} location has been labelled we now turn our attention to the remaining 99 streams. Statement **B** guarantees that of the remaining 99 streams at least 97 have the property:

$$\text{(P1)}\ \eta_{t,r_\phi}[0] = \eta_{t,r_\phi}[1] = 0.$$

Statement **C** guarantees that the number of streams with the property:

$$\text{(P2)}\ \eta_{t,r_\phi}[0] = 0, \eta_{t,r_\phi}[1] = 1,$$

is at most 2 and at least 1. If the number of streams that satisfy (P1) is 98 and (P2) is 1, then the lone stream satisfying (P2) must have been produced due to fault on location

99 of R. This immediately implies that $\eta_{t,r_{67}}[1] = 0$ which by Equation (7.1) in turn implies that $r_0^t = 0$. Else if the number of streams satisfying (P1) is 97 and (P2) is 2 then it implies that the streams satisfying (P2) were produced due to faults in location $67, 99$ of R. This implies $\eta_{t,r_{67}}[1] = r_0^t = 1$.

Repeating the entire process on Register S, one can similarly obtain the vectors $\Delta_{s_\phi} Z_t$ and the differential streams $\eta_{t,s_\phi} = Z_t \oplus \Delta_{s_\phi} Z_t$ for all values of $t \in [0, 100]$. As before the streams η_{t,s_ϕ} are unlabeled. Let us now study the implications of **D, E, F**.

Implication of D: For any $t \in [0, 100]$, $\Psi^2_{s_0}[0] = 1$ guarantees that there is at least one differential stream with $\eta_{t,s_\phi}[0] = 1$ whereas $\Psi^1_{s_\phi}[0] = 1, \forall \phi \in [1, 99]$ guarantees that that there is exactly one differential stream with this property. This implies that out of the 100 differential streams for any PRGA round t the one and only differential stream with this property must have been produced due to a fault on the 0^{th} location in S.

Implication of E, F: Once the differential stream corresponding to the 0^{th} location has been labelled we now turn our attention to the remaining 99 streams. The statement **E** guarantees that of the remaining 99 streams at least 97 have the property

$$\text{(P3) } \eta_{t,s_\phi}[0] = \eta_{t,s_\phi}[1] = 0.$$

Statement **F** guarantees that the number of streams with the property

$$\text{(P4) } \eta_{t,s_\phi}[0] = 0, \eta_{t,s_\phi}[1] = 1,$$

is at most 2 and at least 1.

Case 1. If the number of streams that satisfy (P3) is 98 and (P4) is 1 then the lone stream satisfying (P4) must have been produced due to fault at location 99 of S. Once the stream corresponding to location 99 of S had been labelled, we can use Equation (7.3) to determine $CS_t = \eta_{t,s_{99}}[2]$.

Case 2. If the number of streams satisfying (P3) is 97 and (P4) is 2 then it implies that the streams satisfying (P4) had been produced due to faults in location $34, 99$ of S.

(i) Now if the bit indexed 2 of both these vectors are equal then we can deduce $CS_t = \eta_{t,s_{99}}[2] = \eta_{t,s_{34}}[2]$.

(ii) A confusion occurs when $\eta_{t,s_{99}}[2] \neq \eta_{t,s_{34}}[2]$. In such a situation we would be unable to conclusively determine the value of CS_t.

Assuming independence, we assume that **Cases 1, 2** have equal probability of occurrence. Given that **Case 2** occurs, we can also assume that one of **2(i), 2(ii)** occurs with equal probability. Therefore, the probability of confusion, i.e., the probability that we are unable to determine the value of CS_t for any t can be estimated as $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$. Let γ denote the number of $t \in [0, 100]$ such that CS_t cannot be conclusively determined then γ is distributed according to $\gamma \sim Binomial(101, \frac{1}{4})$. Therefore the expected value of γ is $\mathbf{E}[\gamma] = 101 \cdot \frac{1}{4} = 25.25$. Also the probability that

$$P(\gamma > 35) = \sum_{k=36}^{101} \binom{101}{k} \left(\frac{1}{4}\right)^k \left(\frac{3}{4}\right)^{101-k} \approx 0.01.$$

In such a situation, the adversary must guess the γ number of bit values of CS_t to perform the attack, which implies that the adversary must perform the calculations in Section 7.4.1 and Lemma 7.1, Lemma 7.2 a total of 2^γ times to complete the attack. For the correct value of the guesses, the calculated state R_0, S_0 will produce the given fault-free key-stream sequence. We present a complete description of the attack in Algorithm 7.1.

7.4.3 Issues related to the length of the IV

It is known that MICKEY 2.0 employs a variable length IV of length at most 80. So if v is the length of the IV then the cipher will run for $v+80$ (Key loading) $+100$ (Preclock) clock rounds before entering the PRGA phase. Our attack requires that the first faults are to be injected at the beginning of the PRGA. In order to do that the adversary must know the value of v. This not a strong assumption as IVs are assumed to be known. However even if the adversary does not know the IV or its length the attack can be performed. Since $0 \leq v \leq 80$ must be satisfied, the strategy of the adversary who does not know the value of v will be as follows. He will inject the first set of faults at clock round 260 which corresponds to the PRGA round $p = 260 - 180 - v = 80 - v$. After performing the attack, the adversary will end up constructing the internal state R_p, S_p instead of R_0, S_0. Finding the value of p by looking at the fault-free key-stream sequence is straightforward. However, finding R_0, S_0 is a slightly stronger result because, as reported in [80], there is a finite entropy loss for each state update operation in the MICKEY PRGA.

7.4.4 Complexity of the Attack

As mentioned in Section 7.4.2, the attack requires the adversary to obtain 100 different faulty key-streams corresponding to all the fault locations in R for PRGA rounds $t \in$

Generate and record the fault-free key-stream $z_0, z_1, z_2, \ldots$ for some Key-IV K, IV
$t \leftarrow 0$;
while $t \leq 100$ **do**
  while 100 *different faulty key-stream sequences* $\Delta_{r_\phi} Z_t$ *have not been obtained* **do**
    Re-key the cipher with Key-IV K, IV;
    Inject a fault at a random unknown location $\phi \in [0, 99]$ in R at PRGA round t;
    Record the faulty key-stream sequence $\Delta_{r_\phi} Z_t$;
  end
  $t \leftarrow t + 1$;
end
Calculate $r_0^t, CR_t, \ \forall t \in [0, 100]$ using **A, B, C**;
$t \leftarrow 0$;
while $t \leq 100$ **do**
  while 100 *different faulty key-stream sequences* $\Delta_{s_\phi} Z_t$ *have not been obtained* **do**
    Re-key the cipher with Key-IV K, IV;
    Inject a fault at a random unknown location $\phi \in [0, 99]$ in S at PRGA round t;
    Record the faulty key-stream sequence $\Delta_{s_\phi} Z_t$;
  end
  $t \leftarrow t + 1$;
end
Using **D, E, F** calculate CS_t, for all such $t \in [0, 100]$ for which there is no confusion;
Let the number of undecided CS_t bits be γ;
for *Each of the* 2^γ *guesses of the undecided* CS_t*'s* **do**
  Use techniques of Subsection 7.4.1 to compute
  $r_0^t, \ r_{99}^t, \ CR_t, \ s_0^t, \ s_{99}^t, \ CS_t, \ \ \forall t \in [0, 99]$;
  Use Lemma 7.1, Lemma 7.2 to compute R_0, S_0;
  if R_0, S_0 *produce the sequence* $z_0, z_1, z_2, \ldots$ **then**
    Output the required state R_0, S_0;
  end
end

Algorithm 7.1: Fault Attack against MICKEY 2.0

$[0, 100]$. This requires $101 \cdot 100 \cdot \sum_{i=1}^{100} \frac{1}{k} \approx 2^{15.68}$ faults on an average. The same process must be repeated for the register S and hence the expected number of total faults is $2^{16.68}$. The computational overload comes from guessing the γ bits of CS_t which cannot be found by observing the differential key-streams. This requires a computational effort proportional to 2^γ. Since γ is distributed according to $Binomial(101, \frac{1}{4})$, the expected value of γ is 25.25. The expected value of the computation complexity is therefore given by

$$\mathbf{E}[2^\gamma] = \sum_{k=0}^{101} \binom{101}{k} \left(\frac{1}{4}\right)^k \left(\frac{3}{4}\right)^{101-k} 2^k \approx 2^{32.5}.$$

7.5 Case of Multiple bit faults

In this section we explore the situation in which the adversary is unable to induce a single bit flip of the internal state every time he injects a fault. We assume that the injection of fault may affect the bit values of at most three consecutive locations of the state (indeed this can be extended further, but the analysis will become very tedious). This gives rise to three situations (a) the attacker flips exactly one register bit (100 possibilities), (b) he flips 2 consecutive locations $i, i+1$ of R or S (99 possibilities), (c) he flips 3 consecutive locations $i, i+1, i+2$ of R or S (98 possibilities). Studying such a model makes sense if we attack an implementation of MICKEY where the register cells of the R and S registers are physically positioned linearly one after the other. Now, this attack scenario gives rise to $100 + 99 + 98 = 297$ different instances of faults due to any single fault injection, and we will assume that all these instances are equally likely to occur. As before we will assume that the adversary can re-key the device with the original Key-IV and obtain all the 297 faulty streams for any PRGA round $t \in [0, 100]$ by randomly injecting faults in either the R or S register. For each PRGA round, the attacker thus needs around $297 \cdot \sum_{i=1}^{297} \frac{1}{i} \approx 2^{10.7}$ faults. Thus the fault requirement for the register R is $101 \cdot 2^{10.7} = 2^{17.4}$. The process has to be repeated for the S register and so the total fault requirement is $2 \cdot 2^{17.4} = 2^{18.4}$.

Let $\mathbf{\Phi} = \{\phi_1, \phi_2, \ldots, \phi_k\}$ denote the set of indices of k ($k \leq 3$) continuous locations in the R (or S) register. The notations $R_{t,\Delta r_{\mathbf{\Phi}}}(t_0), S_{t,\Delta r_{\mathbf{\Phi}}}(t_0)$, $R_{t,\Delta s_{\mathbf{\Phi}}}(t_0)$, $S_{t,\Delta s_{\mathbf{\Phi}}}(t_0)$, $z_{i,\Delta r_{\mathbf{\Phi}}}(t_0), \Delta_{r_{\mathbf{\Phi}}} Z_t, \eta_{t,r_{\mathbf{\Phi}}}$, $\Psi^1_{r_{\mathbf{\Phi}}}[i], \Psi^2_{r_{\mathbf{\Phi}}}[i]$, and $\Psi^1_{s_{\mathbf{\Phi}}}[i]$ and $\Psi^2_{s_{\mathbf{\Phi}}}[i]$ will be used in their usual meanings in the context of multiple faults at all locations in $\mathbf{\Phi}$.

To begin with, in the single bit fault case, the attack depends on the successful identification of the faulty streams produced due to faults in locations $0, 67$ of R and 99 of S. In the multiple bit fault case too, the success of the attack depends on the identification of faulty streams that have been produced due to faults in these locations. We will deal each of these situations separately.

7.5.1 The bit r_0 is affected.

This could happen in 3 ways: a) r_0 alone is toggled, b) r_0, r_1 are toggled, c) r_0, r_1, r_2 are toggled. Let us state the following technical result.

Proposition 7.5. $\Psi^1_{r_{\mathbf{\Phi}}}[0] = 1, \forall \mathbf{\Phi}$ *such that* $0 \notin \mathbf{\Phi}$, *but* $\Psi^2_{r_{\mathbf{\Phi}}}[0] = 1, \forall \mathbf{\Phi}$ *that contain* 0.

Proof. Since θ_0 is a function of r_0, s_0 only we will have

$$z_t \oplus z_{t,\Delta r_{\mathbf{\Phi}}}(t) = \theta_0(R_t, S_t) \oplus \theta_0(R_{t,\Delta r_{\mathbf{\Phi}}}(t), S_{t,\Delta r_{\mathbf{\Phi}}}(t))$$
$$= \begin{cases} 0, & \text{if } 0 \notin \mathbf{\Phi}, \\ 1, & \text{if } 0 \in \mathbf{\Phi} \end{cases}$$

Hence the result. □

This implies that any faulty stream with its first bit different from the fault-free first bit must have been produced due to a fault that has affected r_0 and vice versa. Thus 3 out of the 297 faulty streams have this property and they can be identified easily. Furthermore since $\theta_1(R_t, S_t) \oplus \theta_1(R_{t,\Delta r_{\mathbf{\Phi}}}(t), S_{t,\Delta r_{\mathbf{\Phi}}}(t)) = s_{34}^t \oplus r_{67}^t = CR_t \ \forall \mathbf{\Phi}$ containing 0, the second bit in the all these faulty streams are equal and the difference of this bit with the second fault-free bit gives us the value of CR_t.

7.5.2 The bits r_{67} and r_{99} are affected.

r_{67} could be affected in 6 ways : a) r_{67} alone is toggled, b) r_{66}, r_{67} are toggled, c) r_{67}, r_{68} are toggled, d) r_{65}, r_{66}, r_{67} are toggled, e) r_{66}, r_{67}, r_{68} are toggled and f) r_{67}, r_{68}, r_{69} are toggled. Also note that r_{99} could be affected in 3 ways: a) r_{99} is toggled, b) r_{98}, r_{99} are toggled and c) r_{97}, r_{98}, r_{99} are all toggled. Again we state the following propositions.

Proposition 7.6. $\Psi^1_{r_{\mathbf{\Phi}}}[0] = \Psi^1_{r_{\mathbf{\Phi}}}[1] = 1, \forall \mathbf{\Phi}$ *such that the indices* $0, 67, 99 \notin \mathbf{\Phi}$.

Proposition 7.7. *If* $99 \in \mathbf{\Phi}$ *then* $\Psi^2_{r_{\mathbf{\Phi}}}[1] = 1$. *If* $67 \in \mathbf{\Phi}$ *then* $\Psi^2_{r_{\mathbf{\Phi}}}[1] = 0$.

Proof. Note that θ_0 is a function of only r_0, s_0 and θ_1 is a function of $r_0, r_{67}, r_{99}, s_{34}, s_{99}$ only.

$$z_{t+1} \oplus z_{t+1,\Delta r_{\mathbf{\Phi}}}(t) = \begin{cases} 0, & \text{if } 0, 67, 99 \notin \mathbf{\Phi}, & \text{(G)} \\ CR_t, & \text{if } 0 \in \mathbf{\Phi}, & \text{(H)} \\ r_0^t, & \text{if } 67 \in \mathbf{\Phi}, & \text{(K)} \\ 1, & \text{if } 99 \in \mathbf{\Phi}. & \text{(L)} \end{cases}$$

Hence the result. □

In the above, (G) implies that out of the remaining 294 differential streams at least $294 - 6 - 3 = 285$ satisfy

$$\text{(P5) } \eta_{t,r_{\mathbf{\Phi}}}[0] = \eta_{t,r_{\mathbf{\Phi}}}[1] = 0$$

and (L) implies that the number of differential streams with the property

$$\text{(P6) } \eta_{t,r_{\mathbf{\Phi}}}[0] = 0, \eta_{t,r_{\mathbf{\Phi}}}[1] = 1$$

is at least 3. A direct implication of (K) is that if the number of differential streams satisfying (P5) is 285 and (P6) is 9 then $r_0^t = 1$ and on the other hand if, the number of streams satisfying (P5) is 291 and (P6) is 3 then $r_0^t = 0$. These are exclusive cases, i.e., the number of streams satisfying (P5) can be either 285 or 291. Since the values of r_0^t, CR_t for all $t \in [0, 100]$ are now known, the attacker can now use the techniques of Section 7.4.1 and Lemma 7.1 to calculate the entire initial state R_0.

7.5.3 The bits s_0, s_{34} and s_{99} are affected.

Following previous descriptions, we know that there are respectively $3, 6, 3$ possibilities of faults affecting s_0, s_{34}, s_{99}. Again, we present the following technical results before describing the attack.

Proposition 7.8. $\Psi^1_{s_\Phi}[0] = 1, \forall \mathbf{\Phi}$ *such that* $0 \notin \mathbf{\Phi}$, *but* $\Psi^2_{s_\Phi}[0] = 1, \forall \mathbf{\Phi}$ *that contain* 0.

Proposition 7.9. $\Psi^1_{s_\Phi}[0] = \Psi^1_{s_\Phi}[1] = 1, \forall \mathbf{\Phi}$ *such that the indices* $0, 34, 99 \notin \mathbf{\Phi}$.

Proposition 7.10. *If* $99 \in \mathbf{\Phi}$ *then* $\Psi^2_{s_\Phi}[1] = 1$. *If* $34 \in \mathbf{\Phi}$ *then* $\Psi^2_{s_\Phi}[1] = 0$.

