Design and Analysis of Symmetric Cryptographic Primitives

Thesis

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

SUSIL KUMAR BISHOI

(Reg. No. 197501)

DEPARTMENT OF MATHEMATICAL AND COMPUTATIONAL SCIENCES

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

SURATHKAL - 575025

June 2023

Dedication

"You are for the Lord
not for others;
You are for the Lord
and so for others."

Sri Sri Thakur Anukulchandra

DECLARATION

I hereby *declare* that the Research Thesis entitled **"DESIGN AND ANALYSIS OF SYMMETRIC CRYPTOGRAPHIC PRIMITIVES"**, which is being submitted to the ***National Institute of Technology Karnataka, Surathkal*** in partial fulfilment of the requirements for the award of the Degree of **Doctor of Philosophy** in **Mathematical and Computational Sciences** is a *bonafide report of the research work carried out by me*. The material contained in this thesis has not been submitted to any University or Institution for the award of any degree.

Susil Kumar Bishoi
Register No.: 197501
Department of Mathematical and Computational Sciences

Place: NITK, Surathkal

Date: June 2023

CERTIFICATE

This is to *certify* that the Research Thesis entitled **"DESIGN AND ANALYSIS OF SYMMETRIC CRYPTOGRAPHIC PRIMITIVES"**, submitted by **Susil Kumar Bishoi** (Register Number: 197501) as the record of the research work carried out by him, is *accepted* as the *Research Thesis submission* in partial fulfillment of the requirements for the award of degree of **Doctor of Philosophy**.

Dr. Kedarnath Senapati
Research Guide
Assistant Professor
Department of MACS
NITK Surathkal - 575025

Prof. B R Shankar
Research Co-Guide
Professor
Department of MACS
NITK Surathkal - 575025

Chairperson - DRPC
(Signature with Date and Seal)

Acknowledgements

Without close interaction, inspiration, advice, and support from numerous individuals, this thesis would not have been possible. It is in its current form due to my supervisors Dr. Kedarnth Senapati and Dr. B R Shankar. The contents and ingredients of the thesis are primarily because of their vision and help; I owe them a lot.

I am extremely grateful to the Research Progress Assessment Committee members Dr. J Ramalingam, Dept. of MACS, and Dr. A V Narasimhadhan, Department of ECE for their insightful suggestions.

I express my gratitude to Prof. P C Das and Prof. G Das, Prof. S Pattanayak for motivating me toward a career in mathematics during my MSc courses in Mathematics. To each of my teachers, I am extremely grateful for their wonderful teaching which worked as a ladder to reach this stage.

I am thankful to Director, CAIR, DRDO for giving me the opportunity to pursue my research work. I am thankful to Shri K Ravi Shankar, Ms V P Persia for their support during my PhD course. My heartfelt regards to Shri T.S. Raghavan and many thanks to Himanshu, for their guidance and valuable discussions during my initial period at CAIR. I would like to thank Bhupendra, Moon, Lexy, and Alok for their useful discussions with me.

I acknowledge my sincere thanks to Suryanarayan Maharana, Bhagabanjee, and Satyabrata for spending their valuable time on a careful reading of an early draft of this thesis. I thank Smitha, Abhishek, Sumukh, Jain, Sai Prasanna, ex and present staff members of the MACS, NITK for their help and suggestions with various technical/nontechnical works.

The amount of love and support I get from my wonderful family can't be expressed in words, especially from Dhanu, Sudeetta, and Sandeepaa. Without my Dhanu's cooperation and sacrifice, it would be impossible to reach this stage. I am blessed as a disciple of Sri Sri Thakur who guide me to lead a spiritual, disciplined, and happy lifestyle. Last but not least I am very proud of my parents and in-laws for their understanding, never-ending blessings, faith, and support.

ABSTRACT

Stream ciphers are well-known primitives that are used to ensure privacy over a communication channel. The central theme of a stream cipher is a keystream generator that produces a pseudorandom bit sequence. In such algorithms, the keystream bits are usually XORed with the plaintext bits to produce the ciphertext bits. Therefore, the design and analysis of new keystream generators are important. This thesis presents the design and analysis of several word-oriented keystream generators.

The word-based LFSR also known as the multiple recursive matrix method (MRMM) is very attractive as it possesses most of the randomness properties like LFSR. It also takes advantage of modern word-based processors and thus increases the throughput. We introduce a generalized form of the feedback function for word-oriented feedback shift registers (WFSRs) along with some special cases. A necessary and sufficient condition for nonsingular WFSR is also provided.

Like LFSR, the major drawback of MRMM is that it has very low linear complexity. In order to address the low linear complexity drawback in MRMM, the concept of several bit-oriented generators has been introduced. First, word-oriented shrinking and self-shrinking generators are studied where the lower bound for the period as well as for the linear complexity of the bitstream is shown to be exponential. Further experimentation and research have resulted in the identification and then mathematical verification of the exact period of self-shrinking generators.

Different word-based cascade systems along with their periods are studied. We provide experimental results on avalanche property on the states of all cascade systems and then analyze the statistical results of the keystream. We present a cryptanalytic attack on the cascade systems and suggest its countermeasure. In the later part of the thesis, we extend the idea of bit-oriented alternating step generators and nonlinear combination generators to the respective word-oriented generators.

Keywords: Stream Cipher, LFSR, LFG, Multiple-Recursive Matrix Method, Cascade Generator, Linear Complexity, Shrinking Generator, Alternative Step Generator, Statistical Tests

Contents

List of Figures

List of Tables

Acronyms

ASG	Alternating Step Generator
CS	Cascade System
FSM	Finite State Machine
FSR	Feedback Shift Register
LFSR	Linear Feedback Shift Register
LRR	Linear Recurring Relation
LFG	Lagged Fibbonacci Generator
MRMM	Multiple Recursive Matrix Method
NLFSR	Non-linear Feedback Shift Register
PRNG	Pseudorandom Number Generator
PKC	Public Key Cryptography
SKC	Symmetric-Key Cryptography
SSG	Self-Shrinking Generator
Sigma LFSR	σ-LFSR
WFSR	Word-oriented Feedback Shift Register

Chapter 1

Introduction

In the 21st century, information technology and electronic communication play a crucial role in our lives. They have become even more significant in the post-pandemic era due to their contactless nature. The volume of electronically exchanged and stored data is growing rapidly and is expected to continue its upward trend in the future. However, the increase in digital technologies also brings about a rise in online frauds, scams, intrusions, and security breaches. In such a scenario, the challenge is to ensure secure communication over insecure channels and secure storage of digital data. The cryptography [37, 46, 50] takes a leading role to solve these challenges. Cryptography has a long and fascinating history, showcasing remarkable advancements in the art of secret communication.

A widespread example is the technique used by the Roman military and political leader Julius Caesar on secret writing during the first century BC. Another well-known milestone is the "Enigma system", developed by German scientists in the 1920s and famously used by Nazi Germany. The Enigma machine, considered the first specialized computer used for encryption, was eventually deciphered by Marian Rejewski. This breakthrough allowed the Western Allies to exploit Enigma-enciphered messages and significantly shortening the duration of the war. The Enigma system marks a pivotal moment in the evolution of modern cryptography.

Earlier cryptography was mostly restricted to government agencies and the military.

Now it has been extensively used by the telecommunication industry, the financial world, the healthcare sectors, the entertainment industry, etc.

Fig. 1.1 illustrates the typical path of information flow in an insecure channel, comprising four main stages:

1. Source Coding: Redundancy is eliminated from the source, resulting in compressed information. Source encoders like ZIP and RAR are commonly used for this purpose. Source decoding reverses the compression to restore the original data.

2. Encryption: The compressed information undergoes encryption, transforming it into ciphertext, which is an encrypted message.

3. Channel Coding: Redundancy is added to the data stream to facilitate error detection and correction during transmission. Error-correcting codes, employed in applications such as media compact discs and mobile communication, help ensure data integrity.

4. Modulation: The modulation process alters the properties (amplitude, frequency, or phase) of a radio frequency or light wave to transmit the information effectively through the channel.

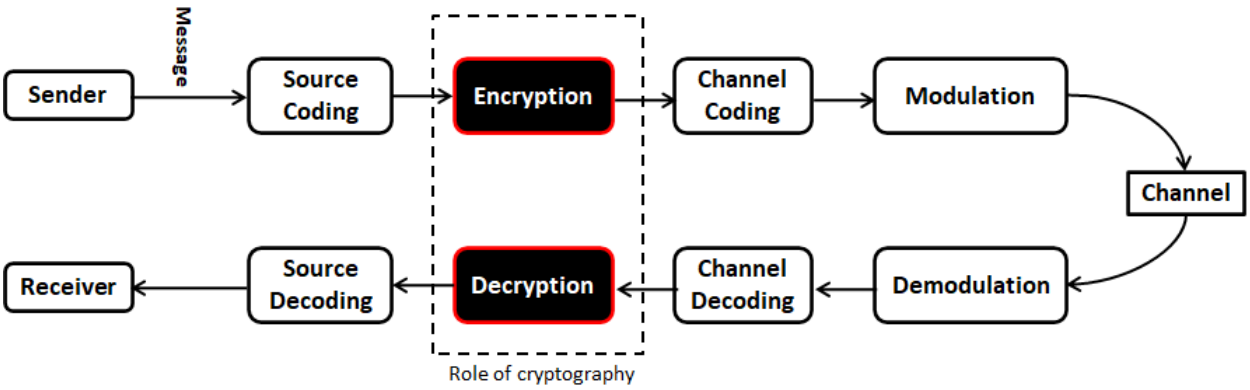

Figure 1.1: Role of cryptography in a communication system.

Encryption is used for secret storage or transmission of the message from one end to another such that any attacker or eavesdropper cannot understand the meaning of the message.

Only intended people can read/understand after using the reverse process called decryption. So, encryption safeguards the personal security of millions of people and the national security of countries around the world. After encryption, the ciphertext is transmitted by the sender to the intended receiver over an insecure communication channel. At the receiver end, the plaintext is recovered using the decryption algorithm. The algorithm used for performing encryption and decryption is called the cryptosystem or cipher.

There are two types of cryptosystems given as follows:

1. Symmetric key cryptosystems (also called secret key cryptosystems).

2. Asymmetric key cryptosystems (also known as public key cryptosystems).

This thesis aims mostly on the area of type 1. For asymmetric key cryptosystems, refer [37, 50].

1.1 Symmetric key cryptosystems

Symmetric key encryption is important for secure communication and storage. In a symmetric key cryptosystem, the decryption key K_d is either easily derivable from K_e or the same as the encryption key K_e. But in most of the ciphers, both K_e and K_d are the same key K [37]. So we consider both K_e and K_d to be the same. Both sender and receiver share the same secret key and the same symmetric key cipher before the communication starts. Symmetric key cryptosystems are usually categorized into block cipher and stream cipher.

1.1.1 Block cipher

In block cipher, the input message M breaks into successive blocks, where each message block is of a fixed length known as block size (typically containing 64-256 bits) and suitable padding is used if the length of the last message block is less than the block size. Suppose the message $M = M_1||M_2 \dots ||M_n$. Then the block cipher takes each message block M_i one by one along with the secret key and outputs a fixed-length ciphertext C_i for $1 \leq i \leq n$. That is,

M is encrypted to $C = C_1||C_2 \ldots ||C_n$ where $C_1 = E_k(M_1), C_2 = E_k(M_2), \ldots, C_n = E_k(M_n)$ and further C is transmitted across the channel to the receiver. At the receiver end, M is recovered using the decryption algorithm as $D_k(C_i) = M_i$ for $1 \leq i \leq n$. The literature on block cipher is extremely rich and one may refer to [37, 50] for more details. Advanced Encryption Standard (AES) [38], DES [39], RC6 [44], SERPENT [4], TWOFISH [47], etc are some of the popular block ciphers.

1.1.2 Stream cipher

An important class of encryption algorithms is Stream cipher. They are very simple block ciphers having block length equal to one. Prior to the eSTREAM project, several well-known stream ciphers, such as A5, E0, HELIX [18], FISH, RC4, SNOW [17], were in use. The eSTREAM project [16], initiated by ECRYPT, spanned multiple years to discover and evaluate new and promising stream ciphers. The project solicited proposals for new stream ciphers started in November 2004. These proposals were intended to satisfy either a hardware-oriented or a software-oriented profile or both. With an announcement of a portfolio of eight stream ciphers [16], four in each profile, the project ended in April 2008. The hardware-oriented cipher, F-FCSR-H v2, was eliminated from this portfolio in September 2008. The other seven ciphers were Grain v1, HC-128, MICKEY 2.0, Rabbit, Salsa20/12, SOSEMANUK, and Trivium.