Proof. The proof is similar to those of previous propositions. Since θ_0 is a function of only r_0, s_0 and θ_1 is a function of $r_0, r_{67}, r_{99}, s_{34}, s_{99}$ only, we have

$$\begin{aligned} z_t \oplus z_{t,\Delta s_\Phi}(t) &= \theta_0(R_t, S_t) \oplus \theta_0(R_{t,\Delta s_\Phi}(t), S_{t,\Delta s_\Phi}(t)) \\ &= \begin{cases} 0, & \text{if } 0 \notin \mathbf{\Phi}, \\ 1, & \text{if } 0 \in \mathbf{\Phi} \end{cases} \end{aligned}$$

$$z_{t+1} \oplus z_{t+1,\Delta s_\Phi}(t) = \begin{cases} 0, & \text{if } 34, 99 \notin \mathbf{\Phi}, & \text{(M)} \\ r_0^t, & \text{if } 34 \in \mathbf{\Phi}, & \text{(N)} \\ 1, & \text{if } 99 \in \mathbf{\Phi}. & \text{(O)} \end{cases}$$

□

Proposition 7.8 proves that there are exactly 3 differential streams out of 297 which have $\eta_{s_\Phi}[0] = 1$. Further, (M) implies that of the remaining 294 streams, at least $294 - 3 - 6 = 285$ satisfy

$$\text{(P7) } \eta_{t,s_\Phi}[0] = \eta_{t,s_\Phi}[1] = 0$$

and (O) implies that the number of streams that satisfy

$$\text{(P8) } \eta_{t,s_\Phi}[0] = 0, \eta_{t,s_\Phi}[1] = 1$$

is at least 3.

CASE I.

If the number of streams that satisfy (P7) is 291 and (P8) is 3 then the streams satisfying (P8) must have been produced due to the faults affecting s_{99}. For these streams $\eta_{s_{\mathbf{\Phi}}}[2]$ is given by:

$$z_{t+2} \oplus z_{t+2,\Delta s_{\mathbf{\Phi}}}(t) = \begin{cases} CS_t, & \text{if } \mathbf{\Phi} = \{99\}, \\ 1 \oplus CS_t, & \text{if } \mathbf{\Phi} = \{98, 99\} \\ 1 \oplus CS_t. & \text{if } \mathbf{\Phi} = \{97, 98, 99\} \end{cases}$$

So, for 2 of these 3 streams we have $\eta_{s_{\mathbf{\Phi}}}[2] = 1 \oplus CS_t$. Hence, our strategy will be to look at the bit indexed 2 of these 3 streams. Two of them will be equal and we designate that value as $1 \oplus CS_t$.

CASE II.

If the number of streams that satisfy (P7) is 285 and (P8) is 9 then the streams have been produced due to faults that have affected s_{34} and s_{99}. We have the identity

$$\sum_{\mathbf{\Phi}:\ 34 \in \mathbf{\Phi}} \eta_{t,s_{\mathbf{\Phi}}}[2] = r_0^t \cdot r_{67}^t \cdot s_{34}^t \oplus r_{99}^t \cdot s_{34}^t.$$

Therefore, the sum of the bits indexed 2 of all the differential streams that satisfy (P8) is

$$\sum_{\mathbf{\Phi}:\ 34 \text{ or } 99 \in \mathbf{\Phi}} \eta_{t,s_{\mathbf{\Phi}}}[2] = CS_t \oplus r_0^t \cdot r_{67}^t \cdot s_{34}^t \oplus r_{99}^t \cdot s_{34}^t.$$

At this time the entire initial state of the register R and all the values of CR_t for $t \in [0, 100]$ are known to us. Hence, by Lemma 7.2, all values of r_i^t for all $t > 0$ can be calculated by clocking the register R forward. Also, since $CR_t = r_{67}^t \oplus s_{34}^t$ is known, $s_{34}^t = CR_t \oplus r_{67}^t$ can be calculated easily. Therefore, in the previous equation, CS_t becomes the only unknown and thus its value can be calculated immediately.

At this point of time we have r_0^t, CR_t, CS_t for all values of $t \in [0, 100]$. Now using the techniques of Section 7.4.1 and Lemmata 7.1, 7.2, we will be able to determine the entire initial state R_0, S_0. Note that using this fault model although the fault requirement increases, the adversary does not have to bear the additional computational burden of guessing γ values of CS_t.

7.6 Improvement Using SAT Solver

The main idea of algebraic cryptanalysis is to solve multivariate polynomial systems that describe a cipher and this has been successfully exploited in DFA also. For a very brief introduction in this, one may refer [109, Section 5]. The DFA on Trivium [109] requires only 2 faults. The work on DFA against Grain family described in Section 5.5 also shows that the number of faults can be reduced significantly (not more than 10). With this motivation, we tried to exploit similar ideas for fault attacks against MICKEY 2.0. Our analysis shows improvements over our result in Section 7.4.4; however, not as significant as what could be achieved for Trivium or Grain family. Nevertheless, we identify several other combinatorial patterns towards the improved DFA against MICKEY 2.0 in this section. We will start with the following simple technical result.

Lemma 7.11. *Suppose $r_0^t = 0$ for some $t \in [0, 99]$. Then the location of a random fault can be identified deterministically when it injects the 99^{th} location of R.*

Proof. This follows from Theorem 7.4B, 7.4C. We have already seen in Section 7.4.2, that for any t, if $r_0^t = 0$, then the number of differential streams satisfying (P2) is exactly 1. It follows from Theorem 7.4B, 7.4C, that this differential stream must have been produced due to fault on location 99 of R. □

Now we will prove another result when $r_0^t = 1$.

Lemma 7.12. *Suppose $r_0^t = 1$ for some $t \in [0, 99]$. Then to decide that r_0^t is indeed 1 and furthermore to find the value of CR_t, one needs to inject around 183.33 faults on average.*

Proof. From Theorem 7.4A, 7.4B, 7.4C and their implications, it is clear that if $r_0^t = 1$, then the number of differential streams satisfying (P2) is 2 (produced due to faults on locations $67, 99$ of R) and if $r_0^t = 0$, then the number of differential streams satisfying (P2) is 1. Hence for any t, in the process of applying random faults, as soon as the attacker obtains 2 streams satisfying (P2), he can conclude that $r_0^t = 1$.

Also from the implication of Theorem 7.4A, we know that finding CR_t requires the faulty key-stream from location 0 of R. So, for any fixed t, if $r_0^t = 1$, then deducing r_0^t and CR_t requires faulty key-streams from locations $0, 67, 99$ of R only, i.e, the faulty keystream from location 0 of R and the two faulty keystreams satisfying (P2). By injecting random faults, the attacker can expect to inject these 3 locations by applying $100 + \frac{100}{2} + \frac{100}{3} = 183.33$ random faults. Hence the result. □

Note that, this is much less than the $2^{9.02}$ faults required to obtain the 100 distinct faulty key-streams corresponding to each fault location in R as discussed in Section 7.4.2.

Hence when $r_0^t = 1$, we do not need to inject fault at every location of R_t to find the value of r_0^t and CR_t. On the other hand when $r_0^t = 0$, using Lemma 7.11, we can identify the faulty key-stream resulting from fault on location 99 of R. We will use these faulty key-streams as location of fault is known in our attack.

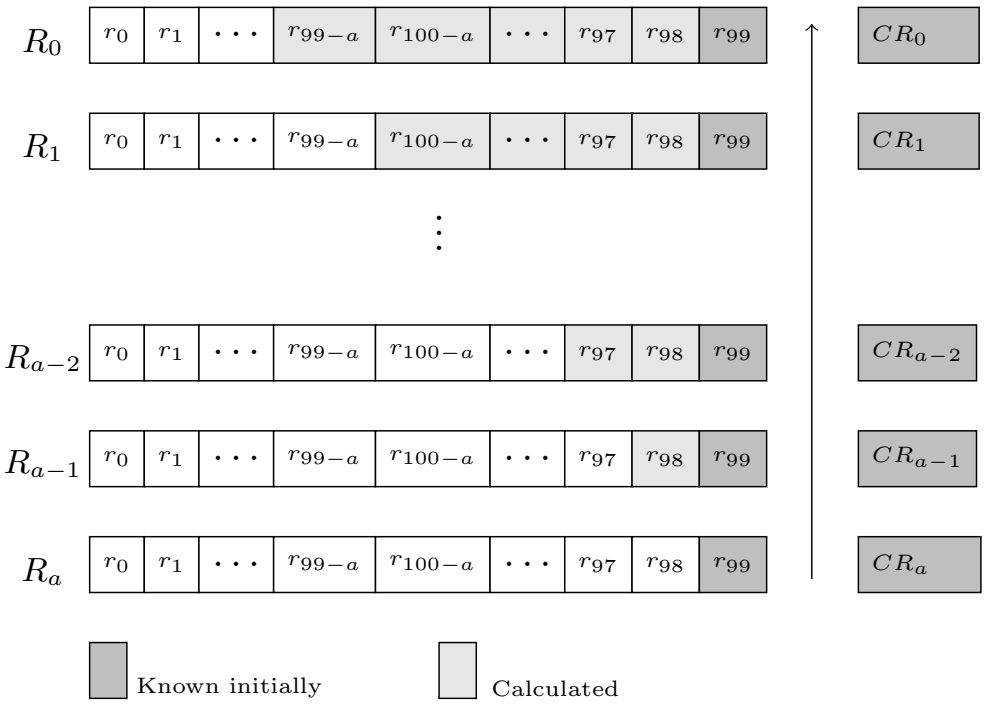


FIGURE 7.4: Constructing the last a bits of the state R_0.

We will now state a more general form of Lemma 7.1.

Lemma 7.13. *Let $a \in [0, 99]$ be an integer. If we assume that r_{99}^t and CR_t are known $\forall t \in [0, a]$, then the state bits $r_{99-a}^0, r_{100-a}^0, \ldots, r_{99}^0$ of the initial state R_0 may be calculated efficiently.*

Proof. The proof is exactly similar to that of Lemma 7.1. In Lemma 7.1, we started with 1 bit of R_{99}, i.e., r_{99}^{99}, and then worked backwards to calculate the last 2 bits of R_{98}, 3 bits of R_{97} and in this manner the entire of R_0. In this case we will start with 1 bit of R_a, i.e., r_{99}^a and backtrack to calculate the last 2 bits of R_{a-1}, 3 bits of R_{a-2} and in this manner the bits $r_{99-a}^0, r_{100-a}^0, \ldots, r_{99}^0$ of R_0. The process is explained pictorially in Figure 7.4. □

We will now investigate the situation when attacker injects faults at each round $t \in [0, a]$. Using Theorem 7.4 and its implications, the attacker can deduce the values of r_0^t and

$CR_t \ \forall \ t \in [0, a]$. He can then find the values of r^t_{99}, $\forall \ t \in [0, a]$, using the arguments of Section 7.4.1. Then using Lemma 7.13, he can compute $r^0_{99-a}, r^0_{100-a}, \ldots, r^0_{99}$. Now, let us write the state R_0 as

$$[r^0_0, x_1, \ldots, x_{98-a}, r^0_{99-a}, r^0_{100-a}, \ldots, r^0_{99}],$$

where x_i's are unknown for $1 \leq i \leq 98-a$ and r^0_i are known for $i = 0$ and $99-a \leq i \leq 99$. We can write the state S_0 as

$$[y_0, y_1, \ldots, y_{99}],$$

where y_i's are unknown for $0 \leq i \leq 99$.

We will now describe the technique to formulate multivariate equations in x_i, y_i over GF(2) which we will solve using a SAT Solver. We will formulate equations for the fault-free key-stream bits first. We have already seen that the state bits of $R_1, S_1, R_2, S_2, \ldots, R_k, S_k, \ldots$ can be expressed as polynomials over the state bits of R_0, S_0. However, the algebraic degree and complexity of these polynomials increase exponentially with increasing k. So much so that we could not compute the form of these polynomials for $k > 4$ on a normal Desktop PC. To circumvent this situation, we take resort to introducing new variables at every PRGA round of the cipher.

In the first round of PRGA, we introduce 200 new variables u^1_i and v^1_i for $0 \leq i \leq 99$, where u^1_i corresponds to the state R_1 and v^1_i corresponds to S_1. Hence we formulate 201 new equations which are

1. $z_0 = r^0_0 \oplus y_0$
2. $u^1_i = \rho_i(r^0_0, \ldots, x_{99-a}, r_{100-a}, \ldots, r_{99}, y_0, \ldots, y_{99})$
3. $v^1_i = \beta_i(r^0_0, \ldots, x_{99-a}, r_{100-a}, \ldots, r_{99}, y_0, \ldots, y_{99})$.

Hence, the states R_1 and S_1, obtained after running one round of PRGA, becomes

$$[u^1_0, \ldots, u^1_{99}] \text{ and } [v^1_0, \ldots, v^1_{99}]$$

respectively. This technique is repeated in each successive round accompanied by the introduction of 200 new variables. As MICKEY's state update function is highly non linear, this approach enables us to compute the symbolic forms (via a series of equations) of any PRGA state R_T, S_T. Instead, if at each round $k > 0$, the variables u^k_i, v^k_i were replaced by their equivalent algebraic expressions in x_i, y_i, this would never have been possible efficiently. By introducing new variables, after T rounds, we have a total of $201T$ equations.

We will now formulate equations generated due to faulty key-stream bits. The attacker can determine any faulty key-stream conclusively when it has been produced due to fault at location 0 of R. So after T rounds, we have a total of T faulty key-stream sequences generated due to fault on 0^{th} location of R. To use these faulty key-streams, we proceed as follows. Consider the case when an injected fault has toggled the location 0 of R at $t = 0$. We denote this faulty state by the vector

$$[1 \oplus r_0^0, x_1, \ldots, x_{98-a}, r_{99-a}^0, r_{100-a}^0, \ldots, r_{99}^0] \text{ and}$$

$$[y_0, y_1, \ldots, y_{99}].$$

As before we use 200 new variables $\overline{u}_i^1, \overline{v}_i^1$ to the next faulty state. So we again get 201 new equations

1. $z_{0,\Delta r_0}(0) = 1 \oplus r_0^0 \oplus y_0$
2. $\overline{u}_i^1 = \rho_i(1 \oplus r_0^0, \ldots, x_{99-a}, r_{100-a}, \ldots, r_{99}, \ldots, y_{99})$
3. $\overline{v}_i^1 = \beta_i(1 \oplus r_0^0, \ldots, x_{99-a}, r_{100-a}, \ldots, r_{99}, \ldots, y_{99})$.

As before, we repeat the above for T' rounds with 200 new variables in each round. Again, this results in a total of $201T'$ equations. The process can be repeated for fault at any round $t \in [0, T]$. New equations and variables are formulated accordingly in each case.

Again from Lemma 7.11, we know that $\forall\ t$, we can identify any faulty key-stream sequence produced due to fault on location 99 of R, when $r_0^t = 0$. So whenever $r_0^t = 0$, we can formulate more equations. For example if $r_0^0 = 0$, we start with the state

$$[r_0^0, x_1, \ldots, x_{98-a}, r_{99-a}^0, r_{100-a}^0, \ldots, 1 \oplus r_{99}^0] \text{ and}$$

$$[y_0, y_1, \ldots, y_{99}],$$

and thereafter form equations by the introduction of new variables in each round.

7.6.1 Experiments

We assume all except the first 25 bits bits of R_0 have been found out by injecting faults and thereafter using Lemma 7.13, i.e., we take $a = 75$. We need to find S_0 which contains 100 unknown bits. To restrict the total number of equations, we use only first 38 key-stream bits, i.e., we take $T = 38$. We also use faulty key-stream bits for only first $T' = 12$ rounds when the location of a faulty key-stream can be conclusively identified.

We feed the equation so formed into the SAT Solver Cryptominisat-2.9.5 [126] available with SAGE 5.7 [129]. The solver is able to find the remaining 125 unknown bits in 1345.80 seconds on an average (averaged over 100 trials) on a PC powered by an Intel Dual Core Processor, with a CPU speed of 1.83 GHz and 2 GB RAM.

Fault Requirement: Since $a = 75$, we need to apply faults in the first 75 PRGA rounds. Among the 75 rounds, we can assume value of r_0^t will be 1 at expected $\frac{75}{2}$ times. Whenever $r_0^t = 1$, by Lemma 7.12, only 183.33 faults are sufficient. So expected number of faults required in our attack is

$$\frac{75}{2} \cdot 100 \sum_{i=1}^{100} \frac{1}{i} + \frac{75}{2}\left(100 + \frac{100}{2} + \frac{100}{3}\right) \approx 2^{14.68}.$$

Thus we have a four-fold improvement in the number of faults compared to Section 7.4.4, where expected number of fault was $2^{16.68}$. This is the improvement achieved when we solve non-linear equations using a SAT solver.

7.6.2 Multiple bit faults

From the discussion in Section 7.5, it is clear that the attacker can not conclusively determine whether a given faulty key-stream has been produced due to a single bit or a multiple bit fault. Hence the attacker cannot use faulty key-streams to formulate equations. The best he can do is as follows. Find a bits of R_0 by applying faults and then find the remaining bits of R_0, S_0 by formulating equations for the fault-free key-stream bits. By extensive experimentation, we have found that to obtain a solution in reasonable time, the value of a has to be 100, i.e., we need to find out the entire state of R_0 before using the SAT solver. Using the technique of formulating equations using extra variables, which was described in the previous subsection, we were able to find the entire S_0 using the SAT Solver, within 1206.18 seconds on an average (averaged over 100 trials).

Fault Requirement: The number of different $\mathbf{\Phi}$ for which $0 \in \mathbf{\Phi}$ is 3 and as per Proposition 7.5, these are used to calculate the value of CR_t. Assuming $r_0^t = 1$, among the remaining $297 - 3 = 294$ different faulty key-streams, 9 would satisfy (P6). Of these 9, three are due to fault on location 99 and six are due to location 67. However when $r_0^t = 0$, the number of streams satisfying (P6) is only 3. Hence for any t, as soon as the attacker can obtain four different key-streams satisfying (P6), he can conclude $r_0^t = 1$.

When $r_0^t = 1$, we will prove in Theorem 7.14 that the attacker requires 187.25 faults on average to deduce the value of r_0^t and CR_t.

Theorem 7.14. *187.25 faults are sufficient to deduce $r_0^t = 1$ and find CR_t, in the multiple bit-flip scenario a fault flips the logic in atmost three contiguous register locations.*

Proof. We know that we have a total of 297 different faulty streams. To deduce that $r_0^t = 1$ and find CR_t, by injecting random faults, we need to obtain 4 different streams out of a set of 9 specific streams (satisfying (P6)) and 1 out of a set of 3 other streams (due to fault at location 0 of R) respectively. To find the expected number of faults to achieve this target, we will use the following proposition.

Proposition 7.15. *Consider five real numbers $a_1, \ldots, a_5$ in $(0, 1)$. Then, we have the following identities*

1. $$\sum_{r_1=0}^{\infty} \cdots \sum_{r_5=0}^{\infty} \left[\prod_{i=1}^{5} a_i^{r_i} \right] = \prod_{i=1}^{5} \frac{1}{(1-a_i)}$$
2. $$\sum_{\substack{r_1=0 \\ \vdots \\ r_5=0}}^{\infty} \left[\sum_{i=1}^{5} r_i \cdot \prod_{i=1}^{5} a_i^{r_i} \right] = \sum_{i=1}^{5} \frac{a_i}{(1-a_i)^2} \prod_{\substack{j=1 \\ j \neq i}}^{5} \frac{1}{(1-a_j)}$$

Suppose, we first obtain the 4 streams of the set of 9 in $r_1+1, r_1+r_2+2, r_1+r_2+r_3+3$ and $r_1 + r_2 + r_3 + r_4 + 4$ attempts respectively. Thereafter, we obtain the remaining streams from the set of 3 after another r_5+1 trials, i.e., we require $r_1+r_2+r_3+r_4+r_5+5$ faults in total. We call this event $\mathcal{E}_{r_1,\ldots,r_5}$. Then $\mathsf{Pr}(\mathcal{E}_{r_1,\ldots,r_5}) =$

$$\begin{aligned} & a_1^{r_1} \frac{9}{297} \cdot a_2^{r_2} \cdot \frac{8}{297} \cdot a_3^{r_3} \cdot \frac{7}{297} \cdot a_4^{r_4} \cdot \frac{6}{297} \cdot a_5^{r_5} \cdot \frac{3}{297} \\ & = a_1^{r_1} a_2^{r_2} a_3^{r_3} a_4^{r_4} a_5^{r_5} \cdot \frac{9072}{297^5}, \end{aligned}$$

where $a_1 = \frac{285}{297}, a_2 = \frac{286}{297}, a_3 = \frac{287}{297}, a_4 = \frac{288}{297}, a_5 = \frac{294}{297}$. Here a_i's denote the failure probabilities, i.e., a_i denotes the probability that, after obtaining $i-1$ required streams, a random fault produces no stream of interest.

We may also fulfill our target by some other "ordering" of events. For example, we first obtain 3 streams from the set of 9, then the single stream from the other set of 3 and finally the remaining stream from the first set. There are 5 orderings in total. Denote by b_i, c_i, d_i, e_i the failure probabilities, in each of the other orderings.

It is easy to see that, $b_1 = c_1 = d_1 = e_1 = a_1$, $b_2 = c_2 = d_2 = a_2, b_3 = c_3 = a_3$, $b_4 = a_4$, $b_5 = c_5 = d_5 = e_5 = \frac{291}{297}$, $c_4 = d_4 = e_4 = \frac{290}{297}$, $d_3 = e_3 = \frac{289}{297}$, $e_2 = \frac{288}{297}$.

Considering all cases, the required expected value is

$$E = \sum_{\substack{r_1=0 \\ \vdots \\ r_5=0}}^{\infty} \left(5 + \sum_{i=0}^{5} r_i\right) \left(\prod_{i=1}^{5} a_i^{r_i} + \cdots + \prod_{i=1}^{5} e_i^{r_i}\right) \cdot \frac{9072}{297^5}$$

Now using Proposition 7.15, we get $E = 187.25$.

□

On the other hand if $r_0^t = 0$, there is a total of 291 different faulty streams which satisfy (P5) and only 3 which satisfy (P6). Now in the process of applying random fault and resetting, as soon as we obtain 286 streams that satisfy (P5), we can conclude that $r_0^t = 0$. Hence in this case, the expected number of faults is approximately $297 \cdot \sum_{i=6}^{291} \frac{1}{i} = 1178.77$.

Thus, the expected number of faults required to find R_0 is

$$\frac{100}{2}\left(187.2 + 1178.77\right) \approx 2^{16.06}.$$

This is more than four-fold improvement over the $2^{18.4}$ faults reported in Section 7.5.

7.7 Conclusion

A differential fault attack against the stream cipher MICKEY 2.0 is presented. The work is one of the first cryptanalytic attempts against this cipher and requires reasonable computational effort. The attack works due to the simplicity of the output function and certain register update operations of MICKEY 2.0 and would have been thwarted had these been of a more complex nature. It would be interesting to study efficient counter-measures with minimum tweak in the design.

Given our work in this chapter, differential fault attacks are now known against all of the three ciphers in the hardware portfolio of eStream. The attacks on all the 3 ciphers use exactly the same fault model that is similar to what described in this chapter. Let us now summarize the fault requirements in Table 7.4.

To the best of our knowledge, there was no published fault attack on MICKEY 2.0. prior to our work. One of the reasons this remained open for such a long time could be that the cipher uses irregular clocking to update its state registers. Hence it becomes difficult to determine the location of a randomly applied fault injected in either the R or S register by simply comparing the faulty and fault-free key-streams. The idea

Cipher	State size	Average # Faults
Trivium [109]	288	2
Grain v1 [120] (Described in Section 5.5)	160	≈ 10
MICKEY 2.0	200	$\approx 2^{14.7}$

TABLE 7.4: A summary of the best Fault Attacks reported against the hardware portfolio of eStream

explained in Theorem 7.4 and its implications are instrumental in mounting the attack. The total number of faults is indeed much higher when we compare it with the other two eStream hardware candidates. However, this seems natural as MICKEY 2.0 has more complex structure than Trivium or Grain v1. This is also important to point out that while Grain and Trivium are susceptible to DFA with very few faults when SAT solvers are exploited, such drastic results could not be attained for MICKEY 2.0.

Chapter 8

Improved Scan-Chain based Attacks and Related Countermeasures

Scan-chains are one of the most commonly-used DFT (Design for Testability) techniques. DFT refers to design techniques that add certain testability features to a micro-electronic hardware product design. However, the presence of scan-chains makes the device vulnerable to scan-based attacks from cryptographic point of view. Techniques to cryptanalyze stream ciphers like Trivium, with additional hardware for scan-chains, are already available in literature (Agrawal et. al. Indocrypt 2008 [11]). However, extending such ideas to more complicated stream ciphers like MICKEY 2.0, is not possible as the state update function used by MICKEY 2.0 is far more complex. In this chapter, we will describe a general strategy to perform a scan-chain based attack on MICKEY 2.0. Furthermore, we will look at the XOR-CHAIN based countermeasure that was proposed by Agrawal et. al. in Indocrypt 2008, to protect Trivium from such scan-based attacks. We will show that such an XOR-CHAIN based countermeasure is vulnerable to a SET attack. As an alternative, we propose a novel countermeasure that can protect scan-chains against such attacks.

8.1 Introduction

While manufacturing any hardware product, DFT techniques are design efforts that are specifically employed to ensure that a device is testable. Single scan-chains are one of the most popular and effective ways of providing testability to any hardware device. The objective of the scan-chain is to make testing easier by providing a way to set and observe

every flip-flop in an IC. Unlike the functional tests that check chip functionality, scan tests cover stuck-at-faults, caused by manufacturing problems. Physical manufacturing defects, such as

- silicon defects, photo-lithography defects, mask contamination
- process variation or defective oxide etc.

may lead to electrical defects such as shorts (bridging faults), opens, transistors stuck on open, changes in threshold voltage etc. which may lead to digital logic being stuck at either the 0 or 1 value at one or many of the flip-flops. It may also lead to slower transitions among the flip-flops causing delay faults which hamper the proper functioning of a cryptosystem.

Scan-chain testing can be done to check whether a chip is functioning normally or not. It provides the designer an easy way to ascertain whether the device has succumbed to the above mentioned defects or not. In this design methodology, all the flip-flops in the design are replaced with scan type flip-flops. The design is made controllable and observable by chaining all these flip-flops together and shifting test data in and out. Scan type flip-flop contains a multiplexer to select either a normal mode functioning or a scan mode functioning. By suitably altering the control value to the multiplexer, the chip can be used for normal or scan test mode of operation. After selecting scan-test mode, the user is able to input test patterns of his choice into the device and thereafter scan out the contents of all the flip-flops connected to the scan-chain. It therefore gives the following opportunities to the user:

1. **Controllability**: The ability to set the flip-flops to certain states or logic values.
2. **Observability**: The ability to observe the state of these flip-flops.

The flip side of this design paradigm is that this makes certain cryptosystems implemented with such scan-chains in hardware, vulnerable to scan-based side channel attacks. Scan-based attack is a semi-intrusive side channel attack that does not require the attacker to actively tamper with the functioning of the cryptosystem as in optical/laser fault attacks. The attacker takes advantage of the scan-chain already implemented in the device and stops the normal mode of operation of the cryptosystem at some suitably chosen time instant and scans out the content of all the flip-flops in the system. The flip-flops usually store the internal state bits of the cryptosystem, and if the adversary can deduce the correspondence between the individual bits of the scanned out vector and the internal state bits, this may be enough to break the system. However, ascertaining

such a correspondence is usually non-trivial and thus a fascinating area for cryptanalytic research. Scan-based attacks have already been reported against block ciphers like AES [135] and DES [134] and stream-ciphers like RC4 [121] and Trivium [11].

8.1.1 Our Results

In [11], the stream cipher Trivium [43] was successfully cryptanalyzed using scan-based attack. We will show that due to the complex structure of the state-update functions used in MICKEY 2.0, extending the attack of [11] to MICKEY 2.0 is impossible. Our contributions are therefore threefold:

1. We will propose a strategy to perform the scan-based attack on MICKEY 2.0 that is independent of any specific physical implementation of the cipher.
2. In [11], the XOR-CHAIN based countermeasure was suggested to protect cryptosystems from such attacks. We will show that such a countermeasure is vulnerable to the SET attack.
3. As an alternative, we provide a countermeasure for scan-chains that will thwart a SET or RESET attack. We will prove that incorporating such a countermeasure will still allow the designer to control and test the scan-chain.

The organization of the chapter is as follows. In Section 8.2 we will give some background on how scan-based attacks are mounted and carried out against cryptosystems. In Section 8.3, we will outline the details of the attack against MICKEY 2.0. In Section 8.4, we propose an attack against the XOR-CHAIN based countermeasure and show how MICKEY 2.0 can be attacked even in the presence of such a countermeasure. In Section 8.5 we will outline the proposed countermeasure and discuss its security features in detail. Section 8.6 concludes the chapter.

8.2 Scan-Chain Attack: Background and Preliminaries

A scan-based test involves construction of one or more scan-chains in a chip by connecting the internal registers and flip-flops of a device and by making either ends of the chain available to the boundary scan interface, via the SCAN-IN and SCAN-OUT ports (See Fig 8.1). During testing, test vectors can be scanned in serially through the SCAN-IN pin. The contents of the chain can also be scanned out in a serial manner through the SCAN-OUT pin. During the testing phase, all flip-flops are disconnected from the combinatorial digital logic of the device and connected in single or multiple connected chains.

As shown in Figure 8.2, this is done by placing a multiplexer infront of the D input of each flip-flop controlled by the SCAN-ENABLE signal. In normal mode of operation, the SCAN-ENABLE signal is set to 0, so the flip-flop accepts the D-input and the device behaves normally. In test mode, the SCAN-ENABLE signal is set to 1 and in this event the flip-flop accepts the SCAN-IN input. Scan-chains are automatically inserted into the design by a Computer aided synthesis tool. The chain is usually organized according to the physical positions of the flip-flops. One may note that any arbitrary pattern can be given as input into the scan-chain, and the state of every flip-flop can be read out. We will now discuss some salient features related to scan-chains and state clearly the

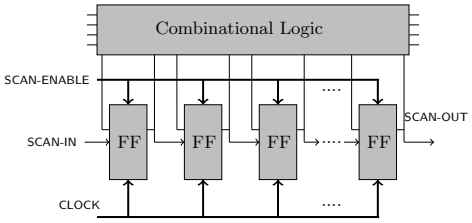


FIGURE 8.1: Diagram of a Scan-chain

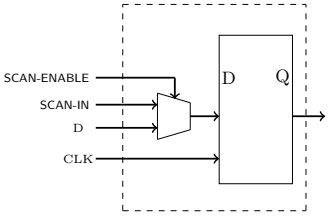


FIGURE 8.2: Scan-enabled D FF

cryptographic model that is employed to mount a scan-based attack.

⋆ **Ability to assert the SCAN-ENABLE signal :** We assume that the adversary is able to control the SCAN-ENABLE input to the device i.e. he has the ability to run the device under normal mode or test mode interchangeably. It is also reasonable to assume that the adversary can time the changing of the SCAN-ENABLE signal in synchronization with the system clock signal. In other words, he is able to stop the normal mode of operation of the device after any given number of clock rounds, drive the device into test mode and scan out the contents of the flip-flops of the scan-chain serially, via the SCAN-OUT port.

⋆ **Knowledge of the cryptosystem used in the device :** We assume that the adversary knows the high-level algorithmic design details of the cryptosystem implemented in the device (in this case MICKEY 2.0). Although, the adversary is expected to know the general hardware structure of the cryptosystem, he does not know exactly how many flip-flops have been used in the design. For example, a typical implementation of MICKEY 2.0 ([42, 64, 69, 92, 117]) is expected to have around 211 flip-flops (100 each for the R and S registers, 7 for the counter register, and one each for INPUT_BIT_R, INPUT_BIT_S, CONTROL_BIT_R, CONTROL_BIT_S). But different implementations of the cipher may use extra flip-flops as per the requirements of the designer. In this sense, the adversary must be able

to come up with an attack strategy that is independent of any specific hardware implementation of the cipher.

⋆ **Ability to manipulate the public variables :** We assume that the adversary is able to operate the cryptosystem using any public variable of his choice. In this case, this implies that he can run the cipher using any IV of his choice, while the Key remains secret. The Secret Key is usually stored in the memory (RAM) which is not connected to the scan-chain.

⋆ **Knowledge of Scan-chain structure :** Here, we assume that the adversary does not have any knowledge of the structure of the scan-chain that ties the flip-flops of the device together. The flip-flops in a scan-chain are not connected according to their positions in their respective registers. Rather, a Computer-aided tool optimizes the scan-chain according to the physical locations of the individual flip-flops. He also does not know the number of flip-flops in the scan-chain, but as shown in Section 8.3.1, finding this number is not difficult.

⋆ **Putting it all together :** The online attack procedure of the scan-based attack is very simple. The adversary lets the cryptosystem run in the normal mode for a fixed number of clock rounds. He then asserts the SCAN-ENABLE signal, which halts the normal operation mode, and scans out the content of the scan-chain serially via the SCAN-OUT port. He will get a scanned out vector $\mathbf{V}$, of the length of the scan-chain. This scanned out vector contains all the state bits of the cryptosystem at the clock round after which the normal mode of operation was halted. However, since the structure of the scan-chain is unknown to the adversary, he is unable to deduce the correspondence between the individual bits of $\mathbf{V}$ and individual state bits of the cryptosystem. For example, he knows with certainty that some element of $\mathbf{V}$ is equal to the 0^{th} bit of the register R at the round when the normal mode was halted, but he does not know which element. In other words, he is unaware of the permutation π between the scanned out vector $\mathbf{V}$ and the internal state of the cipher.