In a stream cipher, if $k_1, k_2, \ldots, k_t$ are the keystream bits generated from the key K and $m_1, m_2, \ldots, m_t$ are the message bits of message M, then the ciphertext bits $c_1, c_2, \ldots, c_t$ are in general produced as $c_i = m_i \bigoplus k_i$, $1 \leq i \leq t$, where t is the length of M in bits. The classical example of stream cipher is the one-time pad in which keystream bits are generated randomly and independently. It is proven by Shannon [48] that the one-time pad is unconditionally secure against a ciphertext-only attack. A necessary condition for a symmetric-key encryption scheme to be unconditionally secure is that the entropy of K is greater or equal to the entropy of M [48]. So, to ensure unconditional security, the length of the plaintext should be less than or equal to the length of the key, and that

makes key distribution and key management challenging. This motivates the design of stream ciphers based on the keystream generated by a pseudorandom bit generator. While pseudorandom bit generator-based stream ciphers are not unconditionally secure, their main goal is to achieve computational security. A pseudorandom bit generator is a deterministic algorithm that takes a smaller size random key K and produces a random-looking binary sequence of a large period. So, the problem in designing stream cipher is to construct a good pseudorandom bit generator. Fig. 1.2 illustrates the typical application of stream cipher in communication. Feedback shift registers (FSRs) are often used in stream cipher for pseudorandom bit generation that we discuss in detail in Chapter 2.

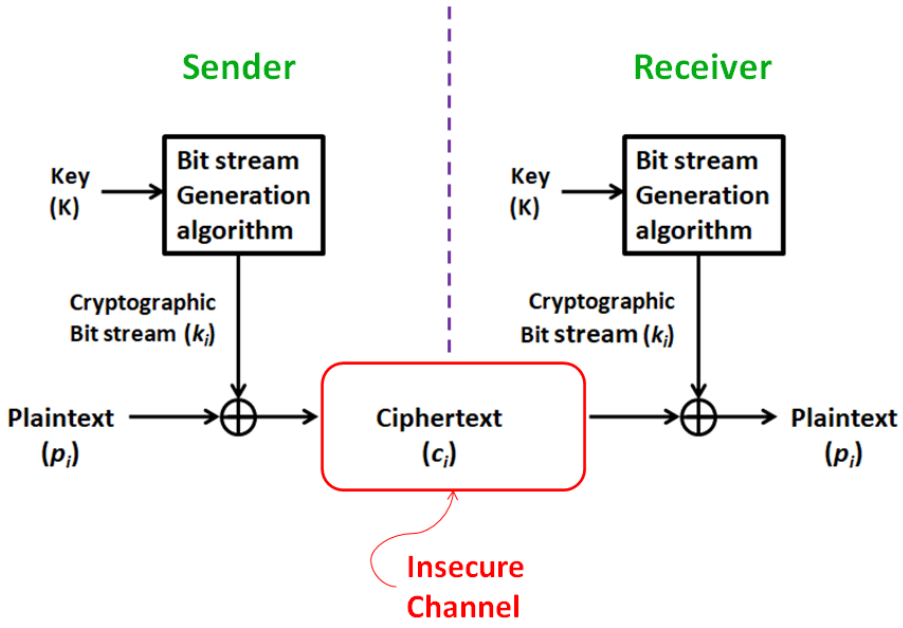

Figure 1.2: Stream Cipher

Stream ciphers are commonly classified as being synchronous or self-synchronizing. A synchronous stream cipher is a finite state machine for which the keystream is generated from a key, but independent of the plaintext message and the ciphertext. This dissertation discusses only the synchronous stream ciphers.

1.2 Motivation

In stream ciphers, LFSRs are popularly used as one of the suitable primitives in the keystream generators due to their simple structure, good statistical properties, and for low cost hardware implementations. However, the bitstream generated by LFSRs has low linear complexity, which makes them vulnerable due to the Berlekamp-Massey algorithm [35]. To overcome such an attack, it is necessary that the bitstream must have high linear complexity. There are several techniques available in the literature to generate bitstream with high linear complexity. Nonlinear feedback shift registers, nonlinear combination generators, nonlinear filter generators, and clock-controlled generators are some popular techniques [37].

The above mentioned generators use only bit-oriented finite state machine (FSM) and all those bit-oriented pseudorandom number generators (PRNGs) do not take advantage of the available modern word-based processors. Instead, the word-oriented primitives like word-based LFSRs called MRMMs [5, 6, 54], lagged Fibonacci generators (LFGs) [14] and Xorshift RNGs [7]) are preferred to take this advantage. Like LFSR, the major drawback of MRMM is that it possesses low linear complexity. This motivates us to explore word-oriented, processor-friendly FSRs with large period and linear complexity along with good randomness properties.

1.3 Objectives of the Thesis

The main objective of this thesis is the design and analysis of several word-oriented pseudo-random number generators.

1.4 Organisation of the Thesis

The rest of this thesis is organized as follows:

Chapter 2: This chapter discusses both bit-oriented and word-oriented FSRs. We introduce a generalized form of the feedback function for word-oriented feedback shift registers

(WFSRs) along with some special cases. A necessary and sufficient condition for nonsingular WFSR is also provided. The linear WFSR called MRMM is discussed and some results pertaining to the keystream generated by it are presented. We discuss two algorithms for generation of efficient primitive MRMMs and perform a comparative analysis.

Chapter 3: In this chapter, the concept of both bit-oriented shrinking and self-shrinking generators are introduced in the case of MRMMs. In both cases, the lower bound for the period as well as for the linear complexity of the bitstream are shown to be exponential. Further, we have investigated some results on the periodicity and statistical properties of the bitstream in self-shrinking MRMMs. This helps to find and prove the exact period of the bitstream produced by the self-shrinking generators.

Chapter 4: Different word-based cascade systems along with their periods are studied in this chapter. We present experimental results on the avalanche property on the states of the cascade systems, also the statistical analysis of the generated keystream by these cascade systems is discussed. Finally, we present a cryptanalytic attack on the cascade systems and suggest its countermeasure.

Chapter 5: We extend the idea of bit-oriented alternating step generators and nonlinear combination generators to the respective word-oriented generators in this chapter. Further, we study their periodicity and linear complexity.

Chapter 6: In this chapter, we conclude the thesis with a summary of our work and related open problems.

Chapter 2

Feedback shift registers

In the stream cipher design, feedback shift registers (FSRs, see [26, 32, 37, 50]) serve as one of the important basic building blocks. Both linear feedback shift registers (LFSRs) and nonlinear feedback shift registers (NLFSRs) [19, 26, 37, 46] are quite useful in the design of symmetric-key crypto algorithms, especially for stream cipher design. LFSRs have a simple structure with an exponential period, good statistical properties, and also low cost of hardware implementations. However, they have very low linear complexity, which can be computed efficiently using the well-known Berlekamp-Massey algorithm [35]. Because of this algorithm, it is possible to find the initial states of an LFSR of order n along with the feedback polynomial from any $2n$ consecutive output bits. To overcome this attack, it is necessary that the bitstream must have high linear complexity. The linear complexity of a bitstream is defined as the shortest length LFSR which reproduces the given bitstream. There are several techniques available in the literature to generate bitstream with high linear complexity. Nonlinear feedback shift registers, nonlinear combination generators, nonlinear filter generators, and clock-controlled generators are some well-known techniques [37]. In nonlinear combination generators and nonlinear filter generators, the component LFSRs are clocked regularly and the nonlinearity is introduced by a nonlinear boolean function. However, in a clock-controlled generator [15, 36], nonlinearity is introduced into the keystream by controlling the output of one LFSR by the other LFSR/LFSRs. In [37, Section 6.3.3], two

clock-controlled generators i.e., the alternating step generator [27] and the shrinking generator [15] are described. The alternating step generator was introduced by C.G. Gunther, where three LFSRs are used for keystream generation and the output bit of the first LFSR controls the clocking of the other two LFSRs.

On the other hand, two LFSRs are used in the case of the shrinking generator [15] proposed by Coppersmith, Krawczyk, and Mansour. In this generator, both the LFSRs are always clocked together. As the name suggests, the bits are produced by shrinking the output sequence of one LFSR under the control of the second LFSR. The output bit of the second LFSR is selected as a keystream bit if the control bit (i.e., the output bit of the first LFSR) is 1, otherwise, it is discarded. Another generator used for achieving the high linear complexity is the self-shrinking generator [36] proposed by Meier and Staffelbach. Here only one LFSR is used for the keystream generation. For $k = 0, 1, \ldots$, if the $(2k+1)^{th}$ output bit of the LFSR is 1, then the $(2k+2)^{th}$ output bit is added to the keystream bit, otherwise the $(2k+2)^{th}$ bit is discarded. In [36], it is shown that the shrinking generator can be realized as a self-shrinking generator and vice versa.

The above mentioned generators use only bit-oriented finite state machine (FSM). If the order of the FSM is n, then it needs n shifting and some operations for feedback value computation in each cycle. Note that in the case of processors, the cost for shifting 1-bit is the same as that of shifting an m-bit word, where m is the bit size of the machine processor. In most modern computer systems, m is 16 or 32, or 64. So the bit-oriented pseudorandom number generators (PRNGs) like LFSR, shrinking generators and others do not take advantage of the available modern word-based processors. Instead, the word-oriented primitives like word-based LFSRs called MRMMs [5, 6, 43, 51, 54], lagged Fibonacci generators (LFGs) [14] and Xorshift RNGs [7]) are preferred to take this advantage.

The bit or word sequences generated by these FSMs have a large period which is exponential in terms of its order n and word size m. It has also very good statistical properties except for low linear complexity. Thus, in order to increase the linear complexity of the bit sequences generated by word-based FSMs, similar kind of methods can be used as in the

case of bit-oriented LFSRs. In this dissertation, we have extended some of those ideas of the bit-oriented generator to the respective word-oriented generator. Then we investigate the period and linear complexity of those generators along with their randomness properties. The word-oriented shrinking generators and word-oriented cascade systems are discussed in Chapter 3 and Chapter 4, respectively. In Chapter 5, we cover word-oriented alternate step generators and nonlinear combination generators.

Now we introduce some notations, definitions, and results concerning primitive LFSRs and MRMMs.

2.1 Preliminaries

Since we study properties of bitstream generated by several pseudorandom bit generators, our base field is the binary field, i.e., Galois field of two elements denoted by $\mathbb{F}_2 = \{0, 1\}$. Denote by $\mathbb{F}_2^m$ the m-dimensional vector space over $\mathbb{F}_2$. The symbol $\oplus$ denotes the addition in $\mathbb{F}_2$ and the symbol $+$ is the addition in the residue class ring $\mathbb{Z}/2^m\mathbb{Z}$. The set of all $m \times m$ matrices with entries in $\mathbb{F}_2$ is denoted by $M_m(\mathbb{F}_2)$ and $\mathrm{GL}_m(\mathbb{F}_2)$ represents the set of all $m \times m$ invertible matrices in $M_m(\mathbb{F}_2)$. The $m \times m$ identity matrix is denoted by I_m. For any matrix $C \in M_m(\mathbb{F}_2)$, $\det(C)$ denotes its determinant. Since two finite dimensional vector spaces $\mathbb{F}_{2^m}$ and $\mathbb{F}_2^m$ are isomorphic [40], the element of $\mathbb{F}_{2^m}$ may be thought of as a column vector of size m over $\mathbb{F}_2$ and hence, for any $\mathbf{s} \in \mathbb{F}_{2^m}$ and $C \in M_m(\mathbb{F}_2)$ the matrix-vector multiplication $C\mathbf{s}$ is a well-defined element of $\mathbb{F}_2^m$. We use bold letter variable if it belongs to $\mathbb{F}_{2^m}$ i.e., $\boldsymbol{x} \in \mathbb{F}_{2^m}$, whereas the normal letter x as a bit i.e., $x \in \mathbb{F}_2$.