So any scan-based attack usually proceeds in two phases :

(a) Pre-processing stage → In this stage, the adversary performs various tests on the device to gain information about the structure of the scan-chain and deduce the structure of the permutation π. This is the stage that requires rigorous cryptanalysis. Once the permutation π has been ascertained, the adversary proceeds to the online stage.

(b) Online stage → The adversary lets the device get initialized with some unknown Key and IV, and halts the device at some suitable clock round (in the case of MICKEY 2.0, he stops the normal mode at the beginning of the PRGA). He

then scans out the content of the flip-flops in the scan-chain through the SCAN-OUT port and therefore gets the vector $\mathbf{V}$. Since the permutation π is now known to him, he is able to reconstruct the internal state of the cipher from $\mathbf{V}$ and this completes the attack.

8.3 Attacking MICKEY 2.0

The keystream generator makes use of two registers R and S (100 bits each). The registers are updated in a non-linear manner using the control variables: INPUT_BIT_R, INPUT_BIT_S, CONTROL_BIT_R, CONTROL_BIT_S. As referred to earlier, any implementation of the cipher contains flip-flops for the R, S registers and the 4 control variables. Furthermore, there must be 7 flip-flops for the counter register to keep track of the number of rounds in the Preclock stage. For more details please refer to [16].

In [11], a scan-based attack on Trivium was presented. As we go along we will show that the attack idea of [11] can not be extended to MICKEY-like ciphers whose state update functions are much more complex. The keystream production stage in MICKEY 2.0 is preceded by the three stages:- IV Loading, Key Loading and Preclock. Initially the R, S registers are initialized to the all zero state. Then at each clock round, a variable length IV $[iv_0, iv_1, \ldots, iv_{v-1}]$ and the 80 bit Key $[k_0, k_1, \ldots, k_{79}]$ is used to update the state by successively executing the function CLOCK_KG$(R, S, 1, iv_i)$, (for $i \rightarrow 0$ to $v-1$) and CLOCK_KG$(R, S, 1, k_i)$, (for $i \rightarrow 0$ to 79). The strategy of the adversary in our attack will be to operate the cipher in the IV loading stage using certain IV′s of his choice and then halt the normal mode by asserting the SCAN-ENABLE signal and read out the contents of the scan-chain. He will repeat this exercise multiple number of times. By observing the scanned out vector in each case, he will attempt to deduce the structure of the scan-chain.

8.3.1 Finding the length of the scan-chain

The attacker begins by resetting all the flip-flops of the scan-chain to zero and then asserts the SCAN-ENABLE signal. The first input to SCAN-IN port is set to 1. As is obvious from Figure 8.1, if there are n flip-flops in the chain, the scanned out vector will contain n zeros (which is the initial state of the scan-chain) followed by the single 1 which comes from the first input to the scan-chain. Thus the attacker can deduce the value of n easily.

8.3.2 Strategy to find the location of the counter bits

Initially, the task of the adversary is to ascertain the location of the bits of the counter register in the scanned out vector. In [11], the strategy to find the counter bits for Trivium was as follows (note that Trivium uses an 11-bit counter register). The attacker would initially RESET all the flip-flops in the scan-chain and run the cipher in the normal mode for $2^{10} - 1$ clock rounds. The structure of Trivium is such that if all the registers are initialized to the all-zero state, then it will continue to be in this state as long as the cipher runs. Hence, the only bits which will change are those in the flip-flops of the counter register which operates independently of the combinational logic of the cipher. After $2^{10} - 1$ rounds the 10 least significant bits of the counter become all 1s. If the adversary now asserts the SCAN-ENABLE signal and reads out the content of the scan-chain, he will observe exactly ten ones in the scanned-out vector. The bit locations of these 1s indicate the position of the ten LSBs of the counter register in the scan-chain. However, such an attack can not be extended to MICKEY 2.0. Initially, all the flip-flops in the R, S registers are set to 0. If the IV Loading stage runs even for a single clock round with the first IV bit equal to zero, the state of the S register evaluates to `0858041280104086 43BC42800`. This vector itself has 24 ones and so it is clear that the strategy of [11] can not be extended to MICKEY 2.0. The strategy that we propose is as follows. To find the location of the LSB of the counter the attacker will use such IVs whose length is of the form $l_0 = 2\alpha + 1$ i.e. an odd number. Whatever be the value of the IV, after l_0 rounds, the LSB of the counter register will always evaluate to 1. After l_0 rounds of IV loading, the attacker asserts the SCAN-ENABLE signal and reads out the contents of the scan-chain. The attacker does this for n_0 many IVs of odd length and performs the bitwise AND of each of the n_0 scanned out vectors. If n_0 is sufficiently large, all but one of the elements of this product vector becomes 0. Clearly, the only non-zero element in the product vector corresponds to the location of the LSB of the counter register.

To find the location of the next LSB, the attacker chooses IVs of length $l_1 = 4\alpha + 2$ or $4\alpha + 3$. It is clear that irrespective of the values of the IV, after l_1 rounds of IV loading, the second LSB of the counter register always evaluates to 1. The attacker repeats the above process with n_1 IVs of this form and as above, computes the bitwise AND of all the scanned out vectors. If n_1 is sufficiently large, all but one of the elements of this product vector becomes 0 and the only non-zero element in the product vector corresponds to the location of the second LSB of the counter register. The process can similarly be extended to find the location of all the other bits of the counter register. The above arguments have been formalized in Algorithm 8.1.

Input: The index of the counter LSB k
Output: The Set A_k of IVs which determine the location of the k^{th} LSB of counter register
Output: The location index β_k of the k^{th} LSB of counter register in the scan-chain

---

Let $\mathbf{P}$ be a vector of n elements (n is the length of the scan-chain)
$\mathbf{P} \leftarrow 1^n$;
$w \leftarrow 0$;
while $\| \mathbf{P} \| \neq 1$ **do**
  `/* ‖ P ‖ denotes the number of 1s in P */`
  Generate a random Initial Vector IV_w of length $l_k = 2^{k+1}\alpha + 2^k + r, \quad (0 \leq r < 2^k)$;
  Append IV_w to the Set A_k ;
  Reset cipher and perform IV Loading with IV_w ;
  Assert the SCAN-ENABLE signal and read the scanned out vector $\mathbf{V}$.
  $\mathbf{P} = \mathbf{P}\&\mathbf{V}$ (& denotes bitwise AND);
  $w \leftarrow w + 1$;
end
for $i = 1$ TO n **do**
  if $\mathbf{P}[i] = 1$ **then**
    $\beta_k \leftarrow i$;
  end
end
Return A_k, β_k;

Algorithm 8.1: Algorithm to determine the location of counter bits

Algorithm 8.1 returns the location β_k of the k^{th} LSB of the counter register in the scan-chain. It also returns the IV set A_k which helps determine β_k. The values of A_k $\forall k \in [0, 6]$ for an implementation of MICKEY 2.0, that uses the 211 flip-flops given above, are included in the Table 8.1. Note that the IVs are of the form 0^i and the values of i are listed in the table.

k^{th} LSB of counter	Set A_k of IVs to find k^{th} LSB
0^{th} bit (LSB):	$3, 5, 7, 9, 11, 13, 15, 17$
1^{st} bit:	$3, 7, 11, 15, 19, 23, 27, 31$
2^{nd} bit:	$4, 12, 20, 28, 36, 44, 52, 60$
3^{rd} bit:	$8, 9, 10, 11, 12, 13, 14, 15$
4^{th} bit:	$16, 17, 18, 19, 20, 21, 22, 23$
5^{th} bit:	$32, 33, 34, 35, 36, 37, 38, 39$
6^{th} bit:	$64, 65, 66, 67, 68, 69, 70, 71$

TABLE 8.1: Set A_k of IVs which can determine the location of the k^{th} LSB of counter register

8.3.3 Strategy to find the location of the other internal state bits

We will briefly recall the strategy of [11] to find the location of the state bits of Trivium. Note that Trivium has an internal register of 288 bits. The 80 bit Secret Key is directly loaded on to the first 80 bits of the register. The 80 bit IV is loaded on to the 94^{th} to the 173^{rd} bits of the register. The remaining 128 bits are loaded with a fixed initialization constant. In order to find the location of the 2^{nd} bit of the state register (say), the adversary initializes the cipher with the Key 8000 0000 0000 0000 and the all zero IV. The remaining bits of the register are initialized to all zero. As a result, before the initialization stage begins, only the first register bit holds the value 1 and the rest 0. The cipher is run in normal mode for one clock round, and then the SCAN-ENABLE signal is asserted and the contents of the chain are scanned out. The state update of Trivium is such that after the first initialization round, only the 2^{nd} register bit has the value 1 and the rest 0. So the scanned out vector that the adversary obtains has two 1s: one in the location corresponding to the LSB of the counter register and the other in the position corresponding to the 2^{nd} bit of the state register. Since the attacker already knows the location of the counter bits, he can easily deduce the location of the 2^{nd} state register bit. This approach cannot be extended to MICKEY 2.0 for two reasons:

- As we have already seen, the state update function of MICKEY 2.0 is way too complex.
- Unlike Trivium, MICKEY 2.0 does not allow direct loading of Key and IV bits on to the state register. As mentioned earlier, initially the R, S registers are initialized to the all zero state. Then a variable length IV $[iv_0, iv_1, \ldots, iv_{v-1}]$ and the 80 bit Key $[k_0, k_1, \ldots, k_{79}]$ is used to update the state by successively executing CLOCK_KG$(R, S, 1, iv_i)$, (for $i \rightarrow 0$ to $v-1$) and thereafter the routine CLOCK_KG$(R, S, 1, k_i)$, (for $i \rightarrow 0$ to 79).

The strategy that we propose is as follows. To find the location of the i^{th} bit of the register R (say), the attacker will use such Initial Vectors $\text{IV}_w = [iv_0, iv_1, \ldots, iv_{v-1}]$ which after the IV Loading stage i.e. executing the routine CLOCK_KG$(R, S, 1, iv_i)$, (for $i \rightarrow 0$ to $v-1$) successively, leaves the i^{th} bit of the register R at value 1. Since the CLOCK_KG routine is known publicly, the attacker can easily select such IVs by random selection. By standard randomness assumptions, one out of every two randomly selected IVs will result in the i^{th} bit of R being equal to 1 at the end of the IV Loading stage. The rest of the attack is same as before. After IV Loading, the attacker asserts the SCAN-ENABLE signal and reads out the scanned-out vector $\mathbf{V}$. The attacker does this for m_i many IVs that result in the i^{th} bit of R being 1 and performs the bitwise AND of each of the m_i scanned out vectors. If m_i is sufficiently large, all but one of the elements of this product

vector becomes 0. Clearly, the only non-zero element in the product vector corresponds to the location of the i^{th} bit of R.

Input: The internal state bit χ whose location in the scan-chain is to be determined
Input: The set of index locations scan-chain **T** whose correspondence has been determined
Output: The Set A_χ of IVs which determine the location of the state bit χ
Output: The location index β_χ of the state bit χ in the scan-chain

---

Let **P** be a vector of n elements (n is the length of the scan-chain)
$\mathbf{P} \leftarrow 1^n$;
$w \leftarrow 0$;
for $\forall\ i \in \mathbf{T}$ **do**
  $\mathbf{P}[i] = 0$;
end
while $\| \mathbf{P} \| \neq 1$ **do**
  `/* ||P|| denotes the number of 1s in P */`
  Generate a random Initial Vector $\text{IV}_w = [iv_0, iv_1, \ldots, iv_{v-1}]$;
  Set $R \leftarrow \mathbf{0}$, $S \leftarrow \mathbf{0}$;
  Execute CLOCK_KG$(R, S, 1, iv_i)$, (for $i \rightarrow 0$ to $v-1$);
  if *The state bit* $\chi = 1$ **then**
    Append IV_w to the Set A_χ ;
    Reset cipher and perform IV Loading with IV_w ;
    Assert the SCAN-ENABLE signal and read the scanned out vector **V**.
    $\mathbf{P} = \mathbf{P}\&\mathbf{V}$ (& denotes bitwise AND);
  end
  $w \leftarrow w + 1$;
end
for $i = 1$ TO n **do**
  if $\mathbf{P}[i] = 1$ **then**
    $\beta_\chi \leftarrow i$;
  end
end
Append β_χ to the set **T**;
Return A_χ, β_χ;

Algorithm 8.2: Algorithm to determine location of the state bit χ

Experimentally, the values of n_i, m_i for an implementation of MICKEY 2.0 with 211 flip-flops has been found to be around 8 to 20. To speed up the process, the attacker can omit those bit locations of the scanned out vector **V** whose correspondence to some flip-flop in the design have already been found out. For example, the attacker can omit the bits of **V** which correspond to the counter register. He may also omit any other bits

i	Set A_χ of IVs to find $R[i]$	Set A_χ of IVs to find $S[i]$
0	3, 10, 11, 12, 14, 16, 17, 19, 20, 21, 22, 25, 26, 29, 30, 31, 34, 35, 36, 37	13, 16, 19, 22, 23, 25, 27, 28, 29, 33, 34, 36, 37, 38, 40
1	3, 4, 10, 12, 13, 16, 18, 22, 23, 25, 27, 28, 29, 31, 32, 34, 36	13, 14, 16, 17, 19, 20, 21, 22, 23, 25, 26, 27, 30, 31, 33, 34, 36, 39
2	4, 5, 6, 7, 11, 12, 14, 15, 17, 19, 23, 24, 26, 28, 29, 32, 33, 34, 37, 38, 39	13, 14, 16, 17, 19, 21, 24, 25, 26, 27, 29, 31, 32, 35, 36, 37, 38
3	3, 5, 7, 8, 10, 11, 14, 15, 18, 21, 22, 24, 27, 28, 30, 31, 33, 34, 39, 40	14, 17, 19, 22, 23, 26, 28, 30, 32, 33, 36, 39
4	3, 4, 6, 7, 8, 9, 10, 12, 14, 15, 20, 21, 28, 31, 32, 35, 36, 37, 40	1, 4, 7, 10, 16, 18, 21, 22, 23, 24, 26, 27, 28, 30, 31, 33, 35, 36, 37, 39

TABLE 8.2: The Set A_χ of IVs which can determine the location of the bits of Registers R, S. (The IVs are of the form 0^i. The values of i are listed in the table.)

of $\mathbf{V}$ whose correspondence with some internal state bit have already been determined. The arguments have been formalized in Algorithm 8.2.

Algorithm 8.2 returns the location β_χ of the state bit χ in the scan-chain. It also returns the IV set A_χ which helps determine β_χ. The values of A_χ for the first 5 bits of the registers R, S, for an implementation of MICKEY 2.0 that use the 211 flip-flops given above, are included in Table 8.2.

8.4 Attacking the XOR-CHAIN Countermeasure Scheme

The Flipped-Scan countermeasure technique to protect scan-chains was proposed in the work [121]. This involved placing inverters at random points in the scan-chain. Security stemmed from the fact that an adversary could not guess the number and positions of the inverters. This technique was cryptanalyzed in [11] using a RESET attack. It was shown that if all flip-flops in the scan-chain are initially RESET, then the positions of the inverters can be completely determined by the $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions in the scanned-out vector. As an alternative, the XOR-CHAIN based countermeasure was proposed in [11]. The technique involves placing XOR gates at random points of the chain as described in Figure 8.3. Security again stems from the fact that an adversary is unable to guess the number and positions of the XOR gates.

Notations

We assume that there are n flip-flops in the scan-chain. The state of the i^{th} flip-flop $(1 \leq i \leq n)$ at clock round t $(t \geq 0)$ after the SCAN-ENABLE signal is asserted, is given by the symbol S_i^t. The τ^{th} $(\tau \geq 1)$ round input to the scan-chain is given as x_τ.

Similarly the τ^{th} round output of the scan-chain is denoted as y_τ. We also define the sequence $a_i,\ (1 \leq i \leq n)$ over GF(2) as follows. If there is an XOR gate before the i^{th} flip-flop then $a_i = 1$ else $a_i = 0$. The goal of the attacker is to determine the value of the vector $[a_1, a_2, \ldots, a_n]$.