We denote by $\mathcal{MP}_m[\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}] = M_m(\mathbb{F}_2)[\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}]/(\boldsymbol{x}_0{}^2 \oplus \boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}{}^2 \oplus \boldsymbol{x}_{n-1})$, the set of all multivariate polynomial $F(.)$ in n variables $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}$ with coefficients in $M_m(\mathbb{F}_2)$ such that $F(\mathbf{0}, \ldots, \mathbf{0}) = \mathbf{0}$ and each $\boldsymbol{x}_i \in \mathbb{F}_{2^m}$. If $F \in \mathcal{MP}_m[\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}]$, then it can be expressed as follows.

$$F(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}) = \sum_{I \in P(N)} C_I \prod_{k \in I} \boldsymbol{x}_k \tag{2.1}$$

where $P(N)$ denotes the power set of $N = \{0, \ldots, n-1\}$ and $C_I \in M_m(\mathbb{F}_2)$. For example, if $N = \{0, 1, 2\}$ then the general expression of $F(\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2) = C_0\boldsymbol{x}_0 + C_1\boldsymbol{x}_1 + C_2\boldsymbol{x}_2 + C_{01}\boldsymbol{x}_0\boldsymbol{x}_1 + C_{02}\boldsymbol{x}_0\boldsymbol{x}_2 + C_{12}\boldsymbol{x}_1\boldsymbol{x}_2 + C_{012}\boldsymbol{x}_0\boldsymbol{x}_1\boldsymbol{x}_2$. In Eq. (2.1), $\mathbf{X} = \prod_{k\in I} \boldsymbol{x}_k$ is computed first, which returns an m-bit number and then $C_I\boldsymbol{X}$ is calculated using matrix-vector multiplication. Here the product $\prod$ can be either field multiplication or modular integer multiplication $*$ or bitwise AND operation &. Similarly, the sum $\sum$ is either modular integer addition $+$ or bitwise XOR operation $\oplus$. As field multiplication is an expensive operation compared to the other two, so we only focus on the other two multiplication operations $*$ and &. The algebraic degree of F denoted as $deg(\mathrm{F})$ can be defined as $\max\{|I| : C_I \neq 0\}$, where $|I|$ denotes the size of I.

2.2 Word-oriented FSR

An n-stage word-oriented FSR (WFSR) is an FSR where each stage stores a word of size m-bits. A state of a WFSR is a vector $(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1})$, where $\boldsymbol{x}_i$ indicates the content of stage i. Let $\mathbf{S}_0 = (\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_{n-1})$ be the initial states of WFSR and Suppose $F \in \mathcal{MP}_m[\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}]$ is the feedback function for WFSR. At every clock pulse, there is a transition from the state $\mathbf{S}_t = (\mathbf{s}_t, \mathbf{s}_{t+1}, \ldots, \mathbf{s}_{t+n-1})$ to the state $\mathbf{S}_{t+1} = (\mathbf{s}_{t+1}, \mathbf{s}_{t+2}, \ldots, \mathbf{s}_{t+n})$ for integer $t \geq 0$, where $\mathbf{s}_{n+t} = F(\mathbf{S}_t)$. After consecutive clock pulses, the WFSR outputs a word sequence $[\mathbf{s}] = \{\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_n, \ldots\}$. The sequence $[\mathbf{s}]$ is *ultimately periodic* if there are integers r, n_0 with $r \geq 1$ and $n_0 \geq 0$ such that $\mathbf{s}_{j+r} = \mathbf{s}_j$ for all $j \geq n_0$. If $n_0 = 0$, then $[\mathbf{s}]$ is said to be periodic and in such case, the least positive integer r is called the *period* of the sequence $[\mathbf{s}]$. A WFSR is called a linear WFSR if its feedback function F is linear i.e., $deg(F) = 1$ and a nonlinear WFSR otherwise.

Before discussing WFSR further, we would like to present three WFSRs through the following example.

Example 2.2.1. Consider a three-stage WFSR with word size 2 called WFSR_1. The feed-

back functions of WFSR_1 is $F_1(\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2) = C_0\boldsymbol{x}_0 + C_{12}\boldsymbol{x}_1\boldsymbol{x}_2$ where $C_0 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ and $C_{12} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. Here modular integer addition and modular integer multiplication are used in the feedback value computation. Using the feedback function, one can compute the next state of a given state. All possible states for a WFSR of length 3 with word size 2 are 4^3 states. Fig. 2.1 shows a part of the state diagram of WFSR_1 with initial state $(1, 2, 3)$. For abuse of notation, we use 123 for the state $(1, 2, 3)$. Here each number is converted to a vector and vice-versa during the feedback value calculation. For this example, $0 = [0, 0]^t, 1 = [0, 1]^t, 2 = [1, 0]^t$ and $3 = [1, 1]^t$. It is easy to verify that the next state of 123 is 231 as $F(1, 2, 3) = 1$. In this case, the state 231 is called the successor of 123, and 123 is called the predecessor of 231. Consider WFSR_2, another three-stage WFSR with word size 2 with feedback function $F_2(\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2) = C_0\boldsymbol{x}_0 + C_{12}\boldsymbol{x}_0\boldsymbol{x}_1\boldsymbol{x}_2$ where the matrix coefficients C_0 and C_{12} are same as in WFSR_1. Fig. 2.2 shows part of the state diagram of WFSR_2 with the same initial state 123. Suppose WFSR_3 is a three-stage WFSR with word size 2 having feedback function $F_3(\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2) = C_0\boldsymbol{x}_0 + C_{12}(\boldsymbol{x}_1 \& \boldsymbol{x}_2)$. The feedback function is the same as in WFSR_1 except the operation & is used in place of $*$. Fig. 2.3 shows part of the state diagram of WFSR_3 with the same initial state 123.

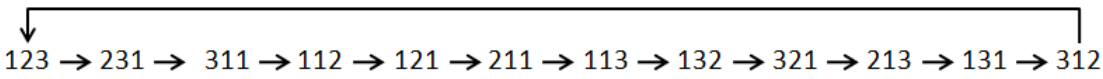

Figure 2.1: The part of state diagrams of WFSR_1 with initial state 123

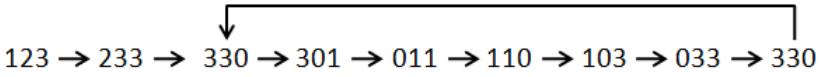

Figure 2.2: The part of state diagrams of WFSR_2 with initial state 123

Unlike to WFSR_1 and WFSR_3, distinct vectors do not have distinct successors in WFSR_2. The states 233 and 330 of WFSR_2 have common successors 301 i.e., 301 does not have unique

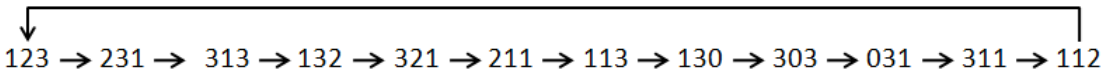

Figure 2.3: The part of state diagrams of WFSR_3 with initial state 123

predecessors. A WFSR (or its feedback function) is called nonsingular if each state has a unique predecessor i.e., its state diagram consists of disjoint cycles. In the case of bit-oriented FSR, it is shown in [26] that the FSR is nonsingular if and only if its feedback function is of type

$$f(x_0, x_1, \ldots, x_{n-1}) = x_0 \oplus g(x_1, x_2, \ldots, x_{n-1}) \tag{2.2}$$

where the function $g(.)$ does not depend on the variable x_0. Is there any necessary and sufficient condition for WFSR to be nonsingular? It is easy to show that the WFSR is nonsingular if its feedback function is of the form as in Eq. (2.2). There are other WFSRs whose feedback functions are in different forms as WFSR_1 of Example 2.2.1. In the following, we provide a necessary and sufficient condition for which WFSR is nonsingular.

Theorem 2.2.2. *An n-stage WFSR is nonsingular if and only if its feedback function* $F \in \mathcal{MP}_m[\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}]$ *can be represented as*

$$F(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}) = f_0(\boldsymbol{x}_0) \odot f_1(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1}) \tag{2.3}$$

where f_0 *is a bijective function and* f_1 *is an arbitrary function from* $\mathbb{F}_{2^m}^{n-1} \to \mathbb{F}_{2^m}$. *The operation* $\odot$ *is either* $+$ *or* $\oplus$.

Proof. Suppose WFSR is nonsingular, then distinct vectors have distinct successors. If the feedback function F does not contain any $\boldsymbol{x}_0$ term, then it is easy to get two distinct vectors having a common successor. Thus, the expression of F must contains $\boldsymbol{x}_0$ term therefore, $F(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1})$ can be expressed as $f_0(\boldsymbol{x}_0) \odot \boldsymbol{x}_0 f_2(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1}) \odot f_1(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1})$ where $f_0(\boldsymbol{x}_0)$ contains all terms of $\boldsymbol{x}_0$ and both the functions f_1 and f_2 are independent of $\boldsymbol{x}_0$. Also f_2 does not have any constant term as all the $\boldsymbol{x}_0$ are in f_0 and hence $f_2(0, \ldots, 0) = 0$. If f_0 is not bijective, then there exists $\boldsymbol{x}_0', \boldsymbol{x}_0''$, such that $F(\boldsymbol{x}_0', 0, \ldots, 0) = F(\boldsymbol{x}_0'', 0, \ldots, 0)$, where $\boldsymbol{x}_0' \neq \boldsymbol{x}_0''$. Thus $(\boldsymbol{x}_0', 0, \ldots, 0)$ and $(\boldsymbol{x}_0'', 0, \ldots, 0)$ have common successor. This is a

contradiction and hence f_0 is bijective. Next, to prove that $f_2 = 0$, suppose f_2 has some nonzero terms, then $f_2(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1})$ must contain a term of $\boldsymbol{x}_k$ for $k \neq 0$. In this case, $F(0, 2, \ldots, 2) = F(2^{m-1}, 2, \ldots, 2)$. This is not possible as WFSR is nonsingular and therefore $f_2(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1}) = 0$. This proves the necessary part.

To prove WFSR is nonsingular, we need to show that distinct vectors have distinct successors. If the two vectors $(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1})$ and $(\boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{n-1})$ differ in any component other than the first, then their successors $(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)$ and $(\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n)$ are still distinct. Thus, WFSR is nonsingular if and only if $(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1})$ and $(\boldsymbol{y}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1})$ have distinct successors for $\boldsymbol{x}_0 \neq \boldsymbol{y}_0$. Suppose WFSR is singular, then there exists $\boldsymbol{x}_0 \neq \boldsymbol{y}_0$ such that $f(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n-1}) = f(\boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{n-1})$. This implies that $f_0(\mathbf{x}_0) = f_0(\mathbf{y}_0)$ for $\boldsymbol{x}_0 \neq \boldsymbol{y}_0$. This is a contradiction as f_0 is bijective. This completes the proof.

□

Corollary 2.2.3. Bit-oriented FSRs are nonsingular if and only if the feedback function $f(x_0, x_1, \ldots, x_{n-1}) = x_0 \oplus g(x_1, x_2, \ldots, x_{n-1})$.

Proof. In the case of bit-oriented FSR, $m = 1$. Thus, there are two bijective functions from $\mathbb{F}_2 \longrightarrow \mathbb{F}_2$. In both cases, the feedback function f is expressed in the desired form. □

Note that if $deg(F) = 1$, then the feedback function F in Eq. (2.1) satisfies the criteria of Theorem 2.2.2 and so the feedback function is always nonsingular.

2.3 Some special cases of WFSR

In this section, we discuss some well-known FSRs as the special case of WFSR by putting different restrictions in Eq. (2.1).

2.3.1 Bit-oriented FSRs

The WFSR becomes a bit-oriented FSR when $m = 1$. In this case, the $m \times m$ matrix coefficient C_k of Eq. (2.1) becomes a scalar $c_k \in \mathbb{F}_2$. If $deg(F) = 1$, then $F(x_0, \ldots, x_{n-1}) =$

$\bigoplus_{k=0}^{n-1} c_k x_k$. This expression is the feedback function of well known FSR called LFSR [26] and the theory of LFSRs is well-developed. If the feedback polynomial of LFSR is primitive, then it generates maximal periodic bitstreams for any nonzero initialization of the LFSR state and these bitstreams satisfy most of the statistical properties.

If $deg(F) > 1$ then bit-oriented FSR becomes NLFSR. There is no efficient way to find a feedback function such that its corresponding NFSR can generate bitstream with guaranteed large periods. Also, it is hard to determine the periods of NFSR sequences for a given feedback function. However, Golomb [26] proved a necessary and sufficient condition for all the sequences produced by an NFSR to be periodic is that its feedback function is nonsingular. The maximum period that can be achieved by an n-stage NFSR is 2^n and in this case, it is known as de Bruijn sequences. Unlike LFSRs, the theory of NFSRs is not well-investigated due to its complexity.

2.3.2 MRMM

If $deg(F) = 1$, word size $m > 1$ and $\oplus$ is used in place of $\odot$, then the feedback function as expressed in Eq. (2.1) becomes $F(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}) = \bigoplus_{k=0}^{n-1} C_k \boldsymbol{x}_k$ where $C_0, C_1, \ldots, C_{n-1} \in M_m(\mathbb{F}_2)$. This is the general expression for the feedback function of MRMM [40, 41, 42]. For a periodic word sequence $[\mathbf{s}]$, it is always possible to have a relation called linear recurring relation (LRR) among the elements as

$$\mathbf{s}_{i+n} = \sum_{k=0}^{n-1} C_k \mathbf{s}_{i+k} \quad i \geq 0. \tag{2.4}$$

The Eq. (2.4) for $[\mathbf{s}]$ can be mapped to an MRMM of order n over $\mathbb{F}_{2^m}$ which we denote as MRMM(m, n). The sequence $[\mathbf{s}]$ is referred to as the sequence generated by the MRMM(m, n) and the polynomial associated with Eq. (2.4) denoted as $M(x) = I_m x^n - C_{n-1}x^{n-1} - \cdots - C_1 x - C_0$ with matrix coefficients is called the *matrix polynomial* of the MRMM(m, n). The determinant of the matrix polynomial i.e., $|M(x)|$ is a polynomial of degree mn over $\mathbb{F}_2$ and it is known as the characteristic polynomial of the MRMM. Again, for a matrix polynomial

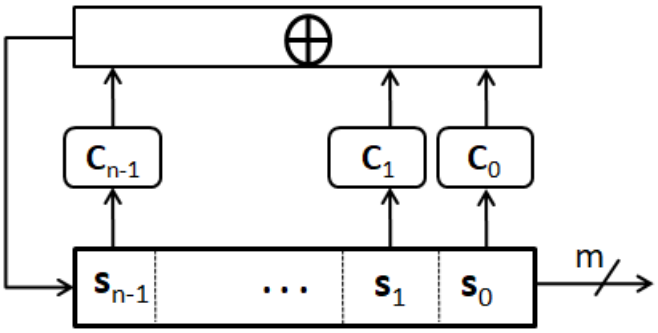

Figure 2.4: The n^{th} order MRMM over $\mathbb{F}_{2^m}$

$M(x) \in M_m(\mathbb{F}_2)[x]$, it is always possible to associate an (m, n)-block companion matrix $T \in M_{mn}(\mathbb{F}_2)$ of the following form

$$T = \begin{pmatrix} \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} & C_0 \\ I_m & \dots & \mathbf{0} & \mathbf{0} & C_1 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \mathbf{0} & \dots & I_m & \mathbf{0} & C_{n-2} \\ \mathbf{0} & \dots & \mathbf{0} & I_m & C_{n-1} \end{pmatrix}, \tag{2.5}$$

where I_m denotes the $m \times m$ identity matrix over $\mathbb{F}_2$, while $\mathbf{0}$ indicates the zero matrix in $M_m(\mathbb{F}_2)$. It is easy to see that the characteristic polynomial of $T = |T - xI|$ is the same as the characteristic polynomial of the MRMM.

Note that $(1, n)$-block companion matrix is the same as the transition matrix of the polynomial $f(x) = x^n + c_{n-1}x^{n-1} + \ldots + c_1x + c_0 \in \mathbb{F}_2[x]$ and we denote the transition

matrix as

$$A = \begin{pmatrix} 0 & \dots & 0 & 0 & c_0 \\ 1 & \dots & 0 & 0 & c_1 \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & \dots & 1 & 0 & c_{n-2} \\ 0 & \dots & 0 & 1 & c_{n-1} \end{pmatrix}, \tag{2.6}$$

The following proposition [22, Proposition 4.2] gives some basic facts about the MRMMs.

Proposition 2.3.1. For the sequence $[\mathbf{s}]$ generated by the MRMM of order n over $\mathbb{F}_{q^m}$, we have

(i) $[\mathbf{s}]$ is ultimately periodic and its period is no more than $q^{mn} - 1$;

(ii) if C_0 is nonsingular, then $[\mathbf{s}]$ is periodic. Conversely, if $[\mathbf{s}]$ is periodic whenever the initial state is of the form $(b, 0, \dots, 0)$, where $b \in \mathbb{F}_{q^m}$ with $b \neq 0$, then C_0 is nonsingular.

An MRMM(m, n) is called primitive if, for any choice of the nonzero initial state, the sequence generated by that MRMM is periodic with period $2^{mn} - 1$. It is shown in [5] that if the MRMM(m, n) is primitive if and only if the determinant of the matrix polynomial $M(x)$ of MRMM(m, n) is a primitive polynomial of degree mn over $\mathbb{F}_2$.

In Eq. (2.4), it is clear that to produce m-bit word, the MRMM needs n state shifting operations along with a feedback value computation. Here, in the case of feedback value calculation, it needs several matrix-vector multiplications and XOR operations. In general, the computational complexity for matrix-vector multiplication of order m is $O(m^2)$. However, the matrices used in the search algorithm [54] and the construction algorithm [5] are of special kind of matrices for the generation of primitive MRMMs. Here the matrix-vector multiplication complexity is $O(m)$. Therefore, both the search algorithm and the construction algorithm produce efficient primitive MRMMs. The search algorithm for the primitive MRMM, first constructs the matrix polynomial of order n, where each coefficient is an $m \times m$

matrix. Then checks the primitiveness of the characteristic polynomial $g(x)$ which is the determinant of the constructed matrix polynomial. But in the construction algorithm for the primitive MRMM, it first finds a primitive polynomial $g(x)$ of degree mn and then using n^{th} order Horner's form, it constructs the (m,n)-block companion matrix T as given in Eq. (2.5). Then from T, the matrix coefficients C_k are derived.

Let the word sequence $[\mathbf{s}]$ be generated by a primitive MRMM(m,n). For $j = 1, 2, \ldots, m$, consider $[s^{(j)}] = \{s_0^{(j)}, s_1^{(j)}, \ldots, \}$ be the bit sequence, where $s_k^{(j)}$ is the j^{th} least significant bit (lsb) of m-bit word $\mathbf{s}_k$. Then, $\mathbf{s}_k = (s_k^{(m)}, \ldots, s_k^{(2)}, s_k^{(1)}) \in \mathbb{F}_2^m$, for $k = 0, 1, \ldots,$ and each ${s_k}^{(j)} \in \mathbb{F}_2$. Now, we have the following result due to Niederreiter [40, Lemma-1].

Lemma 2.3.2. Let $[\mathbf{s}]$ be an arbitrary recursive vector sequence with the characteristic polynomial $g(x) \in \mathbb{F}_2[x]$ of degree mn and each vector is m-bit wide. Then, the bit sequence $[s^{(j)}]$ for $1 \leq j \leq m$ will be a linear recurring sequence in $\mathbb{F}_2$ with the same characteristic polynomial g(x). So if $g(x)$ is primitive, then the word sequence $[\mathbf{s}]$ and each bit sequence $[s^{(j)}]$ attain the maximal period i.e., $(2^{mn} - 1)$.

Remarks 2.3.3. Note that in the case of a primitive MRMM(m,n), each bit sequence $[s^{(j)}]$ has period $(2^{mn} - 1)$ and is a circular shifted version of any other bit sequence $[s^{(k)}]$ for $1 \leq j, k \leq m$.

The following corollary trivially follows from Lemma 2.3.2 and gives the component-wise linear complexity of the sequences generated by primitive MRMM.

Corollary 2.3.4. Let

$$\mathbf{s}_i = \left(s_i^{(m)}, \ldots, s_i^{(1)}\right) \in \mathbb{F}_2^m \simeq \mathbb{F}_{2^m} \quad i = 0, 1, \ldots,$$

be a sequence of words generated by a primitive MRMM of order n over $\mathbb{F}_{2^m}$. Then for each $1 \leq j \leq m$, the linear complexity of the j^{th} coordinate sequence $[s^{(j)}] = \{s_0^{(j)}, s_1^{(j)}, \ldots, \}$ over $\mathbb{F}_2$ is mn.

2.3.3 LFG

If $deg(F) = 1$, word size $m > 1$ and $+$ is used in place of $\odot$, then $F(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1}) = C_0\boldsymbol{x}_0 + C_1\boldsymbol{x}_1 + \ldots + C_{n-1}\boldsymbol{x}_{n-1}$. If all $C_k \in \{I_m, 0\}$, then $F(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1})$ is the general expression of the feedback function of an additive LFG. If $f(x) = x^n - c_{n-1}x^{n-1} - \ldots - c_1x - c_0$ is the corresponding characteristic primitive polynomial of LFG, then it is proved by R. P. Brent [10] that the period of the recurrence relation is $2^{m-1}(2^n - 1)$ for all $m \geq 1$ if and only if $f(x)^2 + f(-x)^2 \neq 2f(x^2)$ (modulo 8) and $f(x)^2 + f(-x)^2 \neq 2(-1)^n f(-x^2)$ (modulo 8). Using matrix theory, George Marsaglia *et al.* [34] also have shown, for the transition matrix A corresponding to the characteristic primitive polynomial, the recurrence relation has the maximal period $(2^n - 1)2^{m-1}$ for all $m \geq 1$ for every initial state with at least one odd number if and only the order of A satisfies the following

- order $j = 2^n - 1$ in the group of nonsingular matrices for mod 2,
- order $2j$ for mod 2^2,
- order $4j$ for mod 2^3.

When an LFG has a maximal period, it is known as a primitive LFG. In [14], it is shown that an n^{th} order LFG can be visualized as a scrambler of m binary LFSRs. In this case, each LFSR is of length n with the same feedback function, and for $k = 1, 2, \ldots, m$; the k^{th} LFSR corresponds to the k^{th} least significant bit (lsb) of each of the m-bit state of the LFG. Except 1^{st} lsb LFSR, all other $(m-1)$ LFSRs run in scrambler mode, i.e., feedback value is modified by an external bit value. In fact, the i^{th} carry bit of feedback value of k^{th} LFSR affects the feedback bit value of $(i+k)^{th}$ LFSR, for $k = 1, 2, \ldots, (m-1)$. The jumping concept in LFG is also introduced in [14]. It has been known that in the case of LFSR and σ-LFSR a jump always exists if the characteristic polynomial is primitive. However, the primitivity of the characteristic polynomial is not sufficient for an LFG to have a jump. An LFG may jump in the same cycle or jump across different cycles depending on the characteristic polynomial. Moreover, the period of the jump LFG and the hyper jump LFG is the same as that of the LFG. It is proved that an LFG jumps across the same cycle if and only if the associated

characteristic polynomial is primitive and of the form $x^n - x - 1$, otherwise it will have a hyper jump.

2.4 Algorithms for efficient primitive MRMM

For a general $m \times m$ matrix, the number of multiplication in the matrix-vector multiplication is $O(m^2)$. However, there are special sets of matrices for which matrix-vector multiplication needs $O(m)$ operations. We say MRMM is efficient if its feedback calculation takes $O(m)$ operations for matrix-vector multiplication. First, we discuss the search algorithm for efficient MRMM.

2.4.1 Search algorithm

The primitive MRMMs generated by the search algorithm [54, Algorithm 1] proposed by Zeng *et al.* have a very efficient and fast software implementation. These MRMMs use some special linear transformation, especially the word-based operations provided by modern processors. The list of those operations is left (right) shift operation, combination of left shift, right operations, circular rotation, AND, OR, and XOR.

1. Left shift operation $\mathbf{L}$ and right shift operation $\mathbf{R}$.
   For $\boldsymbol{x} = (x_0, \ldots, x_{m-1})$, the left shift operator $\mathbf{L}$ is defined as $\mathbf{L}(\boldsymbol{x}) = (x_1, \ldots, x_{m-1}, 0)$ and the right shift operator $\mathbf{R}$ is defined as $\mathbf{R}(\boldsymbol{x}) = (0, x_0, \ldots, x_{m-2})$. The matrix

representation of the operator $\mathbf{L}$ is as follows

$$\mathbf{L} = \begin{pmatrix} 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 0 & 0 \end{pmatrix}_{m \times m}$$

where as the matrix representation of $\mathbf{R}$ is the transpose of the matrix $\mathbf{L}$, i.e.,

$$\mathbf{R} = \mathbf{L}^T = \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}_{m \times m}$$

2. Combination of left shift and right shift operation $\bigsqcup_{s,t}$.

   For given positive integers $0 < s, t < m$, the operator $\bigsqcup_{s,t} = \mathbf{L}_s + \mathbf{R}_t$, i.e., it is an $m \times m$ matrix having all entries are 1 in s^{th} upper sub-diagonal and t^{th} lower sub-diagonal.