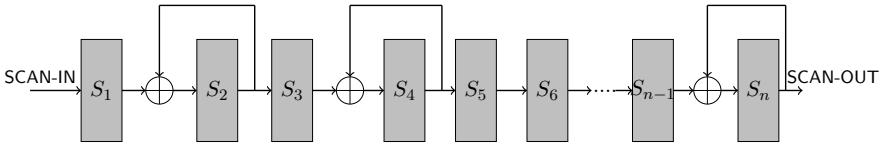


FIGURE 8.3: Diagram of the XOR-CHAIN scheme proposed in [11]

8.4.1 The SET attack on the XOR-CHAIN structure

Most standard VLSI designs provide a GLOBAL SET/RESET (GSR) pin to initialize all flip-flops to some known state INIT during configuration [62]. The INIT value of flip-flop primitives like FDP, FDPE, FDS etc. in the Xilinx Virtex 6 library [9] is 1 by default, i.e. these flip-flops are SET after configuration. Our strategy to attack the XOR-CHAIN would be to SET all the flip-flops in the chain and then assert the SCAN-ENABLE signal. We will prove in Theorem 8.1, that by observing the values of the scanned-out vector, the adversary will be able to determine the number and positions of the XOR gates in the chain.

Theorem 8.1. *If the scan-chain is initially* SET *i.e.* $S_i^0 = 1,\ \forall i \in [1, n]$, *and* $x_1 = 1$, *then the value of the vector* $[a_1, a_2, \ldots, a_n]$ *can be determined efficiently by observing the output bits* $y_i,\ \forall i \in [1, n]$ *of the scanned-out vector.*

Proof. Define the symbol $S_0^t = x_{t+1}$. Due to the architecture of the scan-chain, the following equation holds for any $t > 0,\ 1 \leq i \leq n$:

$$S_i^t = S_{i-1}^{t-1} \oplus a_i \cdot S_i^{t-1} \tag{8.1}$$

We will give a general outline of the proof first and sort out the more technical details later. As can be seen from Figure 8.3, the first output bit y_1 is given by $y_1 = S_n^0 = 1$. The second output bit is given by

$$y_2 = S_n^1 = S_{n-1}^0 \oplus a_n \cdot S_n^0 = 1 \oplus a_n.$$

The value of a_n is therefore given by $1 \oplus y_2$. Now look at the equation governing y_3.

$$\begin{aligned} y_3 = S_n^2 = S_{n-1}^1 \oplus a_n \cdot S_n^1 &= S_{n-2}^0 \oplus a_{n-1} \cdot S_{n-1}^0 \oplus a_n \cdot (S_{n-1}^0 \oplus a_n \cdot S_n^0) \\ &= S_{n-2}^0 \oplus (a_{n-1} \oplus a_n) \cdot S_{n-1}^0 \oplus a_n \cdot S_n^0 \\ &= 1 \oplus a_{n-1}. \end{aligned}$$

The value of a_{n-1} is therefore given by $1 \oplus y_3$. Now look at the equation for y_4.

$$\begin{aligned} y_4 =& S_n^3 = S_{n-1}^2 \oplus a_n \cdot S_n^2 \\ =& S_{n-2}^1 \oplus a_{n-1} \cdot S_{n-1}^1 \oplus a_n \cdot \left(S_{n-2}^0 \oplus (a_{n-1} \oplus a_n) \cdot S_{n-1}^0 \oplus a_n \cdot S_n^0\right) \\ =& S_{n-3}^0 \oplus (a_{n-2} \oplus a_{n-1} \oplus a_n) \cdot S_{n-2}^0 \oplus (a_{n-1} \oplus a_n \oplus a_n a_{n-1}) \cdot S_{n-1}^0 \oplus a_n \cdot S_n^0 \\ =& 1 \oplus a_{n-2} \oplus a_n \oplus a_n a_{n-1}. \end{aligned}$$

Since the values of a_n, a_{n-1} are known, the value of a_{n-2} may be calculated as $1 \oplus y_4 \oplus a_n \oplus a_n a_{n-1}$. Proceeding in this manner we can deduce a_{n-3} from y_5, a_{n-4} from y_6, ..., a_{n-i+3} from y_{i-1}. At the i^{th} stage, y_i, $(1 \leq i \leq n+1)$ can be written as

$$y_i = S_{n-i+1}^0 \oplus b_{i,n-i+2} \cdot S_{n-i+2}^0 \oplus b_{i,n-i+3} \cdot S_{n-i+3}^0 \oplus \cdots \oplus b_{i,n} \cdot S_n^0 \tag{8.2}$$

It can be shown that **(i)** $b_{i,n-i+2} = a_n \oplus a_{n-1} \oplus \cdots \oplus a_{n-i+2}$, **(ii)** $b_{i,n-i+3}, b_{i,n-i+4}, \ldots, b_{i,n}$ are functions of $a_n, a_{n-1}, \ldots a_{n-i+3}$ only. Since $a_n, a_{n-1}, \ldots a_{n-i+3}$ are already known, a_{n-i+2} equals

$$1 \oplus y_i \oplus a_n \oplus a_{n-1} \oplus \cdots \oplus a_{n-i+3} \oplus b_{i,n-i+3} \oplus \cdots \oplus b_{i,n}.$$

The values of $a_n, a_{n-1}, \ldots, a_1$ may be found out in this manner. Now all that remains to be shown are the proofs of **(i), (ii)**. We will proceed by mathematical induction. Let $P(i)$ be the proposition defined as follows. For all $k \in [1, n]$, (and $k - i \geq 0$)

$$S_k^i = S_{k-i}^0 \oplus c_{k-i+1} \cdot S_{k-i+1}^0 \oplus \ldots \oplus c_k \cdot S_k^0$$

then

A. $c_{k-i+1} = a_k \oplus a_{k-1} \oplus \cdots a_{k-i+1}$,

B. $c_{k-i+2}, \ldots, c_k$ are functions of $a_{k-i+2}, a_{k-i+3}, \ldots, a_k$ only.

For $i = 1$, $S_k^1 = S_{k-1}^0 \oplus a_k \cdot S_k^0$ and so $P(1)$ is true. Assume $P(i)$ is true for $i = 2, 3, \ldots, u$. For $i = u + 1$,

$$\begin{aligned} S_k^{u+1} &= S_{k-1}^u \oplus a_k \cdot S_k^u \\ &= S_{k-1-u}^0 \oplus (a_{k-1} \oplus a_{k-2} \oplus \cdots \oplus a_{k-u}) \cdot S_{k-u}^0 \oplus \eta_{k-u+1} \cdot S_{k-u+1}^0 \oplus \cdots \oplus \\ &\quad \eta_k \cdot S_k^0 \oplus a_k \cdot (S_{k-u}^0 \oplus \gamma_{k-u+1} \cdot S_{k-u+1}^0 \oplus \cdots \oplus \gamma_k \cdot S_k^0) \\ &= S_{k-u-1}^0 \oplus (a_k \oplus \cdots \oplus a_{k-u}) \cdot S_{k-u}^0 \oplus (\eta_{k-u+1} \oplus a_k \cdot \gamma_{k-u+1}) \cdot S_{k-u+1}^0 \\ &\quad \oplus \cdots \oplus (\eta_k \oplus a_k \cdot \gamma_k) \cdot S_k^0. \end{aligned}$$

This proves **A**. By induction hypothesis on $i = u$, $\eta_{k-u+1}, \ldots, \eta_k, \gamma_{k-u+1}, \ldots, \gamma_k$ are functions of $a_{k-u+1}, \ldots, a_k$ only and so this proves **B** as well. It can be seen that **(i), (ii)** follow from **A, B**. □

8.4.2 Attacking MICKEY 2.0 in presence of XOR-CHAIN

One of the main difficulties of applying Algorithms 8.1 and 8.2, to any implementation of MICKEY 2.0 protected by an XOR-CHAIN structure is that the scanned-out vector will no longer represent the state of the scan-chain before the SCAN-ENABLE signal was asserted. Because of the random placement of the XOR gates in the chain, the scanned out vector $\mathbf{V} = [y_1, y_2, y_3, \ldots y_n]^T$ is a linear combination of the state of the scan-chain $\mathbf{S} = [S_1^0,\ S_2^0,\ S_3^0,\ \ldots,\ S_n^0]^T$. From Equation (8.2), the relation between $\mathbf{V}$ and $\mathbf{S}$ is given as :

$$\begin{aligned} S_n^0 &= y_1 \\ S_{n-1}^0 \oplus b_{1,n} \cdot S_n^0 &= y_2 \\ S_{n-2}^0 \oplus b_{2,n-1} \cdot S_{n-1}^0 \oplus b_{2,n} \cdot S_n^0 &= y_3 \\ &\vdots \\ S_1^0 \oplus b_{n,2} \cdot S_2^0 \oplus \cdots \oplus b_{n,n-2} \cdot S_{n-2}^0 \oplus b_{n,n-1} \cdot S_{n-1}^0 \oplus b_{n,n} \cdot S_n^0 &= y_n \end{aligned}$$

In matrix form these equations may be written as $\mathcal{B} \cdot \mathbf{S} = \mathbf{V}$. Once the attacker has determined the values of $[a_1, a_2, \ldots, a_n]$ he can determine the values of $b_{i,j}$ $\forall i, j$ and hence the matrix $\mathcal{B}$. Now we would like to point out that $\mathcal{B}$ is invertible over $GF(2)$. Clearly, $\mathcal{B}$ is a lower anti-triangular matrix with all the elements in the anti-diagonal equal to 1. Therefore, it follows that $Det(\mathcal{B}) = 1$, and hence the result. The attacker can now deduce the state of the scan-chain before the assertion of the SCAN-ENABLE signal, by computing $\mathbf{S} = \mathcal{B}^{-1} \cdot \mathbf{V}$. The adversary can now apply Algorithms 8.1 and 8.2 to the vector $\mathbf{S}$.

8.5 Securing the Scan-Chain: Using the Double Feedback XOR-CHAIN

Our motivation was to find a structure that would resist the SET and RESET attacks. We found that a few simple tweaks to the XOR-CHAIN structure was sufficient to secure the chain from the aforementioned attacks. The structure we propose is this: We retain the idea of placing XOR gates at random points in the scan-chain. In the original proposal, if a_i was 1, the output of the i^{th} flip-flop was fed back to the XOR gate placed before it. In the structure that we propose, if a_i is 1 ($\forall\ i \in [1, n-1]$), then the output of the i^{th} and the $(i+1)^{th}$ flip-flop would be fed back to the XOR gate placed infront of the i^{th} flip-flop (see Figure 8.4). For $i = n$, (i.e. for the last flip-flop in the chain) we keep $a_n = 0$. We call this the "Double Feedback XOR-CHAIN" structure. We will first prove that such a structure can be used to test the scan-chain efficiently. We will also prove that such a structure would resist the SET and RESET attacks.

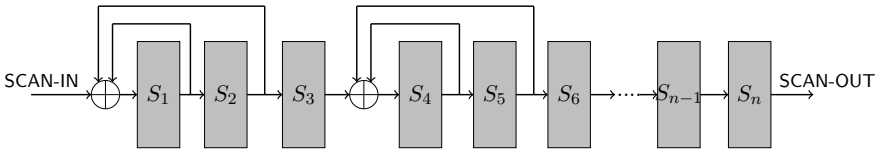


FIGURE 8.4: Double Feedback XOR-CHAIN

8.5.1 Testability

Because of the structure of the Double Feedback XOR-CHAIN, the flip-flops are updated by the following recursive equation (See Figure 8.4).

$$S_i^t = \begin{cases} S_{i-1}^{t-1} \oplus a_i \cdot (S_i^{t-1} \oplus S_{i+1}^{t-1}), & \text{if} \quad 1 < i < n, \\ S_{i-1}^{t-1}, & \text{if} \quad i = n, \\ x_t \oplus a_i \cdot (S_i^{t-1} \oplus S_{i+1}^{t-1}), & \text{if} \quad i = 1. \end{cases} \tag{8.3}$$

Let $X = [x_1, x_2, \ldots, x_n]$ be the inputs in the first n clock rounds, to the scan-chain after the assertion of SCAN-ENABLE signal. In the next n rounds, we read out the vector $Y = [y_{n+1}, y_{n+2}, \ldots, y_{2n}]$ from the SCAN-OUT pin. A necessary and sufficient condition to be able to use Double Feedback XOR-CHAIN structure for testing purposes is that the function mapping $X \rightarrow Y$ must be a bijection [11]. We will prove the bijection in two steps. Denote by $\mathcal{S}_t = [S_1^t, S_2^t, \ldots, S_n^t]^T$ the state of the scan-chain at the t^{th} clock round. We will first prove that the map between $X \rightarrow \mathcal{S}_n$ is a bijection, and then we

prove that the map between $\mathcal{S}_n \rightarrow Y$ is also a bijection. First we note that Equation (8.3) can be written in matrix form as follows:

$$\begin{pmatrix} S_1^t \\ S_2^t \\ S_3^t \\ S_4^t \\ \vdots \\ S_n^t \end{pmatrix} = \begin{pmatrix} a_1 & a_1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 1 & a_2 & a_2 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & a_3 & a_3 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & a_4 & a_4 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix} \begin{pmatrix} S_1^{t-1} \\ S_2^{t-1} \\ S_3^{t-1} \\ S_4^{t-1} \\ \vdots \\ S_n^{t-1} \end{pmatrix} \bigoplus \begin{pmatrix} x_t \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

We can write this in compact form $\mathcal{S}_t = \mathcal{A} \cdot \mathcal{S}_{t-1} \oplus X_t$. Here $\mathcal{A}$ is the tridiagonal matrix defined above and $X_t = [x_t, 0, 0, \ldots, 0]^T$. Before we prove the bijection we will look at a few useful results regarding the structure of $\mathcal{A}$ and its powers.

Lemma 8.2. *Consider the elements in the first column of $\mathcal{A}^p$, $(1 \leq p \leq n-1)$. It can be shown that $\mathcal{A}^p(p+1,1) = 1$ and $\mathcal{A}^p(i,1) = 0$ for all $i \in [p+2, n]$. ($\mathcal{M}(i,j)$ denotes the element in the i^{th} row and j^{th} column of the matrix $\mathcal{M}$)*

Proof. The proof is by mathematical induction. The statement is obviously true for $p = 1$. We assume that the statement is true for $p = 2, 3, \ldots, p_0$. Now $\mathcal{A}^{p_0+1}(p_0+2, 1)$ is given by

$$\mathcal{A}^{p_0+1}(p_0+2,1) = \bigoplus_{k=1}^{n} \mathcal{A}(p_0+2,k) \cdot \mathcal{A}^{p_0}(k,1) = \mathcal{A}(p_0+2, p_0+1) \cdot \mathcal{A}^{p_0}(p_0+1,1) = 1.$$

Also the expansion for $\mathcal{A}^{p_0+1}(i,1)$ for any $i \in [p_0+3, n]$ is given by

$$\mathcal{A}^{p_0+1}(i,1) = \bigoplus_{k=1}^{n} \mathcal{A}(i,k) \cdot \mathcal{A}^{p_0}(k,1) = 0.$$

This completes the proof. □

Lemma 8.3. *Consider the elements in the last row of $\mathcal{A}^p$, $(1 \leq p \leq n-1)$. It can be shown that $\mathcal{A}^p(n, n-p) = 1$ and $\mathcal{A}^p(n,j) = 0$ for all $j \in [1, n-p-1]$.*

Proof. The proof is similar to that of the previous lemma and we will again proceed by induction. The statement is obviously true for $p = 1$. We assume that the statement is true for $p = 2, 3, \ldots, p_0$. Now $\mathcal{A}^{p_0+1}(n, n-p_0-1)$ is given by

$$\begin{aligned} \mathcal{A}^{p_0+1}(n, n-p_0-1) &= \bigoplus_{k=1}^{n} \mathcal{A}^{p_0}(n,k) \cdot \mathcal{A}(k, n-p_0-1) \\ &= \mathcal{A}^{p_0}(n, n-p_0) \cdot \mathcal{A}(n-p_0, n-p_0-1) = 1. \end{aligned}$$

Also the expansion for $\mathcal{A}^{p_0+1}(n,j)$ for any $j \in [1, n-p_0-2]$ is given by

$$\mathcal{A}^{p_0+1}(n,j) = \bigoplus_{k=1}^{n} \mathcal{A}^{p_0}(n,k) \cdot \mathcal{A}(k,j) = 0.$$

This completes the proof. □

Lemma 8.4. *Let* $\mathbf{e}_1 = [1,0,0,\ldots,0]^T$. *Then, the first columns of* $\mathcal{A}^p$, $(1 \leq p \leq n-1)$ *and* I_n *i.e.*

$$I_n \cdot \mathbf{e}_1, \quad \mathcal{A} \cdot \mathbf{e}_1, \quad \mathcal{A}^2 \cdot \mathbf{e}_1, \quad \ldots \quad , \quad \mathcal{A}^{n-1} \cdot \mathbf{e}_1$$

are linearly independent over GF(2). (I_n *is the* $n \times n$ *identity matrix.)*

Proof. If possible let

$$\mathcal{G} = \alpha_0 \cdot I_n \cdot \mathbf{e}_1 \oplus \alpha_1 \cdot \mathcal{A} \cdot \mathbf{e}_1 \oplus \cdots \oplus \alpha_{n-1} \cdot \mathcal{A}^{n-1} \cdot \mathbf{e}_1 = \mathbf{0}, \tag{8.4}$$

for some $\alpha_i \in GF(2)$. From Lemma 8.2, it can be seen that the last element of the column vectors $I_n \cdot \mathbf{e}_1$ and $\mathcal{A}^p \cdot \mathbf{e}_1$ (for $1 \leq p \leq n-2$) are all 0. The last element of $\mathcal{A}^{n-1} \cdot \mathbf{e}_1$ is however 1. Therefore, from Equation (8.4), the last element of $\mathcal{G}$ is α_n which has to be 0 if $\mathcal{G} = \mathbf{0}$. Once $\alpha_n = 0$, by similar arguments, it can be deduced from Lemma 8.2, that the second last element of $\mathcal{G}$ is α_{n-1}. Thus α_{n-2} has to be 0 for Equation (8.4) to hold. Proceeding in this manner we can prove that $\alpha_{n-3} = \alpha_{n-2} = \cdots = \alpha_0 = 0$. □

Lemma 8.5. *Let* $\mathbf{e}_n = [0,0,0,\ldots,1]$. *Then, the last rows of* $\mathcal{A}^p$, $(1 \leq p \leq n-1)$ *and* I_n *i.e.*

$$\mathbf{e}_n \cdot I_n, \quad \mathbf{e}_n \cdot \mathcal{A}, \quad \mathbf{e}_n \cdot \mathcal{A}^2, \quad \ldots \quad , \quad \mathbf{e}_n \cdot \mathcal{A}^{n-1}$$

are linearly independent over GF(2).

Proof. The proof is exactly same as the one for Lemma 8.4. We make use of Lemma 8.3 to show that if $\beta_0 \cdot \mathbf{e}_n \cdot I_n \oplus \beta_1 \cdot \mathbf{e}_n \cdot \mathcal{A} \oplus \beta_2 \cdot \mathbf{e}_n \cdot \mathcal{A}^2 \oplus \ldots \oplus \beta_{n-1} \cdot \mathbf{e}_n \cdot \mathcal{A}^{n-1} = \mathbf{0}$, then $\beta_i = 0$ for all $i \in [0, n-1]$. □

Theorem 8.6. *The function mapping the first* n *inputs to the scan-chain given by* $X = [x_1, x_2, \ldots, x_n]$ *to the state* $\mathcal{S}_n$ *of the scan-chain is a bijection.*

Proof. We have already shown that the successive state vectors $\mathcal{S}_t$ of the scan-chain are related by the equation $\mathcal{S}_t = \mathcal{A} \cdot \mathcal{S}_{t-1} \oplus X_t$, $\forall t > 0$ where $X_t = [x_t, 0, 0, \ldots, 0]^T$. Combining these equations for $t = 1, 2, \ldots n$ we get

$$\mathcal{S}_n = \mathcal{A}^n \cdot \mathcal{S}_0 \oplus \mathcal{A}^{n-1} \cdot X_1 \oplus \mathcal{A}^{n-2} \cdot X_2 \oplus \cdots \oplus \mathcal{A} \cdot X_{n-1} \oplus X_n. \tag{8.5}$$

The scan-chain is usually RESET before the SCAN-ENABLE signal is asserted, and so $\mathcal{S}_0 = \mathbf{0}$ and therefore we have

$$\begin{aligned}\mathcal{S}_n &= \mathcal{A}^{n-1} \cdot X_1 \oplus \mathcal{A}^{n-2} \cdot X_2 \oplus \cdots \oplus \mathcal{A} \cdot X_{n-1} \oplus X_n \\ &= x_1 \cdot \mathcal{A}^{n-1} \cdot \mathbf{e}_1 \oplus x_2 \cdot \mathcal{A}^{n-2} \cdot \mathbf{e}_1 \oplus \cdots \oplus x_{n-1} \cdot \mathcal{A} \cdot \mathbf{e}_1 \oplus x_n \cdot I_n \cdot \mathbf{e}_1.\end{aligned}$$

Now, if possible let $X' = [x'_1, x'_2, \ldots, x'_n] \neq X$ be a vector that leads to the same value of $\mathcal{S}_n$. So we have,

$$\mathcal{S}_n = x'_1 \cdot \mathcal{A}^{n-1} \cdot \mathbf{e}_1 \oplus x'_2 \cdot \mathcal{A}^{n-2} \cdot \mathbf{e}_1 \oplus \cdots \oplus x'_{n-1} \cdot \mathcal{A} \cdot \mathbf{e}_1 \oplus x'_n \cdot I_n \cdot \mathbf{e}_1.$$

Adding the equations we get

$$\mathbf{0} = (x_1 \oplus x'_1) \cdot \mathcal{A}^{n-1} \cdot \mathbf{e}_1 \oplus (x_2 \oplus x'_2) \cdot \mathcal{A}^{n-2} \cdot \mathbf{e}_1 \oplus \cdots \oplus (x_n \oplus x'_n) \cdot I_n \cdot \mathbf{e}_1.$$

By Lemma 8.4, $I_n \cdot \mathbf{e}_1$, $\mathcal{A} \cdot \mathbf{e}_1$, $\mathcal{A}^2 \cdot \mathbf{e}_1$, $\ldots$, $\mathcal{A}^{n-1} \cdot \mathbf{e}_1$ are linearly independent, so we must have $x_i = x'_i$, $\forall i \in [1, n]$. Hence $X = X'$ and so the function mapping $X \to \mathcal{S}$ is certainly an injection. Also both the domain and range of this map is the vector space $GF(2)^n$, which proves that the function is a bijection. □

Theorem 8.7. *The function mapping the state $\mathcal{S}_n$ of the scan-chain to the output vector $Y = [y_{n+1}, y_{n+2}, y_{n+3}, \ldots, y_{2n}]$ is a bijection. Therefore the map between $X \to Y$ is also a bijection.*

Proof. We have $y_{n+1} = S_n^n = \mathbf{e}_n \cdot \mathcal{S}_n$. Similarly, $y_{n+2} = S_n^{n+1} = \mathbf{e}_n \cdot \mathcal{S}_{n+1} = \mathbf{e}_n \cdot (\mathcal{A} \cdot \mathcal{S}_n \oplus X_{n+1}) = \mathbf{e}_n \cdot \mathcal{A} \cdot \mathcal{S}_n$. Generalizing in this manner, we have for all $i \in [1, n]$,

$$\begin{aligned}y_{n+i} = S_n^{n+i-1} &= \mathbf{e}_n \cdot \mathcal{S}_{n+i-1} = \mathbf{e}_n \cdot (\mathcal{A}^{i-1} \cdot \mathcal{S}_n \oplus \mathcal{A}^{i-2} \cdot X_{n+1} \oplus \cdots \oplus X_{n+i-1}) \\ &= \mathbf{e}_n \cdot (\mathcal{A}^{i-1} \cdot \mathcal{S}_n \oplus x_{n+1} \cdot \mathcal{A}^{i-2} \cdot \mathbf{e}_1 \oplus \cdots \oplus x_{n+i-1} \cdot I_n \cdot \mathbf{e}_1)\end{aligned}$$

By Lemma 8.2, the last element of the column vectors $\mathcal{A}^{i-2} \cdot \mathbf{e}_1, \mathcal{A}^{i-3} \cdot \mathbf{e}_1, \ldots, I_n \cdot \mathbf{e}_1$ are all 0 and so their dot product with $\mathbf{e}_n$ will also be 0. And so we have $y_{n+i} = \mathbf{e}_n \cdot \mathcal{A}^{i-1} \cdot \mathcal{S}_n$, $\forall i \in [1, n]$. We can therefore write

$$Y^T = \begin{pmatrix} \mathbf{e}_n \cdot I_n \\ \mathbf{e}_n \cdot \mathcal{A} \\ \vdots \\ \mathbf{e}_n \cdot \mathcal{A}^{n-1} \end{pmatrix} \cdot \mathcal{S}_n = \mathcal{C} \cdot \mathcal{S}_n.$$

By Lemma 8.5, the rows of $\mathcal{C}$ are linearly independent and so $\mathcal{C}$ is invertible. This proves that the function mapping $\mathcal{S}_n \rightarrow Y$ is a bijection. Combining this result with Theorem 8.6, we can say that the function mapping $X \rightarrow Y$ is also a bijection. □

8.5.2 Resistance against SET and RESET attacks

In a Double Feedback XOR-CHAIN structure, it can be shown that if the scan-chain is initially RESET then the output in the first n rounds will be all 0. Similarly, if the chain is initially SET then the output in the first n rounds will be all 1. This is because, the initial contents of the scan-chain are simply shifted across each flip-flop regardless of whether there is a XOR gate placed infront of it. From Equation (8.3), we know that the value of $S_i^1 = S_{i-1}^0 \oplus a_i \cdot (S_i^0 \oplus S_{i+1}^0)$. If $a_i = 0$, i.e. if there is no XOR gate before the i^{th} flip-flop, then the content of the $(i-1)^{th}$ flip-flop is simply shifted to the i^{th} flip-flop in the next round. On the other hand, if $a_i = 1$, then we have $S_i^1 = S_{i-1}^0 \oplus S_i^0 \oplus S_{i+1}^0$. If the scan-chain is initially RESET i.e. $S_j^0 = 0,\ \forall j$, then the updated value of S_i^1 is also 0 and thus equal to S_{i-1}^0. If the scan-chain is initially SET i.e. $S_j^0 = 1,\ \forall j$, then the updated value of S_i^1 is also $1 \oplus 1 \oplus 1 = 1$ and thus also equal to S_{i-1}^0. In either event, the initial values of the flip-flops (whether all SET or RESET) are shifted across the chain and this is the output obtained in the first n rounds. Thus it is clear that in both attack scenarios no meaningful information about the vector $[a_1, a_2, \ldots, a_{n-1}]$ can be obtained from the first n scanned-out bits. Thus the attacker must look at the output bits $y_{n+1}, y_{n+2}, \ldots$ in hope of deducing $[a_1, a_2, \ldots, a_{n-1}]$. From equation (8.5), we have

$$\mathcal{S}_n = \mathcal{A}^n \cdot \mathcal{S}_0 \oplus \mathcal{A}^{n-1} \cdot X_1 \oplus \mathcal{A}^{n-2} \cdot X_2 \oplus \cdots \oplus \mathcal{A} \cdot X_{n-1} \oplus X_n.$$

Note that $\mathcal{S}_0$ is the all 1 or the all 0 column vector in the case of SET or RESET attack respectively. From Theorem 8.7, we have $Y^T = [y_{n+1}, y_{n+2}, \ldots, y_{2n}]^T = \mathcal{C} \cdot \mathcal{S}_n$. Note that, $\mathcal{A}^n, \mathcal{A}^{n-1}, \ldots, \mathcal{A}$ are dense $n \times n$ matrices over the polynomial ring $GF(2)[a_1, a_2, \ldots, a_{n-1}]$. The relation $Y^T = \mathcal{C} \cdot \mathcal{S}_n$ gives rise to n equations of degree $n-1$ each in $a_1, a_2, \ldots, a_{n-1}$. Solving such a system of equations does not seem to be easier than simply guessing the values $[a_1, a_2, \ldots, a_{n-1}]$ or breaking the cipher itself.

8.6 Conclusion

Although, scan-chains are a valuable tool to test an electrical device for faults, deployment of such scan-chains without having proper countermeasures in place can provide an attacker with a potent side channel to cryptanalyze the underlying cryptosystem. In this chapter we outline a strategy to perform a scan-based side channel attack on

MICKEY 2.0 that is independent of the actual implementation of the cipher. We then show that the XOR-CHAIN mechanism which was proposed in [11], is vulnerable to the SET attack. As a countermeasure we propose the Double Feedback XOR-CHAIN structure that resists the SET and RESET attacks. We have also presented detailed analysis, showing that such a structure may indeed be used for DFT purposes.

Chapter 9

Conclusion

In this chapter, we conclude the thesis. We revisit the chapters one-by-one to summarize the thesis. We mention the existing results and prior work (if any) in the direction. Most importantly, we present the crux of the chapters, that is our contributions, improvements and extensions to existing methods. Finally, we also discuss the future scope for research and potential open problems in respective field of study.

9.1 Summary of Technical Results

Chapter 1 provided the introduction to the thesis, while Chapter 2 covered some mathematical topics the reader should know before reading the thesis comfortably. The main technical results of the thesis are discussed in Chapters 3 to 7, and the highlights of these chapters are as follows.

Chapter 3: Analysis of RC4 Variants

RC4+ [100] and GGHN [68] were two of the stream cipher designs that were proposed as an alternative to the ubiquitous RC4 stream cipher after some weaknesses [102, 105] were reported against it. The idea was to provide a robust design that while comparable to RC4 in terms of speed in software, would also be free of the known weaknesses of RC4.

Prior to our work on the GGHN cipher, two distinguishing attacks [115, 132] requiring around 2^{33} and 2^{30} keystream bytes were reported against it. A fault attack against this cipher was also reported in [89]. In this thesis, we have shown two results with regards to the GGHN cipher:

1. First, we demonstrated the existence of numerous short cycles that occur during the evolution of the PRGA phase of the GGHN cipher.

2. Second, using the theory of Markov Chains, we have shown that a randomized variant of the GGHN cipher goes to the all zero state in around 2^{147} iterations. This is significant as once the the cipher reaches the all zero state it does not come out of it and thereafter, continues to produce only the all zero keystream sequence. For an ideal cipher, the all zero state should have taken around 2^{256} iterations to occur.

Next we turned our attention to the RC4+ stream cipher, against which no cryptanalytic advance had been reported prior to our work. Using the fact that the first output byte Z_1 produced by the cipher is negatively biased towards 1, i.e., $\mathsf{Pr}[Z_1 = 1] = \frac{1}{N} - \frac{1}{2N^2}$, (where $N = 256$ is the number of bytes in the internal state of the cipher) we were able to mount a distinguishing attack on the cipher that requires around 2^{26} first keystream bytes produced by either different Secret Keys or the same Secret Key and different IVs.

In the next part of the chapter, we mounted a differential fault attack on the RC4+ stream cipher using around $2^{17.2}$ faults. The main observation used to mount the attack was that if the attacker came to know around 602 values of the internal variable j in successive iterations of the cipher, then he could recover the entire internal state of the cipher. We explain how the attacker would identify the correct value of j in any iteration of the cipher, by applying faults in the internal state and hence complete the attack.

Chapter 4: Related Key-IV pairs of Grain

In this chapter we provided a formal introduction to the Grain family of stream ciphers [13, 73, 74]. We gave a detailed summary of the all the cryptanalytic attacks reported against this family. We observed that due to the nature of the update functions used the LFSR and the NFSR in all the three members of the Grain family, the state updates during both the KSA and the PRGA are reversible. As a result, one can formulate the KSA^{-1} algorithm that given the internal state after the completion of the KSA algorithm, outputs the internal state at the beginning of the KSA phase. We also described the PRGA^{-1} algorithm that inverts the state update of one PRGA round. Using these algorithms we were able to describe probabilistic methods that found related Key-IV pairs in the Grain family that produce keystream bits that are

1. Almost similar in the initial segment, or

2. Exact shifts of each other throughout the generation of the stream. The value of the shifts in this case is approximately 2^{l_p} where l_p is the length of the pad in bits used in the design ($l_p = 16$ for Grain v1 and 32 for Grain-128 and Grain-128a).

We then went on to investigate the possibility of obtaining related Key-IV pairs that produce shifted keystream bits with smaller shifts. In [44], a method for finding related Key-IV pairs that produced i-bit shifted keystream (for Grain v1 and Grain-128) was proposed that required 4^i random trials. The method mainly took advantage of the fact that in both Grain v1 and Grain 128, the symmetric all 1 constant was used as the pad. In this work, we have improved the method to 2^i random trials. Furthermore, in [44], it was observed that devising such a method for Grain-128a was not possible as the pad used in this cipher was asymmetric. In the last part of the chapter, we presented a method to find related Key-IV pairs that produce 32-bit shifted keystream bits for Grain-128a. The method takes around 2^{32} random trials. We presented another method that finds related Key-IV pairs that produces shifted keystream bits for shifts lesser than 32. The second method produces produce ϵ-bit shifted key-streams (for $0 < \epsilon < 32$) using $\frac{2^{32}}{1-2^{-\epsilon}}$ random trials.

Chapter 5: Differential Fault Analysis of Grain

Prior to our work, two fault attacks [35, 85] had been reported against the Grain family, both against Grain-128. No cryptanalytic advance had been made against Grain v1 or Grain-128a. In this chapter we have reported three fault attacks against all the three members of the Grain family, under various adversarial situations:

1. In the first attack we assumed that the attacker can apply single bit flipping, time-synchronized faults in some random unknown location of the internal state of the cipher. In addition, the attacker retains the ability to inject multiple faults in any particular register location. The attacker first deduces the location of the register where the applied fault has flipped the logic. Then, by exploiting certain first order differential properties of the output function h the attacker is able to formulate the requisite number of linear equations to find out the entire internal state of the cipher.