For example

$$\bigsqcup_{m-2,1} = \begin{pmatrix} 0 & 0 & \cdots & 1 & 0 \\ 1 & 0 & \cdots & 0 & 1 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}_{m\times m}$$

It is shown [54, Lemma 1] that $\bigsqcup_{s,t}$ is invertible if and only if $(s+t) \mid m$.

3. Circular rotation operation σ and β.

   The right circular rotation σ is defined as

   $\sigma(x) = \sigma(x_0, x_1, ..., x_{m-1}) = (x_{m-1}, x_0, x_1, ..., x_{m-2})$. The matrix representation of σ is as follows

$$\sigma = \begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}_{m\times m}$$

   Similarly, the left circular rotation β is defined as the transpose of σ , i.e., $\beta(x) =$

$\beta(x_0, x_1, ..., x_{m-1}) = (x_1, ..., x_{m-1}, x_0)$ and the matrix form of β is

$$\beta = \sigma^T = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \cdots & 0 \end{pmatrix}_{m \times m}$$

Note that $\sigma\beta = \beta\sigma = I_m$.

4. AND operation Λ_γ.

   Let $\{\alpha_i\}_{i=0}^{m-1}$ be the basis for $\mathbb{F}_{2^m}$, then each $\gamma \in \mathbb{F}_{2^m}$ can be expressed as $\gamma = \sum_{i=0}^{m-1} c_i \alpha_i$, where $c_i \in \mathbb{F}_2$. Then, for each $x = (x_0, x_1, ..., x_{m-1}) \in \mathbb{F}_{2^m}$, Λ_γ is defined as $\Lambda_\gamma(x) = \sum_{i=0}^{m-1} c_i \alpha_i x_i$. The matrix representation of Λ_γ is given below.

$$\Lambda_\gamma = \begin{pmatrix} c_0 & 0 & \cdots & 0 & 0 \\ 0 & c_1 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & c_{m-1} \end{pmatrix}_{m \times m}$$

Using above four operators, two sets W and V are defined for the search algorithm [54] as follows.

$W = \{\Lambda_\gamma, \mathbf{L}^j, \mathbf{R}^k, \sqcup_{s,t} \mid \gamma \in \mathbb{F}_{2^m}, \gamma \neq 0 \text{ or } 2^m - 1; 0 < j, k, s, t < m\}$

$V = \{\sigma^k, \sqcup_{s,t} \mid -m < k < m; 0 < s, t < m, (s+t) | m\}$

Note that the elements of the set V are invertible. It is clear that the primitive MRMMs with coefficients from V and W are suitable for fast software implementation. The steps of the search algorithm [54] is given in Algorithm 1.

Algorithm 1 : Search algorithm for primitive MRMM

Input: Order of the MRMM n and the word size m
Output: An efficient primitive MRMM of order n over $\mathbb{F}_{q^m}$.

1: Randomly choose the coefficients of matrix polynomial $M(x) = I_m X^n + C_{n-1}X^{n-1} + \cdots + C_1 X + C_0$ having $C_0 \in V$ and $C_1, C_2, \ldots, C_{n-1} \in V \bigcup W$.
2: Compute the determinant $g(x) = |M(x)|$, which will be a polynomial of degree mn over $\mathbb{F}_2$.
3: If $g(x)$ is primitive, then $M(x)$ will generate a primitive MRMM, otherwise go to Step 1

Both software and hardware implementations of the MRMMs obtained through the search algorithm are quite efficient. Yet for larger values of m and n, the search algorithm becomes sluggish to produce primitive MRMM as it takes large number of attempts due to exponential increase in search space size. In the following section, we have computed the exhaustive search space of the search algorithm.

In order to compute the complexity of the search space of Algorithm 1, the number of elements in both the sets V and W need to be calculated. The size of W is first derived in the following proposition.

Proposition 2.4.1. The size of the set W is $(2^m + m^2 - 3)$.

Proof. Since the operators $\Lambda_\gamma, \mathbf{L}^j, \mathbf{R}^k, \sqcup_{s,t}$ are different for all values of $\gamma \in \mathbb{F}_{2^m}$, $\gamma \neq 0, 2^m - 1$ and $0 < j, k, s, t < m$. So, the size of W is

$$\begin{aligned} |W| &= |\Lambda_\gamma| + |\mathbf{L}^j| + |\mathbf{R}^k| + |\sqcup_{s,t}| \\ &= (2^m - 2) + (m-1) + (m-1) + (m-1)^2 \\ &= (2^m + m^2 - 3). \end{aligned} \tag{2.7}$$

□

To compute the size of V, it is necessary to find the number of invertible σ^k for $0 < k < m$

and $\bigsqcup_{s,t}$ for $0 < s, t < m$. Since $\sigma^m = I_m$, σ^k is invertible for all $0 < k < m$. The following lemma tells the relation between the linear transformation $\bigsqcup_{s,t}$ and σ.

Lemma 2.4.2. For $0 < k < m$, $\sigma^k = \bigsqcup_{k,m-k}$.

Proof. The proof is obvious as

$$\sigma^k = \begin{pmatrix} 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}^k_{m\times m} = \mathbf{L}^k + \mathbf{R}^{m-k} = \bigsqcup_{k,m-k}$$

□

Because of the above results, it is easy to see that $V \subseteq W$. Again, to compute $|V|$, i.e., the size of V, it is sufficient to compute the number invertible $\bigsqcup_{s,t}$ for $0 < s, t < m$. It is proved in [54, Lemma 1] that $\bigsqcup_{s,t}$ is invertible if and only if $(s + t) \mid m$. Therefore to find $|V|$, it is required to count the number of distinct pairs (s, t) for which $(s + t) \mid m$. The following lemma computes this number.

Lemma 2.4.3. If the set of all distinct factors (> 1) of m is $\{d_1, d_2, \ldots, d_k\}$, then the number of distinct pairs (s, t) such that $0 < s, t < m$ and $(s + t)|m$ is $\sum_{l=1}^{k}(d_l - 1)$.

Proof. For each factor, Table 2.1 tells the number of distinct pairs (s, t) such that $(s+t)|m$.

So, the total number of such pairs (s, t) is $((d_1-1)+(d_2-1)+\ldots+(d_k-1)) = \sum_{l=1}^{k}(d_l-1)$. This proves the lemma. □

Corollary 2.4.4. If m is prime, then the number of distinct pairs (s, t) such that $0 < s, t < m$ and $(s + t)|m$ is $(m - 1)$.

Factor of m	All possible pairs (s,t)	Number of pairs
d_1	$(1,d_1-1),(2,d_1-2),\ldots,(d_1-1,1)$	(d_1-1)
d_2	$(1,d_2-1),(2,d_2-2),\ldots,(d_2-1,1)$	(d_2-1)
$\vdots$	$\vdots$	$\vdots$
d_k	$(1,d_k-1),(2,d_k-2),\ldots,(d_k-1,1)$	(d_k-1)

Table 2.1: Distinct pairs (s,t) such that $(s+t)|m$

Proof. The only divisor greater than 1 is m itself. So, the required number of distinct pairs is $(m-1)$. □

Corollary 2.4.5. If $m=2^k$, then the number of distinct pairs (s,t) such that $0<s,t<m$ and $(s+t)|m$ is $(2^{k+1}-k-2)$.

Proof. Since $m=2^k$, the set of possible factors (>1) of m are $\{2,2^2,2^3,\ldots,2^k\}$. So, the total number of such pairs (s,t) is $(1+3+7+\ldots+2^k-1)=\sum_{l=1}^{k}(2^l-1)=(2^{k+1}-k-2)$. □

Proposition 2.4.6. If $m=2^k$, then $|V|$ is $(2m-log_2m-2)$.

Proof. Using Corollary 2.4.5, it is clear that $|V|=(2^{k+1}-k-2)=2m-log_2m-2$. □

Since in most of the operating system, the word size is of the form 2^k, now onwards we have considered $m=2^k$ for some positive integer k. Let $I_mX^n+C_{n-1}X^{n-1}+\cdots+C_1X+C_0$ be the primitive σ-polynomial generated by the search Algorithm 1. Then, $C_0\in V$ and $C_1,C_2,\ldots,C_{n-1}\in V\bigcup W=W$. So, the size of the search space is $|V||W|^{n-1}$. Thus, by the Proposition 2.4.1 and 2.4.6, $(2m-log_2m-2)(2^m+m^2-3)^{n-1}$ is the exhaustive search space size.

2.4.2 Construction of efficient primitive MRMM

A search algorithm [54, Algorithm 1] for generating efficient primitive MRMM was proposed by Zeng *et al.*. Their algorithm first constructs the matrix polynomial by choosing some matrices randomly from a specific set. Then test the primitivity of a polynomial obtained by

computing the determinant of a matrix polynomial. The idea behind the search algorithm is, it first constructs efficient MRMM and then checks for its primitiveness. Can the reverse method is possible i.e., from a given primitive polynomial, can an efficient MRMM be constructed? Yes, it is possible, An algorithm called the construction algorithm for primitive MRMM is given for the same [5]. For this, the notion of generalized Horner's form corresponding to a given polynomial is defined. We then construct an efficient primitive MRMM by using it.

Definition 2.4.7. Let $f(X) = \sum_{i=0}^{d} a_i X^i$ be a polynomial of degree d over $\mathbb{F}_2$. For any given positive integer $n \leq d$, we can find integers m and r such that $d = mn + r$, where $0 \leq r < n$. We express $f(X)$ in the following form

$$f(X) = f_0 + X^n \left(f_1 + X^n \left(f_2 + \cdots + X^n \left(f_{m-1} + X^n f_m\right) \cdots\right)\right), \tag{2.8}$$

where,

$$f_i(X) = \sum_{k=in}^{(i+1)n-1} a_k X^{k-in} \quad \text{for} \quad i = 0, 1, \ldots, (m-1) \quad \text{and} \quad f_m(X) = \sum_{k=mn}^{d} a_k X^{k-mn}.$$

The representation of $f(X)$ in Eq. (2.8) is referred to as n-Horner's form [1] of $f(X)$.

Example 2.4.8. Consider the polynomial $f(X) = 1+X^2+X^5+X^7+X^{10}+X^{11}+X^{12} \in \mathbb{F}_2[X]$ of degree 12. Here $d = 12$. For $n = 3$, we have $m = 4$ and $r = 0$. Then 3-Horner's form of $f(X)$ is given by

$$f(X) = f_0 + X^3(f_1 + X^3(f_2 + X^3(f_3 + X^3 f_4))),$$

where $f_0(X) = (1 + X^2)$, $f_1(X) = X^2$, $f_2(X) = X$, $f_3(X) = (X + X^2)$ and $f_4(X) = 1$.

Corresponding to the n-Horner's form of a given polynomial of degree mn over $\mathbb{F}_2$, we can associate an $m \times m$ matrix as defined below. This matrix would play a crucial role

[1] The 1-Horner's form is indeed the usual Horner's form of a polynomial used for computing the polynomial value with less number of multiplications.

in the construction of an efficient multiple-recursive matrix method of order n over $\mathbb{F}_{2^m}$ for generating pseudorandom vectors.

Definition 2.4.9. Let m and n be positive integers and let $f(X) = \sum_{i=0}^{mn} a_i X^i$ be polynomial of degree mn over $\mathbb{F}_2$. The $m \times m$ matrix

$$\begin{pmatrix} X^n & 0 & 0 & \cdots & 0 & f_0 \\ 1 & X^n & 0 & \cdots & 0 & f_1 \\ 0 & 1 & X^n & \cdots & 0 & f_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & X^n & f_{m-2} \\ 0 & 0 & 0 & \cdots & 1 & f_{m-1} + f_m X^n \end{pmatrix}, \tag{2.9}$$

corresponding to the n-Horner's form Eq. (2.8) of $f(X)$ is referred to as n-Horner's matrix of $f(X)$ and denoted as $H_m(n, f)$.

For each $j = 0, 1, \ldots, n-1$, let C_j denotes the $m \times m$ matrix whose entries are coefficients of X^j in the matrix $H_m(n, f)$. It is easy to see that the matrix $H_m(n, f)$ can be written as

$$H_m(n, f) = I_m X^n + C_{n-1} X^{n-1} + \cdots + C_1 X + C_0, \tag{2.10}$$

provided f is monic, that is, $f_m = 1$. It is clear from Eq. (2.10) that we can associate a multiple-recursive matrix method of order n over $\mathbb{F}_{2^m}$ corresponding to this $m \times m$ matrices $C_0, C_1, \ldots, C_{n-1}$.