2. In the second attack, the attacker is still able to apply single bit flipping, time-synchronized faults in random register locations, but can no longer apply multiple faults in a singe register location. This attack uses, higher order differential properties of the output function h to formulate the necessary number of linear equations to solve for the internal state of the cipher.

3. In the third attack, the attacker makes use of non-linear equations to speed up the attack. After deducing the location of the fault, the attacker formulates a number of algebraic equations in the internal state variables corresponding to each faulty and faultless keystream bit output by the cipher. The bank of equations is then fed to a SAT solver, which finds a solution to the equation system in reasonable

amount of time. This leads to a drastic reduction in the number of faults required to complete the attack. Whereas the previous attacks required in excess of 2^9 faults, the third attack takes less than 10 faults for all the three ciphers in the Grain family.

Next, we explored the case when the attacker can apply a time synchronized fault that may disturb the logic in up to 3 continuous register locations. Using certain differential characteristics of the cipher, attacker is able distinguish, with high probability, whether a faulty keystream segment has been produced due to a multiple bit or a single bit-flipping of the original faultless internal state. In the event that the attacker finds a faulty keystream has been produced due to a multiple bit-flip, he simply discards the keystream, and inject fault afresh until and unless that he obtains a faulty keystream produced due to a single bit-flip whose location he can conclusively identify. Finally, we investigated the case when the attacker cannot fully synchronize the timing of his fault with the start of the PRGA, and the best he can do is inject the fault at some PRGA round $\tau \leq \tau_{max}$. We explained how the attacker is able find the value of τ and proceed with the attack.

Chapter 6: Conditional Differential Cryptanalysis of Grain

This chapter introduced the reader with a brief description of Knellwolf's Conditional Differential Cryptanalysis [93, 94] of reduced round Grain v1. The attack found the values of five expressions in the Secret Key bits of a variant of Grain v1 that employs only 97 out of the 160 rounds in its Key Scheduling. In [93], it was oblerved that if certain algebraic conditions involving the Key and the IV bits are satisfied then the following was observed.

$$\mathsf{Pr}[z_{97} \oplus z'_{97} = 0] \;=\; \frac{1}{2} + \epsilon$$

where z_{97}, z'_{97} are the keystream bits produced in the 97^{th} KSA round by two IVs which differ in the 37^{th} bit. Using this bias, 5 expressions in Secret Key bits were deduced. The above bias was obtained purely experimentally and no theoretical explanation was provided in [93, 94] of it. The chapter proposed the differential engine Δ_{ϕ}-$\mathsf{Grain}_{\mathsf{KSA}}$ that tracks differential trails introduced in the cipher during the KSA phase. Using the results obtained from this engine, the distributions of several intermediate internal variables of the cipher were computed and finally the bias in the distribution of $z_{97} \oplus z'_{97}$ was proven.

Chapter 7: Differential Fault Analysis of MICKEY 2.0

This chapter formally introduced the reader to the stream cipher MICKEY 2.0 [16]. Although the cipher is included in the final portfolio of eStream, no cryptanalytic advance

had been made against it prior to our work. In this chapter we mounted a Differential Fault Attack on MICKEY 2.0, under the standard assumption that the attacker is able to inject single bit flipping, time-synchronized faults in some random unknown location of the internal state of the cipher. The attack proceeded in the following manner.

1. We proved that the complete initial state of the cipher may be found out based on the knowledge of some intermediate internal bits, by solving some simple linear equations.
2. We then investigated how well-timed and well-localized fault injections could leak those very intermediate bits of the secret state.
3. We then built a distinguisher that could detect where the fault had occurred, so as to relax the hypothesis about the knowledge of the fault position in space.

We then relaxed the fault model and explored the situation in which a time synchronized fault disturbs the logic in up to 3 continuous register locations. In the second part of the chapter, we reduced the fault requirement by additionally making use of non-linear equations resulting from each faulty keystream bit. As in the case of the Grain family, a bank of equations were constructed which were fed to SAT solver which provided a solution in reasonable time.

Chapter 8: Improved Scan-Chain based Attacks and related Countermeasures

This chapter introduced the reader to Scan-Chain based hardware design and the related vulnerabilities that may creep into a cryptosystem implemented with Scan-Chains. In [11], A Scan based attack on the stream cipher Trivium [43] was presented. It was explained why the same attack could not be extended to MICKEY 2.0. Thereafter, an alternative strategy to attack MICKEY 2.0 via Scan-Chains was suggested. Further, in [11], an XOR gate based countermeasure was suggested to protect Scan Chains from cryptanalytic attacks. We showed that this countermeasure may fail to protect the underlying cryptosystem under the SET attack scenario. A novel Double Feedback XOR-CHAIN countermeasure was then proposed that was shown to be secure against this class of cryptanalytic attack. It was also shown than that such a Double Feedback XOR-CHAIN structure, like an ordinary Scan Chain, could also be used for DFT purposes.

9.2 Open Problems

We will now discuss some open problems in this line of research.

A. "Breaking" the ciphers: A cipher employing an n-bit Secret Key is said to be "broken", if a method for deducing the Secret Key, by simply observing the keystream bits and in the absence of any Side Channel information, is found that takes less than 2^n operations. The attacker can at most be allowed to manipulate the public variables of the cipher. Needless to say, that none of the eStream ciphers analyzed in this thesis have been broken, and so this remains an open problem in this domain.

B. Algebraic Cryptanalysis of Grain: With limited computational resources at our disposal we were able to use SAT solvers to solve for the internal state of the Grain family using as little as $10, 4, 10$ faults for Grain v1, Grain-128 and Grain-128a respectively. Note that each faulty keystream bit is essentially an extra source of information, that provides an extra equation in the internal state variables, which helps the SAT solver to come to a solution faster. This leads us to conjecture that an attacker having access to larger computational power may be able to reduce the number of faults required to attack the Grain family. It may even be possible to come at a solution without making the use of any equation generated due to a faulty keystream bit. The above statements, however, are merely conjectures and whether or not they can be achieved in practice is an extremely interesting topic of research.

C. Improving Knellwolf's attack on Grain v1: Although we were able to prove the correctness of Knellwolf's attack [93] for 97 round Grain v1, the attack at 104 rounds remains unclear. The author of [93] observes that at round 104, a bias is observed in one of the Sets of Chosen IVs in only about 50 % of the cases. It would be a worthwhile exercise, to determine explicitly, the algebraic conditions the Secret Key bits need to satisfy for the bias to be observed. Another open problem in this area is, of course, to use the engine Δ_ϕ-$\mathsf{Grain}_{\mathsf{KSA}}$ to attack a higher number rounds of the KSA of Grain v1.

D. Reducing the Fault requirement for MICKEY 2.0: The differential fault attack proposed against MICKEY 2.0, described in Chapter 6, takes around $2^{14.7}$ faults. This is considerably high when compared to the other two ciphers in the hardware portfolio of eStream (see Table 7.4). This leaves plenty of room for further improvement of this result. It would be worthwhile to explore whether it would be possible to reduce the number of faults required to attack MICKEY 2.0.

E. Fixing RC4+: In Section 3.5, we had observed that the distinguishing attack that was proposed against RC4+ would still work if any even pad other than `0xAA` was used in the design. Whether changing the value of the pad to an odd number would prevent the distinguishing attack is worth looking into.

9.3 Final Words

This thesis discusses a variety of cryptanalytic results against software based stream ciphers like RC4+, GGHN and hardware based stream ciphers like the members of the Grain family and MICKEY 2.0. By exposing some of the weaknesses in the designs of RC4+ and GGHN, we have been able to ascertain that looking for design paradigms that looks to rid RC4 of its known weaknesses and at the same time retains its innate simplicity and elegance is a non-trivial task. Prior to our work, a generic strategy to perform the fault analysis of the Grain family and MICKEY 2.0, were essentially open problems. The thesis also provides useful insights regarding scan based side channel attacks and their countermeasures. The research-work included in this thesis serves as a precursor to what we believe would be a long line of research on stream ciphers and side channel attacks.

Bibliography

[1] Advanced Encryption Standard process. From Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Advanced_Encryption_Standard_process.

[2] Data Encryption Standard. Federal Information Processing Standards Publication 46-3, 1999. http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

[3] Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186-3, 1999. http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf.

[4] RC4 Stream Cipher. http://en.wikipedia.org/wiki/RC4.

[5] PKCS #1 v2.2: RSA Cryptography Standard. RSA Laboratories, 2012. http://www.emc.com/collateral/white-papers/h11300-pkcs-1v2-2-rsa-cryptography-standard-wp.pdf.

[6] SHA-3 Competition (2007-2012). NIST Information Technology Laboratory. http://csrc.nist.gov/groups/ST/hash/sha-3/.

[7] Cryptography. From Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Cryptography.

[8] ETSI/SAGE Specification. Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. Document 2: ZUC Specification; Version: 1.6, 2011.

[9] Virtex-6 Libraries Guide for Schematic Designs, 2011. http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/virtex6_scm.pdf.

[10] Carlisle Adams and Steve LLoyd. *Understanding PKI - Concepts, Standards and Deployment Considerations*. Pearson Education Inc., 2003. ISBN 0-672-32391-5.

[11] Mukesh Agrawal, Sandip Karmakar, Dhiman Saha, and Debdeep Mukhopadhyay. Scan Based Side Channel Attacks on Stream Ciphers and Their Counter-Measures.

In *INDOCRYPT*, volume 5365 of *Lecture Notes in Computer Science*, pages 226–238. Springer, 2008.

[12] Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. A New Version of Grain-128 with Authentication. Symmetric Key Encryption Workshop 2011, DTU, Denmark, February 2011, 2011.

[13] Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: a new version of Grain-128 with optional authentication. *IJWMC*, 5(1):48–59, 2011.

[14] Jean-Philippe Aumasson, Itai Dinur, Luca Henzen, Willi Meier, and Adi Shamir. Efficient FPGA Implementations of High-Dimensional Cube Testers on the Stream Cipher Grain-128. In SHARCS - Special-purpose Hardware for Attacking Cryptographic Systems, 2009.

[15] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and María Naya-Plasencia. Quark: A Lightweight Hash. *J. Cryptology*, 26(2):313–339, 2013.

[16] Steve Babbage and Matthew Dodd. The stream cipher MICKEY 2.0. eSTREAM, ECRYPT Stream Cipher Project Report, 2005. http://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey_p3.pdf.

[17] Steve Babbage and Matthew Dodd. The stream cipher MICKEY-128 2.0. eSTREAM, ECRYPT Stream Cipher Project Report, 2005. http://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey128_p3.pdf.

[18] Subhadeep Banik. Some Insights into Differential Cryptanalysis of Grain v1. In *ACISP*, volume 8544 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2014.

[19] Subhadeep Banik and Anusha Chowdhury. Improved Scan-Chain Based Attacks and Related Countermeasures. In *INDOCRYPT*, volume 8250 of *Lecture Notes in Computer Science*, pages 78–97. Springer, 2013.

[20] Subhadeep Banik and Subhamoy Maitra. A Differential Fault Attack on MICKEY 2.0. In *CHES*, volume 8086 of *Lecture Notes in Computer Science*, pages 215–232. Springer, 2013.

[21] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. On the Evolution of GGHN Cipher. In *INDOCRYPT*, volume 7107 of *Lecture Notes in Computer Science*, pages 181–195. Springer, 2011.

[22] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. A Differential Fault Attack on the Grain Family of Stream Ciphers. In *CHES*, volume 7428 of *Lecture Notes in Computer Science*, pages 122–139. Springer, 2012.

[23] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. A Differential Fault Attack on the Grain Family under Reasonable Assumptions. In *INDOCRYPT*, volume 7668 of *Lecture Notes in Computer Science*, pages 191–208. Springer, 2012.

[24] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. A Differential Fault Attack on Grain-128a Using MACs. In *SPACE*, volume 7644 of *Lecture Notes in Computer Science*, pages 111–125. Springer, 2012.

[25] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. Some Results on Related Key-IV Pairs of Grain. In *SPACE*, volume 7644 of *Lecture Notes in Computer Science*, pages 94–110. Springer, 2012.

[26] Subhadeep Banik, Subhamoy Maitra, Santanu Sarkar, and Meltem Sönmez Turan. A Chosen IV Related Key Attack on Grain-128a. In *ACISP*, volume 7959 of *Lecture Notes in Computer Science*, pages 13–26. Springer, 2013.

[27] Subhadeep Banik, Santanu Sarkar, and Raghu Kacker. Security Analysis of the RC4+ Stream Cipher. In *INDOCRYPT*, volume 8250 of *Lecture Notes in Computer Science*, pages 297–307. Springer, 2013.

[28] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. Improved Differential Fault Attack on MICKEY 2.0. *To appear in Journal of Cryptographic Engineering*, 2014. DOI : 10.1007/s13389-014-0083-9.

[29] Gregory Bard, editor. *Algebraic Cryptanalysis*. Security and Cryptology. Springer, 2009. ISBN 978-0-387-88757-9.

[30] 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. eSTREAM, ECRYPT Stream Cipher Project Report, 2006. http://www.ecrypt.eu.org/stream/p3ciphers/sosemanuk/sosemanuk_p3.pdf.

[31] Côme Berbain, Henri Gilbert, and Alexander Maximov. Cryptanalysis of Grain. In *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 15–29. Springer, 2006.

[32] Daniel J. Bernstein. Cache-timing attacks on AES, 2005. http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

[33] Daniel J. Bernstein. Salsa20/8 and Salsa20/12. eSTREAM, ECRYPT Stream Cipher Project Report, 2006. http://www.ecrypt.eu.org/stream/papersdir/2006/007.pdf.

[34] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak sponge function family, 2011. http://keccak.noekeon.org/specs_summary.html.

[35] Alexandre Berzati, Cecile Canovas, Guilhem Castagnos, Blandine Debraize, Louis Goubin, Aline Gouget, Pascal Paillier, and Stephanie Salgado. Fault Analysis of Grain-128. IEEE International Workshop on Hardware-Oriented Security and Trust, 2009.

[36] Alexandre Berzati, Cécile Canovas-Dumas, and Louis Goubin. Fault Analysis of Rabbit: Toward a Secret Key Leakage. In *INDOCRYPT*, volume 5922 of *Lecture Notes in Computer Science*, pages 72–87. Springer, 2009.

[37] Eli Biham and Adi Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.

[38] Eli Biham, Louis Granboulan, and Phong Q. Nguyen. Impossible Fault Analysis of RC4 and Differential Fault Analysis of RC4. In *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 359–367. Springer, 2005.

[39] Tor E. Bjorstad. Cryptanalysis of Grain using Time/Memory/Date Tradeoffs. eSTREAM, ECRYPT Stream Cipher Project, Report 200508/012, 2008. http://www.ecrypt.eu.org/stream.

[40] Martin Boesgaard, Mette Vesterager, Thomas Christensen, and Erik Zenner. The Stream Cipher Rabbit. eSTREAM, ECRYPT Stream Cipher Project Report, 2006. http://www.ecrypt.eu.org/stream/p3ciphers/rabbit/rabbit_p3.pdf.

[41] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract). In *EUROCRYPT*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

[42] Philippe Bulens, Kassem Kalach, Francois-Xavier Standaert, and Jean-Jacques Quisquater. Hardware performance of eStream phase-III stream cipher candidates. SASC, 2007. http://www.ecrypt.eu.org/stream/papersdir/2007/024.pdf.

[43] Christophe De Cannière and Bart Preneel. TRIVIUM -Specifications. eSTREAM, ECRYPT Stream Cipher Project Report, 2005. http://www.ecrypt.eu.org/stream/p3ciphers/trivium/trivium_p3.pdf.

[44] Christophe De Cannière, Özgül Küçük, and Bart Preneel. Analysis of Grain's Initialization Algorithm. In *AFRICACRYPT*, volume 5023 of *Lecture Notes in Computer Science*, pages 276–289. Springer, 2008.

[45] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2009.

[46] Çagdas Çalik, Meltem Sönmez Turan, and Ferruh Özbudak. On Feedback Functions of Maximum Length Nonlinear Feedback Shift Registers. *IEICE Transactions*, 93-A(6):1226–1231, 2010.

[47] Carlos Cid and Matt Robshaw (editors). The eSTREAM Portfolio in 2012, 16 January 2012, Version 1.0. eSTREAM, ECRYPT Stream Cipher Project Report. http://www.ecrypt.eu.org/documents/D.SYM.10-v1.pdf.

[48] Ronald Cramer and Victor Shoup. A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack. In *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 1998.

[49] Joan Daemen and Paris Kitsos. The self-synchronizing stream cipher Moustique. eSTREAM, ECRYPT Stream Cipher Project Report, 2006. http://www.ecrypt.eu.org/stream/p2ciphers/mosquito/mosquito_p2.pdf.