It is interesting to note that the matrix C_j for $1 \leq j \leq n-1$ has the following form

$$C_j = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & a_j \\ 0 & 0 & 0 & \cdots & 0 & a_{n+j} \\ 0 & 0 & 0 & \cdots & 0 & a_{2n+j} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & a_{(m-2)n+j} \\ 0 & 0 & 0 & \cdots & 0 & a_{(m-1)n+j} \end{pmatrix} \tag{2.11}$$

whose first $(m-1)$ columns are zero. Moreover, the matrix C_0 has the following form

$$C_0 = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & a_0 \\ 1 & 0 & 0 & \cdots & 0 & a_n \\ 0 & 1 & 0 & \cdots & 0 & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & a_{(m-2)n} \\ 0 & 0 & 0 & \cdots & 1 & a_{(m-1)n} \end{pmatrix}. \tag{2.12}$$

It is due to this special structure of these matrices that we are able to construct an efficient multiple-recursive matrix method. In Section 2.4.3, we shall see in greater detail why such construction is fast and efficient.

The following lemma gives the determinant of the matrix $H_m(n, f)$ and will be used in the sequel.

Lemma 2.4.10. Let $H_m(n, f)$ be n-Horner's matrix corresponding to the polynomial $f(X)$ of degree mn over $\mathbb{F}_2$ as defined in Eq. (2.9). Then $\det(H_m(n, f))$ is equal to $f(X)$.

Proof. Add X^n times the n^{th} row to the $(n-1)^{\text{th}}$ row of the matrix $H_m(n, f)$. This will

remove the X^n in the $(n-1)^{\text{th}}$ row and it will not alter the determinant. Next, add X^n times the new $(n-1)^{\text{th}}$ row to the $(n-2)^{\text{th}}$ row. Continue successively until all of the $X^{n\prime}s$ on the main diagonal has been removed. The result is the matrix

$$\begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & f_0 + X^n\left(f_1 + X^n\left(f_2 + \cdots + X^n\left(f_{m-1} + X^n f_m\right)\cdots\right)\right) \\ 1 & 0 & 0 & \cdots & 0 & f_1 + X^n\left(f_2 + \cdots + X^n\left(f_{m-1} + X^n f_m\right)\cdots\right) \\ 0 & 1 & 0 & \cdots & 0 & f_2 + \cdots + X^n\left(f_{m-1} + X^n f_m\right) \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & f_{m-2} + X^n\left(f_{m-1} + X^n f_m\right) \\ 0 & 0 & 0 & \cdots & 1 & f_{m-1} + X^n f_m \end{pmatrix}$$

which has the same determinant as $H_m(n, f)$. We can clean up the last column by adding to it appropriate multiples of the other columns so as to obtain

$$\det\left(H_m(n,f)\right) = \det\begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & f(X) \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix}.$$

Finally, we can slide the last column to the first by successive column interchanges. We need $(m-1)$ interchanges, and so the determinant changes by $(-1)^{(m-1)}$. Further, if we pull out the negative sign in each of the rows in all except the first row, then the determinant gets multiplied by $(-1)^{(m-1)}$. It follows that the determinant of $H_m(n, f)$ is $(-1)^{2(m-1)}$ times the

determinant of the diagonal matrix diag $(f(X), 1, \ldots, 1)$ and this proves the lemma. $\square$

We can construct an efficient MRMM regardless of whether it's reducible, irreducible, or primitive. However, in view of cryptographic applications, we shall focus only on the construction of efficient primitive MRMM.

It is clear that an MRMM Eq. (2.4) is *primitive* if the characteristic polynomial det $(M(X))$ of its transition matrix T is primitive of degree mn over $\mathbb{F}_2$. We shall denote by MRMMP(m, n), the set of all those block companion matrices in MRMM(m, n) whose characteristic polynomial is primitive and by $\mathcal{P}(d)$, the set of primitive polynomials in $\mathbb{F}_2[X]$ of degree d. Then the *characteristic map*

$$\Psi : M_{mn}(\mathbb{F}_2) \longrightarrow \mathbb{F}_2[X] \quad \text{defined by} \quad \Psi(T) := \det(XI_{mn} - T),$$

if restricted to the set MRMMP(m, n) yields the following map

$$\Psi_P : \text{MRMMP}(m, n) \longrightarrow \mathcal{P}(mn).$$

By using the structure of Horner's matrix, we prove the surjectivity of the map Ψ_P in the following theorem. The proof of this theorem would enable us a way to construct efficient primitive MRMM.

Theorem 2.4.11. *The map* $\Psi_P : \text{MRMMP}(m, n) \longrightarrow \mathcal{P}(mn)$ *is surjective.*

Proof. Let $f(X) = \sum_{i=0}^{mn} a_i X^i \in \mathcal{P}(mn)$. Clearly, f is a monic polynomial i.e. $a_{mn} = 1$. Therefore, as in Eq. (2.10), the n-Horner's matrix $H_m(n, f)$ of $f(X)$ can be expressed in the following form

$$H_m(n, f) = I_m X^n + C_{n-1} X^{n-1} + \cdots + C_1 X + C_0, \tag{2.13}$$

where C_i denotes the $m \times m$ matrix whose entries are coefficients of X^i in the Horner's matrix $H_m(n, f)$. Let $\widetilde{T}$ denote the block companion matrix corresponding to the matrix

polynomial Eq. (2.13). Then by Lemma 2.4.10, it follows that

$$\Psi_P(\widetilde{T}) = \det(XI_{mn} - \widetilde{T}) = \det\left(H_m(n, f)\right) = f(X),$$

as desired. □

Now we present an algorithm to find an efficient primitive MRMM of order n over $\mathbb{F}_{2^m}$. In view of the proof of Theorem 2.4.11, we shall begin by first finding a primitive polynomial $f(X)$ of degree mn over $\mathbb{F}_2$ so as to obtain a primitive MRMM of order n over $\mathbb{F}_{2^m}$.

It may be remarked that for checking the primitivity of a polynomial of degree mn over $\mathbb{F}_q$, one needs to know the distinct prime factors of $q^{mn} - 1$ beforehand. The computational complexity of finding distinct prime factors of $q^{mn} - 1$ is very large. In fact, the factors of $q^{mn} - 1$ can not be computed in polynomial time in general. However, for smaller values of q (note that in most applications $q = 2$), many thanks to the Cunningham project [11, 52], the factorization of $2^{mn} - 1$ is known for reasonable values of mn that are needed in most of the practical applications. Our algorithm is based on the assumption that the distinct prime factors of $2^{mn} - 1$ are already known. The sequential steps of the algorithm [5] are described as follows.

In Step 2 of the algorithm, one may use Ben-Or's algorithm [2, 20] for the irreducibility test, which is quite efficient in practice. It is pointed out in [20] that by using fast multiplication [12], $O(m^2n^2 \log mn \log\log mn \log mn)$ is the worst case complexity of Ben-Or's algorithm. As noted in [21, Section 1], in polynomial basis representation of $\mathbb{F}_{2^{mn}}$ over $\mathbb{F}_2$, the exponentiation can be done with $O(m^2n^2 \log mn \log\log mn)$ operations in $\mathbb{F}_2$, with fast multiplication and repeated squaring. Thus the cost of Step 3 is $O(km^2n^2 \log mn \log\log mn)$. Let α denote the probability that a given random monic polynomial of degree mn is primitive. Since the number of primitive polynomials of degree mn over $\mathbb{F}_2$ is $\phi(2^{mn} - 1)/mn$, where ϕ is Euler's totient function. The value of α is given by $\phi(2^{mn}-1)/(mn2^{mn})$. It is clear that the expected number of times the Algorithm 2 is iterated to find a primitive MRMM is $1/\alpha$. So the expected number of times Step 2 to be executed is $1/\alpha$. It is well-known that

Algorithm 2 : Construction of an efficient primitive MRMM

Input: Positive integers m and n, the distinct prime factors $p_1, p_2, \cdots, p_k$ of $2^{mn}-1$.
Output: An efficient primitive MRMM(n, m).

1: Choose a random monic polynomial $f \in \mathbb{F}_2[X]$ of degree mn.
2: Verify if f is irreducible. If f is not irreducible then go to Step 1, otherwise go to Step 3.
3: Verify if f is primitive. If f is not primitive then go to Step 1, otherwise go to Step 4. The primitivity test is done as follows: Compute $h(X) = X^{(2^{mn}-1)/p_i} \bmod f(X)$ for each i. If $h(X) \neq 1$ for all k distinct prime factors p_i then f is primitive.
4: Express $f(X)$ in its n-Horner's form.
5: Construct n-Horner's matrix $H_m(n, f)$ of $f(X)$.
6: Express $H_m(n, f)$ in the form of matrix polynomial as described in Eq. (2.10), i.e.,

$$H_m(n, f) = I_m X^n + C_{n-1} X^{n-1} + \cdots + C_1 X + C_0,$$

where C_i denotes the $m \times m$ matrix whose entries are coefficients of X^i in the Horner's matrix $H_m(n, f)$.
7: Return $C_0, C_1, \cdots, C_{n-1}$.

the probability of a random monic polynomial of degree mn in $\mathbb{F}_2[X]$ being irreducible over $\mathbb{F}_2$ is close to $1/mn$. So the expected number of times Step 3 to be executed is $1/(mn\alpha)$. Thus $\alpha^{-1}O(m^2n^2 \log mn \log\log mn \log mn) + (mn\alpha)^{-1}O(km^2n^2 \log mn \log\log mn)$ is the expected run time of Algorithm 2, which after simplification, can be seen to be equal to $O(\alpha^{-1}mn \log mn \log\log mn(mn \log mn + k))$. Since the number of distinct prime factors of a given number N is at most $\log_2 N$, the value of k in our case is at most mn. As a consequence, the expected run time of Algorithm 2 is given by $O(\alpha^{-1}m^2n^2 \log mn \log\log mn \log mn)$.

Example 2.4.12. Let us consider the same polynomial $f(X)$ as given in Example 2.4.8. One can verify that $f(X)$ is a primitive over $\mathbb{F}_2$. Using Eq. (2.9), the 3-Horner's matrix of $f(X)$ is given by

$$H_4(3,f) = \begin{pmatrix} X^3 & 0 & 0 & (1+X^2) \\ 1 & X^3 & 0 & X^2 \\ 0 & 1 & X^3 & X \\ 0 & 0 & 1 & (X^3+X^2+X) \end{pmatrix}.$$

We can express $H_4(3,f)$ in the form of a following matrix polynomial

$$H_4(3,f) = I_3X^3 + C_2X^2 + C_1x + C_0,$$

where $C_0 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$, $C_1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$, and $C_2 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$. It is clear that $C_0 \in \mathrm{GL}_4(\mathbb{F}_2)$. If $\widetilde{T}$ denotes the block companion matrix corresponding to the matrix polynomial, that is, $\widetilde{T} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & C_0 \\ I_3 & \mathbf{0} & C_1 \\ \mathbf{0} & I_3 & C_2 \end{pmatrix}$, then $\widetilde{T} \in \mathrm{GL}_{12}(\mathbb{F}_2)$ and $o(\widetilde{T}) = 2^{12}-1$. Moreover, $\Psi_P(\widetilde{T}) = f(X)$. Now corresponding to these C_0, C_1, and C_2, we can associate a primitive MRMM of order 3 over $\mathbb{F}_{2^4}$.

2.4.3 Efficient Implementation

As pointed out in the previous section, the efficiency of MRMM constructed through Algorithm 2 is due to the special structure of the matrices $C_0, C_1, \ldots, C_{n-1}$. It was also noted

in Section 2.4.2 that all the columns of matrix C_j $(1 \leq j \leq n-1)$ are zero except the m^{th} column. Moreover, the matrix C_0 has a special structure. In fact, it is easy to see that $C_0 = R + \widehat{C_0}$, where R is right shift operator given by the matrix

$$R = \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}_{m \times m}, \tag{2.14}$$

and $\widehat{C_0}$ has all its columns zero except the m^{th} column, which is essentially the last column of C_0. The structure of $\widehat{C_0}$ is exactly same as C_j, $j \geq 1$.

Thanks to the following lemma that will make implementation of MRMM fast and efficient.

Lemma 2.4.13. For any matrix $A \in M_m(\mathbb{F}_2)$ having all the columns zero except the m^{th} column and for any vector $\mathbf{s} = [s_0, s_1, \ldots, s_{m-1}]^{\text{tr}} \in \mathbb{F}_2^m$, we have

$$A\mathbf{s} = s_{m-1}\mathbf{v}_m,$$

where $\mathbf{v}_m$ represents the m^{th} column of the matrix A.