[50] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer-Verlag, 2002. ISBN 3-540-42580-2.

[51] Apurba Das, Subhamoy Maitra, Goutam Paul, and Santanu Sarkar. Some Combinatorial Results towards State Recovery Attack on RC4. In *ICISS*, volume 7093 of *Lecture Notes in Computer Science*, pages 204–214. Springer, 2011.

[52] Blandine Debraize and Irene Marquez Corbella. Fault Analysis of the Stream Cipher Snow 3G. In *FDTC*, pages 103–110, 2009.

[53] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Trans. on Information Theory*, 22(6):644–654, 1976.

[54] Cunsheng Ding, Guozhen Xiao, and Weijuan Shan. *The Stability Theory of Stream Ciphers*, volume 561 of *Lecture Notes in Computer Science*. Springer, 1991. ISBN 3-540-54973-0.

[55] Itai Dinur and Adi Shamir. Cube Attacks on Tweakable Black Box Polynomials. In *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 278–299. Springer, 2009.

[56] Itai Dinur and Adi Shamir. Breaking Grain-128 with Dynamic Cube Attacks. In *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 167–187. Springer, 2011.

[57] Itai Dinur, Tim Güneysu, Christof Paar, Adi Shamir, and Ralf Zimmermann. An Experimentally Verified Attack on Full Grain-128 Using Dedicated Reconfigurable Hardware. In *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 327–343. Springer, 2011.

[58] Håkan Englund, Thomas Johansson, and Meltem Sönmez Turan. A Framework for Chosen IV Statistical Analysis of Stream Ciphers. In *INDOCRYPT*, volume 4859 of *Lecture Notes in Computer Science*, pages 268–281. Springer, 2007.

[59] Paul Erdős and Alfréd Rényi. On a classical problem of probability theory. *Magyar Tudományos Akadémia Matematikai Kutató Intézetének Közleményei*, 6:215–220, 1961. Available at http://www.renyi.hu/~p_erdos/1961-09.pdf.

[60] Hal Finney. *An RC4 cycle that can't happen. Posting to sci. crypt*, 1994.

[61] Simon Fischer, Shahram Khazaei, and Willi Meier. Chosen IV Statistical Analysis for Key Recovery Attacks on Stream Ciphers. In *AFRICACRYPT*, volume 5023 of *Lecture Notes in Computer Science*, pages 236–245. Springer, 2008.

[62] Thomas L. Floyd, editor. *Digital Fundamentals (10th Edition)*. Prentice Hall, 2008. ISBN 978-0132359238.

[63] Harold Fredricksen. A Survey of Full Length Nonlinear Shift Register Cycle Algorithms. *SIAM Rev.*, 24:195–221, 1982.

[64] Kris Gaj, Gabriel Southern, and Ramakrishna Bachimanchi. Comparison of hardware performance of selected Phase II eSTREAM candidates. SASC, 2007. http://www.ecrypt.eu.org/stream/papersdir/2007/026.pdf.

[65] Simson Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly & Associates, 1991. ISBN 1-56592-098-8.

[66] Benedict Gierlichs, Lejla Batina, Christophe Clavier, Thomas Eisenbarth, Aline Gouget, Helena Handschuh, Timo Kasper, Kerstin Lemke-Rust, Stefan Mangard, Amir Moradi, and Elizabeth Oswald. Susceptibility of eSTREAM Candidates towards Side Channel Analysis. SASC, 2008. http://www.ecrypt.eu.org/stvl/sasc2008/.

[67] Oded Goldreich. *Foundations of Cryptography - Basic Tools*. Cambridge University Press, 2003. ISBN 0-521-79172-3.

[68] Guang Gong, Kishan Chand Gupta, Martin Hell, and Yassir Nawaz. Towards a General RC4-Like Keystream Generator. In *CISC*, volume 3822 of *Lecture Notes in Computer Science*, pages 162–174. Springer, 2005.

[69] Tim Good and Mohammed Benaissa. Hardware performance of eStream phase-III stream cipher candidates. SASC, 2008. http://www.ecrypt.eu.org/stream/docs/hardware.pdf.

[70] C. M. Grinstead and J. L. Snell, editors. *Introduction to Probability*. American Mathematical Society, 2006. ISBN 978-0821807491.

[71] D.H. Habing. Use of Laser to Simulate Radiation-induced Transients In Semiconductors and Circuits. *IEEE Transactions on Nuclear Science*, NS-12(6):91–100, 1965.

[72] Martin Hell, Thomas Johansson, and Willi Meier. Grain - A Stream Cipher for Constrained Environments. eSTREAM, ECRYPT Stream Cipher Project Report, 2005. http://www.ecrypt.eu.org/stream/ciphers/grain/grain.pdf.

[73] Martin Hell, Thomas Johansson, and Willi Meier. Grain - A Stream Cipher for Constrained Environments. eSTREAM, ECRYPT Stream Cipher Project Report, 2005. http://www.ecrypt.eu.org/stream/p3ciphers/grain/Grain_p3.pdf.

[74] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. A Stream Cipher Proposal: Grain-128. eSTREAM, ECRYPT Stream Cipher Project Report, 2008. http://www.ecrypt.eu.org/stream/p3ciphers/grain/Grain128_p3.pdf.

[75] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.

[76] Jonathan J. Hoch and Adi Shamir. Fault Analysis of Stream Ciphers. In *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 240–253. Springer, 2004.

[77] Jeff Hoffstein, Nick Howgrave-Graham, Jill Pipher, and William Whyte. Practical lattice-based cryptography: NTRUEncrypt and NTRUSign. NTRU Cryptosystems, 1998. https://www.securityinnovation.com/uploads/Crypto/lll25.pdf.

[78] Michal Hojsík and Bohuslav Rudolf. Differential Fault Analysis of Trivium. In *FSE*, volume 5086 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 2008.

[79] Michal Hojsík and Bohuslav Rudolf. Floating Fault Analysis of Trivium. In *INDOCRYPT*, volume 5365 of *Lecture Notes in Computer Science*, pages 239–250. Springer, 2008.

[80] Jin Hong and Woo-Hwan Kim. TMD-Tradeoff and State Entropy Loss Considerations of Streamcipher MICKEY. In *INDOCRYPT*, volume 3797 of *Lecture Notes in Computer Science*, pages 169–182. Springer, 2005.

[81] Yupu Hu, Juntao Gao, Qing Liu, and Yiwei Zhang. Fault analysis of Trivium. *Des. Codes Cryptography*, 62(3):289–311, 2012.

[82] A.H. Johnston. Charge Generation and Collection in p-n Junctions Excited with Pulsed Infrared Lasers. *IEEE Transactions on Nuclear Science*, NS-40(6):1694–1702, 1993.

[83] Marc Joye and Michael Tunstall, editors. *Fault Analysis in Cryptography*. Information Security and Cryptography. Springer, 2012. ISBN 978-3-642-29655-0.

[84] David Kahn, editor. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. Scribner, 1996. ISBN 978-0684831305.

[85] Sandip Karmakar and Dipanwita Roy Chowdhury. Fault Analysis of Grain-128 by Targeting NFSR. In *AFRICACRYPT*, volume 6737 of *Lecture Notes in Computer Science*, pages 298–315. Springer, 2011.

[86] Selçuk Kavut, Subhamoy Maitra, and Melek D. Yücel. Search for Boolean Functions With Excellent Profiles in the Rotation Symmetric Class. *IEEE Transactions on Information Theory*, 53(5):1743–1751, 2007.

[87] Auguste Kerkchoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:161–191, 1883.

[88] Shahram Khazaei, Mehdi Hassanzadeh, and Mohammad Kiaei. Distinguishing Attack on Grain. eSTREAM, ECRYPT Stream Cipher Project Report, 2005. http://www.ecrypt.eu.org/stream/papersdir/071.pdf.

[89] Aleksandar Kircanski and Amr M. Youssef. On the structural weakness of the GGHN stream cipher. *Cryptography and Communications*, 2(1):1–17, 2010.

[90] Aleksandar Kircanski and Amr M. Youssef. Differential Fault Analysis of HC-128. In *AFRICACRYPT*, volume 6055 of *Lecture Notes in Computer Science*, pages 261–278. Springer, 2010.

[91] Aleksandar Kircanski and Amr M. Youssef. Differential Fault Analysis of Rabbit. In *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 197–214. Springer, 2009.

[92] Paris Kitsos. On the Hardware Implementation of the MICKEY-128 Stream Cipher. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/059, 2006. http://www.ecrypt.eu.org/stream/papersdir/2006/059.pdf.

[93] Simon Knellwolf. Cryptanalysis of Hardware-Oriented Ciphers, The Knapsack Generator, and SHA-1. PhD Dissertation submitted to ETH Zurich, 2012. http://e-collection.library.ethz.ch/eserv/eth:5999/eth-5999-02.pdf.

[94] Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional Differential Cryptanalysis of NLFSR-Based Cryptosystems. In *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 130–145. Springer, 2010.

[95] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[96] Yuseop Lee, Kitae Jeong, Jaechul Sung, and Seokhie Hong. Related-Key Chosen IV Attacks on Grain-v1 and Grain-128. In *ACISP*, volume 5107 of *Lecture Notes in Computer Science*, pages 321–335. Springer, 2008.

[97] Michael Lehmann and Willi Meier. Conditional Differential Cryptanalysis of Grain-128a. In *CANS*, volume 7712 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2012.

[98] Rudolf Lidl and Harald Niederreter. *Introduction to Finite Fields and their Applications*. Cambridge University Press, 1986. ISBN 0-521-30706-6.

[99] Zhouqian Ma and Dawu Gu. Improved Differential Fault Analysis of SOSEMANUK. In *Eighth International Conference on Computational Intelligence and Security (CIS)*, pages 487–491, Nov 2012. doi: 10.1109/CIS.2012.115.

[100] Subhamoy Maitra and Goutam Paul. Analysis of RC4 and Proposal of Additional Layers for Better Security Margin. In *INDOCRYPT*, volume 5365 of *Lecture Notes in Computer Science*, pages 27–39. Springer, 2008.

[101] Subhamoy Maitra, Goutam Paul, and Sourav Sengupta. Attack on Broadcast RC4 Revisited. In *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 199–217. Springer, 2011.

[102] Itsik Mantin and Adi Shamir. A Practical Attack on Broadcast RC4. In *FSE*, volume 2355 of *Lecture Notes in Computer Science*, pages 152–164. Springer, 2001.

[103] James L. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15(1):122–127, 1969.

[104] Alexander Maximov. Two Linear Distinguishing Attacks on VMPC and RC4A and Weakness of RC4 Family of Stream Ciphers. In *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 342–358. Springer, 2005.

[105] Alexander Maximov and Dmitry Khovratovich. New State Recovery Attack on RC4. In *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 297–316. Springer, 2008.

[106] Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN progress report*, 42(44):114–116, 1978. http://www.cs.colorado.edu/~jrblack/class/csci7000/f03/papers/mceliece.pdf.

[107] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, editors. *Handbook of Applied Cryptography (Discrete Mathematics and Its Applications)*. CRC Press, 1996. ISBN 978-0849385230.

[108] Miodrag Mihaljevic, Sugata Gangopadhyay, Goutam Paul, and Hideki Imai. Internal State Recovery of Grain-v1 Using Normality Order of the Filter Function. *IET Information Security*, 6(2):55–64, 2012.

[109] Mohamed S. E. Mohamed, Stanislav Bulygin, and Johannes Buchmann. Improved Differential Fault Analysis of Trivium. In COSADE 2011, Darmstadt, Germany, February 24–25, 2011.

[110] Johannes Mykkeltveit. The covering radius of the (128, 8) Reed-Muller code is 56 (Corresp.). *IEEE Transactions on Information Theory*, 26(3):359–362, 1980.

[111] Yassir Nawaz, Kishan Chand Gupta, and Guang Gong. A 32-bit RC4-like Keystream Generator. *IACR Cryptology ePrint Archive*, 2005:175, 2005.

[112] Edward D. Palik. *Handbook of Optical Constants of Solids*. Orlando Academic Press, 1985. ISBN 978-0125444231.

[113] Nick J. Patterson and Douglas H. Wiedemann. Correction to 'The covering radius of the (2^{15}, 16) Reed-Muller code is at least 16276' (May 83 354-356). *IEEE Transactions on Information Theory*, 36(2):443, 1990.

[114] 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*, volume 3017 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 2004.

[115] Souradyuti Paul and Bart Preneel. On the (In)security of Stream Ciphers Based on Arrays and Modular Addition. In *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2006.

[116] Matthew J. B. Robshaw and Olivier Billet, editors. *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*. Springer, 2008. ISBN 978-3-540-68350-6.

[117] Marcin Rogawski. Hardware evaluation of eSTREAM Candidates: Grain, Lex, Mickey128, Salsa20 and Trivium. SASC, 2007. http://www.ecrypt.eu.org/stream/papersdir/2007/025.pdf.

[118] O. S. Rothaus. On "Bent" Functions. *J. Comb. Theory, Ser. A*, 20(3):300–305, 1976.

[119] Yaser Esmaeili Salehani, Aleksandar Kircanski, and Amr M. Youssef. Differential Fault Analysis of Sosemanuk. In *AFRICACRYPT*, volume 6737 of *Lecture Notes in Computer Science*, pages 316–331. Springer, 2011.

[120] Santanu Sarkar, Subhadeep Banik, and Subhamoy Maitra. Differential Fault Attack against Grain family with very few faults and minimal assumptions. *To appear in IEEE Transactions on Computers*, 2014. DOI: 10.1109/TC.2014.2339854.

[121] Gaurav Sengar, Debdeep Mukhopadhyay, and Dipanwita Roy Chowdhury. Secured Flipped Scan-Chain Model for Crypto-Architecture. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 26(11):2080–2084, 2007.

[122] Pouyan Sepehrdad, Serge Vaudenay, and Martin Vuagnoux. Statistical Attack on RC4 - Distinguishing WPA. In *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 343–363. Springer, 2011.

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

[124] Sergei P. Skorobogatov. Optically Enhanced Position-Locked Power Analysis. In *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 2006.

[125] Sergei P. Skorobogatov and Ross J. Anderson. Optical Fault Induction Attacks. In *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 2002.

[126] Mate Soos. CryptoMiniSat-2.9.5. http://www.msoos.org/cryptominisat2/.

[127] Arthur Sorkin. Lucifer, A Cryptographic Algorithm, 1983. http://fuseki.com/lucifer.pdf.

[128] Paul Stankovski. Greedy Distinguishers and Nonrandomness Detectors. In *INDOCRYPT*, volume 6498 of *Lecture Notes in Computer Science*, pages 210–226. Springer, 2010.

[129] William Stein. Sage Mathematics Software. Free Software Foundation, Inc., 2009. http://www.sagemath.org (Open source project initiated by W. Stein and contributed by many).

[130] Elmar Tischhauser. Nonsmooth cryptanalysis, with an application to the stream cipher MICKEY. *J. Mathematical Cryptology*, 4(4):317–348, 2011.

[131] Yukiyasu Tsunoo, Teruo Saito, Hiroyasu Kubo, Maki Shigeri, Tomoyasu Suzaki, and Takeshi Kawabata. The Most Efficient Distinguishing Attack on VMPC and RC4A. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/037, 2005. http://www.ecrypt.eu.org/stream/papersdir/037.pdf.

[132] Yukiyasu Tsunoo, Teruo Saito, Hiroyasu Kubo, and Tomoyasu Suzaki. A Distinguishing Attack on a Fast Software-Implemented RC4-Like Stream Cipher. *IEEE Transactions on Information Theory*, 53(9):3250–3255, 2007.

[133] Hongjun Wu. HC-128. eSTREAM, ECRYPT Stream Cipher Project Report, 2006. http://www.ecrypt.eu.org/stream/p3ciphers/hc/hc128_p3.pdf.

[134] Bo Yang, Kaijie Wu, and Ramesh Karri. Scan Based Side Channel Attack on Dedicated Hardware Implementations of Data Encryption Standard. In *ITC*, pages 339–344, 2004.

[135] Bo Yang, Kaijie Wu, and Ramesh Karri. Secure Scan: A Design-for-Test Architecture for Crypto Chips. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 25(10):2287–2293, 2006.

[136] Bin Zhang and Zhenqi Li. Near Collision Attack on the Grain v1 Stream Cipher. In *FSE*, volume 8424 of *Lecture Notes in Computer Science*. Springer, 2013.

[137] Haina Zhang and Xiaoyun Wang. Cryptanalysis of Stream Cipher Grain Family. eSTREAM, ECRYPT Stream Cipher Project, Report 2009/109, 2009. http://www.ecrypt.eu.org/stream.

[138] Bartosz Zoltak. VMPC One-Way Function and Stream Cipher. In *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 210–225. Springer, 2004.