Proof. Proof is obvious. □

By invoking Lemma 2.4.13, the recurrence relation Eq. (2.4) can be written as follows:

$$\begin{aligned} \mathbf{s}_{i+n} &= R\mathbf{s}_i + \widehat{C_0}\mathbf{s}_i + C_1\mathbf{s}_{i+1} + \cdots + C_{n-1}\mathbf{s}_{i+n-1} \\ &= R\mathbf{s}_i + s_0\mathbf{v}_m^0 + s_1\mathbf{v}_m^1 + \cdots + s_{n-1}\mathbf{v}_m^{n-1}, \end{aligned} \tag{2.15}$$

where s_i is the least significant bit (LSB) of $\mathbf{s}_i$, $\mathbf{v}_m^i$ is the m^{th} column of the matrix $C_i (0 \leq$

$i \leq n-1$).

It is clear that Eq. (2.15) can be computed by using only one right shift operation and at most n bitwise XOR operations instead of matrix multiplications and thus, provides an efficient software realization.

As the MRMMs constructed by the construction algorithm use only XOR and shift operations, they belong to the class of xorshift RNGs [33]. It is shown in [7] that the initial states of the xorshift generators produced by the construction algorithm are significant for the quality of pseudorandom vector generation. To avoid this weakness, it is suggested that the initial states of such xorshift RNGs need to be initialized with odd numbers.

Remark 2.4.14. The Algorithm 2 will produce $\frac{\phi(2^{mn}-1)}{mn}$ distinct efficient primitive MRMMs of order n over $\mathbb{F}_{2^m}$, where ϕ is Euler's totient function.

2.4.4 Comparative analysis of the construction algorithm with the search algorithm

Both the search algorithm and construction algorithm for generating efficient primitive MRMMs are randomized algorithms. In the case of the search algorithm [54], it first constructs a random MRMM and then checks the primitiveness whereas the construction algorithm [5] finds a primitive polynomial $f(x)$ over $\mathbb{F}_2$ and constructs a primitive MRMM from $f(x)$. It may be remarked that for checking the primitivity of a polynomial of degree mn over $\mathbb{F}_2$, one needs to know the distinct prime factors of $(2^{mn}-1)$ beforehand. Due to the Cunningham project [11], the factorization of $(2^{mn}-1)$ is known for reasonable values of mn which is sufficient for most of the practical applications. So, all primitive polynomials up to that reasonable value of mn could be generated offline. Then, using the construction algorithm it is possible to generate primitive MRMM in $O(1)$ computational complexity as all the matrix coefficients C_j of the MRMM can be constructed from the given primitive polynomial, just by rearranging the coefficients using Eq. (2.11) and Eq. (2.12).

In order to have a fair comparison, we have implemented both the randomized algorithms (i.e., construction and search algorithms) in Maple-16 to generate primitive MRMMs of

Order of the MRMM n	Avg. time in seconds to generate an MRMM by	
	Construction algorithm (100)	Search algorithm (100)
4	0.004	9.89
8	0.016	19.76
12	0.071	47.49
16	0.086	63.64
20	0.122	116.91

Table 2.2: Average time for $m = 8$

Order of the MRMM n	Avg. time in seconds to generate an MRMM by	
	Construction algorithm (100)	Search algorithm (100)
4	0.031	17.63
8	0.078	43.05
12	0.304	218.59
16	0.723	355.95
20	1.456	492.73

Table 2.3: Average time for $m = 16$

Order of the MRMM n	Avg. time in seconds to generate an MRMM by	
	Construction algorithm (100)	Search algorithm (50)
4	0.094	105.87
5	0.217	181.7
6	0.348	248.58
7	0.578	313.6
8	0.636	397.71
9	1.89	488.73
10	2.11	571.95

Table 2.4: Average time for $m = 32$

varied lengths over different extension fields. The used test machine is Intel Xeon(R) CPU E5645 @ 2.40GHz x 12 with 8 GB memory and a 64-bit Linux operating system. Since in most of the operating system, the word size is of the form 2^k, we have considered $m = 2^k$, for some positive integer k, and for our experiment, we have used $k = 3, 4$ and 5.

The Table 2.2 shows the comparison between the time taken by the construction algorithm and the search algorithm for different values of n with $m = 8$. The first column tells the MRMM's order and the second and third columns tell the average time taken to generate a single primitive MRMM. The number of iterations used to compute the average time is indicated in the column header. In this case, it is 100. Similarly, Table 2.3 and Table 2.4 contain the average time required with fixed values of m as 16 and 32, respectively. Note that in the third column of Table 2.4, the number of iterations used to compute average time is 50.

Based on our experimental results as shown in Tables 2.2, 2.3 and 2.4, it is observed that the construction algorithm is much faster compared to the search algorithm to produce a single primitive MRMM. The exact reason for this observation is yet to be investigated. However, It is shown in [5] that if α is the probability that a given random monic polynomial

of degree mn over $\mathbb{F}_2$ is primitive, then the expected run time complexity of Algorithm 2 is $O(\alpha^{-1}m^2n^2 \log mn \log\log mn \log mn)$, whereas the same for the search algorithm is not yet available in the literature.

2.4.5 Number of similar MRMMs

In this section, we are computing the total number of similar primitive MRMMs to the set of primitive MRMM generated by the construction algorithm [5]. It is noted that the Algorithm 2 maps every primitive polynomial $f(x)$ of degree mn over $\mathbb{F}_2$ into a unique primitive matrix polynomial $M(x)$ and then from $M(x)$, it constructs a unique MRMM of order n over $\mathbb{F}_{2^m}$. Using the similarity transformation, it is possible to construct several similar primitive MRMMs. Let $M(x)$ be the matrix polynomial generated by the construction algorithm and let $\tilde{M}(x)$ be a similar matrix polynomial of $M(x)$, i.e., $\tilde{M}(x) = PM(x)P^{-1}$, for a non-identity matrix $P \in GL_m(\mathbb{F}_2)$. Before showing that $\tilde{M}(x) \neq M(x)$, we first prove the following results.

Lemma 2.4.15. If $C \in M_m(\mathbb{F}_2)$ is a non-zero matrix having first $(m-1)$ columns contain only zeros and $P \in GL_m(\mathbb{F}_2)$ such that $PC = CP$, then m^{th} row of both P and I_m are same.

Proof. Let the matrices P and C be as follows

$$P = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1m} \\ p_{21} & p_{22} & \cdots & p_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m1} & p_{m2} & \cdots & p_{mm} \end{pmatrix}, \; C = \begin{pmatrix} 0 & 0 & \cdots & \alpha_1 \\ 0 & 0 & \cdots & \alpha_2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha_m \end{pmatrix}.$$

As $PC = CP$, we have the following equation

$$\begin{pmatrix} 0 & 0 & \cdots & 0 & \beta_1 \\ 0 & 0 & \cdots & 0 & \beta_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \beta_m \end{pmatrix} = \begin{pmatrix} \alpha_1 p_{m1} & \alpha_1 p_{m2} & \cdots & \alpha_1 p_{mm} \\ \alpha_2 p_{m1} & \alpha_2 p_{m2} & \cdots & \alpha_2 p_{mm} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_m p_{m1} & \alpha_m p_{m2} & \cdots & \alpha_m p_{mm} \end{pmatrix}_{m \times m} \tag{2.16}$$

where $\beta_i = \sum_{k=1}^{m} \alpha_k p_{ik}$. Since $C \neq 0$, at least one of $\alpha_i = 1$. Now comparing both sides of the i^{th} row of the above matrix equation Eq. (2.16), it gives $(0, 0, \cdots, \beta_i) = (p_{m1}, p_{m2}, \cdots, p_{mm})$. This implies $p_{m1} = p_{m2} = \cdots = p_{m(m-1)} = 0$. If $p_{mm} = 0$, then the m^{th} row of P is the zero vector. So P is not invertible, which is a contradiction. Thus, $p_{mm} = 1$ and this proves the lemma.

□

Lemma 2.4.16. Let P $\in GL_m(\mathbb{F}_2)$, where m^{th} row of both P and I_m are the same and R is the right shift matrix defined in Eq. (2.14). Then $PR = RP$ implies $P = I_m$.

Proof. As $PR = RP$, this implies

$$\begin{pmatrix} p_{12} & p_{13} & \cdots & p_{1m} & 0 \\ p_{22} & p_{23} & \cdots & p_{2m} & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ p_{11} & p_{12} & \cdots & p_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ p_{(m-1)1} & p_{(m-1)2} & \cdots & p_{(m-1)m} \end{pmatrix}.$$

Comparing 1^{st} row of both sides, implies $p_{12} = p_{13} = \cdots = p_{1m} = 0$. Again $p_{11} = 1$ as $P \in GL_m(\mathbb{F}_2)$. Then by comparing 2^{nd} and subsequent rows of both sides, it is easy to see that $p_{ij} = \delta_{i,j}$ for $i, j \in \{1, 2, \cdots, m\}$. So $P = I_m$. □

Theorem 2.4.17. *Suppose $M(x)$ be a primitive matrix polynomial generated by Algorithm 2, then for two distinct matrices $P,Q \in GL_m(\mathbb{F}_2)$, $PM(x)P^{-1} \neq QM(x)Q^{-1}$.*

Proof. First we will prove that for $P \in GL_m(\mathbb{F}_2)$, $M(x) = PM(x)P^{-1}$ if and only if $P = I_m$.

As $M(x)$ is generated by Algorithm 2, the matrix polynomial $M(x)$ will have the following form

$$\begin{aligned} M(x) &= Ix^n + C_{n-1}x^{n-1} + C_{n-2}x^{n-2} + \cdots + C_1x + C_0 \\ &= Ix^n + C_{n-1}x^{n-1} + C_{n-2}x^{n-2} + \cdots + C_1x + R + C_0'. \end{aligned}$$

Then if $M(x) = PM(x)P^{-1}$,

$$\begin{aligned} &\Rightarrow \quad PM(x) = M(x)P \\ &\Rightarrow \quad PR = RP \text{ and } PC_i = C_iP, \ \forall \ i = 0, 1, \cdots, n-1. \end{aligned}$$

Thus by Lemma 2.4.15 and Lemma 2.4.16, $P = I_m$. Next, we will see that for $P \neq Q$, $PM(x)P^{-1} \neq QM(x)Q^{-1}$. Define $R = Q^{-1}P$. So $R \in GL_m(\mathbb{F}_2)$ and $R \neq I_m$. So

$$\begin{aligned} & RM(x)R^{-1} \neq M(x) \\ \Rightarrow \quad & (Q^{-1}P)M(x)(P^{-1}Q) \neq M(x) \\ \Rightarrow \quad & PM(x)P^{-1} \neq QM(x)Q^{-1}. \end{aligned}$$

This completes the proof.

□

Using the above theorem, now it is possible to generate a distinct primitive matrix polynomial from a given primitive matrix polynomial. Again these distinct primitive matrix polynomials produce similar MRMMs. The Algorithm 3 produces such MRMMs. The steps of this algorithm are as follows.

Algorithm 3 : Construction of similar MRMM

Input: A primitive polynomial of degree mn.
Output: Returns a pair of similar MRMMs.
1: Using Algorithm 2, construct the matrix polynomial $M(x)$
2: Choose a random non-identity matrix $A \in GL_m(\mathbb{F}_2)$
3: Define $J(x) = A \cdot M(x) \cdot A^{-1}$
4: From the matrix polynomials $J(x)$ and $M(x)$ generate the pair of similar MRMMs.

Now, we are in a position to count the number of primitive similar MRMMs generated by the construction algorithm. The total number of primitive polynomials of degree mn over $\mathbb{F}_2$ is $\frac{\phi(2^{mn}-1)}{mn}$. Again, Algorithm 2 produces unique MRMM from a unique polynomial and so Algorithm 2 will generate $\frac{\phi(2^{mn}-1)}{mn}$ many distinct primitive MRMMs. Now using Theorem 2.4.17 and Algorithm 3, $|GL_m(\mathbb{F}_2)|$ many distinct primitive MRMMs of order n over $\mathbb{F}_{2^m}$ can be produced from a single primitive MRMM. Thus, the total number of distinct MRMMs of order n over $\mathbb{F}_{2^m}$ generated by Algorithm 2 is $\frac{\phi(2^{mn}-1)}{mn}|GL_m(\mathbb{F}_2)|$.

2.5 Word sequence in terms of concatenated bit sequences

We say a bit sequence $[w]$ is a concatenated sequence of two bit sequences $[u]$ and $[v]$ if $w_{2k} = u_k$ and $w_{2k+1} = v_k$, for $k \geq 0$.

Example 2.5.1. Consider the bit sequences $[u] = \{0, 1, 1, 0, 1, 1, \ldots\}$ and $[v] = \{0, 1, 0, 1, \ldots\}$. Then $[w] = \{0, 0, 1, 1, 1, 0, 0, 1, 1, 0, \ldots\}$ is the concatenated sequence. It is easy to check that $Per[w] = 6 = Per[u]Per[v]$.

Example 2.5.2. Consider the sequences $[u] = \{1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, \ldots\}$ and $[v] = \{0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, \ldots\}$ with common minimal polynomial $x^3 + x + 1$. Then the concatenated sequence $[w] = \{1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, \ldots\}$. It is easy to verify that $Per[w] = Per[u] = Per[v]$. In this case, there is no increase in the period of the concatenated sequence. The reason for this is given in Lemma 2.5.5.

Consider a nonzero word sequence $[\mathbf{s}]$ generated by a primitive MRMM(m, n). Then $[\mathbf{s}]$ can be expressed as the concatenated sequence of m bit sequences $[s^{(j)}]$ for $1 \leq j \leq m$. Here, $[s^{(j)}]$ is the j^{th} component bit sequence of $[\mathbf{s}]$. Also it is observed that $[s^{(j)}], 1 < j \leq m$ is a circular shift (left or right) of $[s^{(1)}]$.

Let r_j be the right circular shift constant for $[s^{(j)}]$ i.e., $s_k^{(j)} = (s_k^{(1)} >>> r_j)$ for $k \geq 0$. In Example 2.5.2, the value of r_i for $[v]$ with respect to $[u]$ is 4.

Lemma 2.5.3. All the circular shift constants of a primitive MRMM(m, n) are distinct.

Proof. Let the two right circular shift constants r_j and r_k be the same for a primitive MRMM(m, n). This implies $[s^{(j)}] = [s^{(k)}]$ for $1 \leq j \neq k \leq m$. Now consider the concatenated word sequence $[\tilde{\mathbf{s}}]$ generated from $[s^{(j)}]$ and $[s^{(k)}]$. Then each 2-bit word of $[\tilde{\mathbf{s}}]$ is either 0 or 3 and this leads to a contradiction. The same argument is valid for the left circular shift. □

Remark 2.5.4. For a given primitive MRMM(m, n), it is still unknown about the possible values of circular shift constants r_j for $1 \leq j \leq m$.

Lemma 2.5.5. Suppose $[u]$ and $[v]$ are two maximal nonzero bit sequences having common minimal polynomial of degree $n > 1$, then $Per[u]$ divides $Per[w]$. If $Per[w] = P$, then $L[w] = n$ otherwise $L[w] = 2n$.

Proof. Suppose $f(x)$ is the common minimal polynomial of $[u]$ and $[v]$. Define $[\tilde{u}]$ as $\tilde{u}_{2k} = u_k$ and $\tilde{u}_{2k+1} = 0$, for $k \geq 0$. Then the minimal polynomial for $[\tilde{u}]$ is $\tilde{f}_u(x) = f(x^2)$. Similarly, the minimal polynomial for $[\tilde{v}]$ is $\tilde{f}_v(x) = f(x^2)$ where $\tilde{v}_{2k} = 0$ and $\tilde{u}_{2k+1} = v_k$, for $k \geq 0$. It is obvious that the concatenated sequence $[w] = [\tilde{u}] \oplus [\tilde{v}]$. Thus $Per[w]$ must divide $2P$ and the desired result follows. □

In the following, we prove a result where the concatenated sequence achieves the maximal linear complexity.

Theorem 2.5.6. *Let $[w]$ be the concatenated sequence of $[u]$ and $[v]$. Suppose $Per[u] = P_1$ and $Per[v] = P_2$ and the minimal characteristic polynomials of $[u]$ and $[v]$ are f_u and f_v,*

respectively. If $gcd(f_u, f_v) = 1$, *then* $Per[w] = 2\ lcm(P_1, P_2)$ *and* $L[w] = 2\ (deg(f_u) + deg(f_v))$.

Proof. It is easy to check that $gcd(\tilde{f}_u, \tilde{f}_v) = 1$ as $gcd(f_u, f_v) = 1$. Therefore, by [23, Proposition 1], $Per[w] = lcm(Per[\tilde{u}], Per[\tilde{v}]) = lcm(2P_1, 2P_2) = 2\ lcm(P_1, P_2)$ and $L[w] = L[\tilde{u}] + L[\tilde{v}] = deg(\tilde{f}_u) + deg(\tilde{f}_v) = 2\ (deg(f_u) + deg(f_v))$. □

Theorem 2.5.7. *Let* $[a]$ *be a de Bruijn sequence of span* n_1 *and* $[b]$ *be a maximal sequence of order* n_2. *Let* $[w]$ *be the concatenated sequence of* $[a]$ *and* $[b]$, *then* $Per[w]$ *and* $L[w]$ *satisfy the following*

(i) $Per[w] = 2^{n_1+1}(2^{n_2} - 1)$.

(ii) $n_2 2^{n_1} < L[w] \leq n_2 2^{n_1+1}$.

Proof. As $[a]$ is a de Bruijn sequence and the order of $[b]$ is n_2, the period of $[a]$ and $[b]$ are 2^{n_1} and $2^{n_2}-1$, respectively. Again $gcd(2^{n_1}, (2^{n_2}-1)) = 1$, so by Theorem 2.5.6 and [13, Theorem 1], $Per[w] = 2\ lcm(2^{n_1}, 2^{n_2}-1) = 2^{n_1+1}(2^{n_2}-1)$ and $2n_2(2^{n_1-1}+1) \leq LC[w] \leq 2n_2(2^{n_1})$. □

Let N_0, N_1, respectively be the number of $0s$ and $1s$ in one period of the concatenated sequence $[w]$. Then, $N_0+N_1 = Per[w]$ with $N_0 = 2^{n_1-1}(2^{n_2+1}-1)$ and $N_1 = 2^{n_1-1}(2^{n_2+1}-3)$. This implies $N_1 - N_0 = 2^{n_1}$ which increases exponentially with n_1. Thus for a larger value of n_1, the frequency test [1] fails. Similarly, if $N_{00}, N_{01}, N_{10}, N_{11}$ denote the number of occurrences of $00, 01, 10, 11$, respectively, then it can be shown that $N_{00} = 2^{n_1}(2^{n_2-1} - 1)$ and $N_{11} = 2^{n_1}2^{n_2-1}$. Hence, the serial test also fails for a larger value of n_1. Therefore, the statistical properties are not promising in this type of concatenated sequence, even though it has a large period and linear complexity. Again, by using the Berlekamp-Massey algorithm, one can easily reproduce the initial states of the LFSR from any given $4n_2$ consecutive bits. Therefore, this concatenated bitstream should not be used directly as a keystream.

Chapter 3

Word-oriented shrinking generators

Like LFSRs, MRMMs are quite useful due to their simple structure with an exponential period, and most of the good statistical properties. However, they have also low linear complexity like LFSRs. The Corollary 2.3.4 implies that the linear complexity of a primitive MRMM(m, n) is $O(mn)$ compared to its period $O(2^{mn})$. Then the question arises, how to increase the LC, using primitive MRMMs? In the case of primitive MRMM of order n over $\mathbb{F}_{2^m}$, it produces m-bit output with $O(n)$ number of word operations. In [5], a method is proposed using the Langford arrangement to get a higher LC with complexity $O((mn)^2)$. Using nonlinear filtered and jump-controlled concepts in word-oriented linear feedback shift registers, a couple of techniques are provided in [28] to get exponential linear complexity. In this chapter, we expand the concept of bit-oriented shrinking and self-shrinking generators to their corresponding word-oriented counterparts. We subsequently examine the period, linear complexity, and randomness properties of these generators.

3.1 Shrinking MRMMs

The bit-oriented shrinking generator was introduced by Coppersmith *et al.* [15]. We have the following result due to [15, Theorem 1].

Theorem 3.1.1. *Let A and S form a shrinking generator Z. Denote by T_A, T_S, the periods*

of the A- and S- sequences, respectively. If

- *A and S are of maximal length (i.e. have primitive connections)*
- $gcd(T_A, T_S) = 1$

then the shrunken sequence Z has period $T_A \cdot 2^{|S|-1} = (2^{|A|} - 1) \cdot 2^{|S|-1}$.

Theorem 3.1.2. *If the condition* $gcd(T_A, T_S) \neq 1$ *in the above Theorem 3.1.1, then the shrunken sequence Z has period* $\dfrac{T_A}{gcd(T_A, T_S)} \cdot 2^{|S|-1}$.

Proof. Using the same technique as used in the proof of [15, Theorem 1], it is possible to prove the above result. □

In this section, we have extended the concept of a bit-oriented shrinking generator to that of a word-oriented shrinking generator and the above result helps to calculate its periodicity. Consider two word sequences $[\mathbf{a}] = \{\mathbf{a}_0, \mathbf{a}_1, \ldots\}$ and $[\mathbf{s}] = \{\mathbf{s}_0, \mathbf{s}_1, \ldots\}$ where each word of $[\mathbf{a}]$ is m_1 bit size and each word of $[\mathbf{s}]$ is m_2 bit size. Now, we define a new word sequence $[\mathbf{z}] = \{\mathbf{z}_0, \mathbf{z}_1, \ldots\}$ as shown in Fig. 3.1 using the sequences $[\mathbf{a}]$ and $[\mathbf{s}]$ as follows. For $i = 0, 1, \ldots$, if $\mathbf{s}_i$ is odd, then $\mathbf{a}_i$ is added into the sequence $[\mathbf{z}]$ otherwise discard $\mathbf{a}_i$. So, the resultant word sequence $[\mathbf{z}]$ is a shrunken version of the sequence $[\mathbf{a}]$ and $\mathbf{z}_i = \mathbf{a}_{i_k}$, where i_k is the position of the k^{th} odd number in the sequence $[\mathbf{s}]$. This concept can be implemented with any pair of word sequences. However, our analysis focuses on the scenario where both word sequences are generated by two MRMMs. Let $\text{MRMM}_1(m_1, n_1)$ and $\text{MRMM}_2(m_2, n_2)$ be those two primitive MRMMs. Suppose the word sequence $[\mathbf{a}]$ is generated by MRMM_1 and $[\mathbf{s}]$ is generated by MRMM_2, where both $[\mathbf{a}]$ and $[\mathbf{s}]$ are nonzero word sequences. Then, the period of $[\mathbf{a}]$ is $T_A = 2^{m_1 n_1} - 1$, whereas the period of $[\mathbf{s}]$ is $T_S = 2^{m_2 n_2} - 1$.

It is natural to ask what would be the periodicity and linear complexity of the new sequence $[\mathbf{z}]$. The subsequent theorems demonstrate that the period and linear complexity of the shrunken word sequence $[\mathbf{z}]$ exhibit exponential growth with respect to the order and word size of both MRMMs.

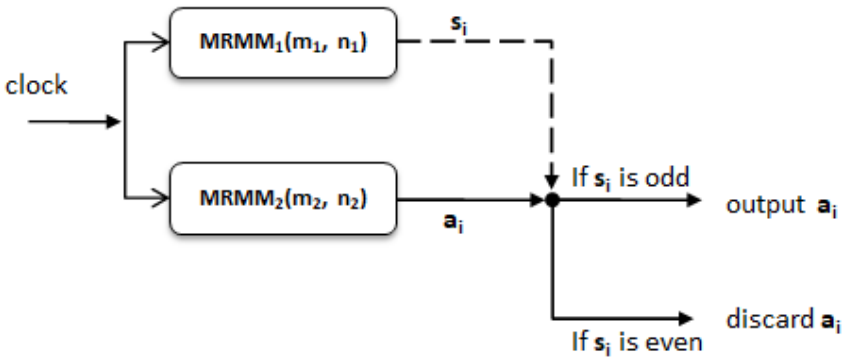

Figure 3.1: The shrinking MRMMs

Theorem 3.1.3. *Let* $[\boldsymbol{a}]$ *and* $[\boldsymbol{s}]$ *be two word sequences generated by two primitive MRMMs and form a shrinking generator* $[\boldsymbol{z}]$ *as defined above. If* T_A *and* T_S *be the period of* $[\boldsymbol{a}]$ *and* $[\boldsymbol{s}]$*, respectively, then the shrunken word sequence* $[\boldsymbol{z}]$ *has period* $\dfrac{T_A}{gcd(T_A, T_S)} \cdot 2^{m_2 n_2 - 1}$.

Proof. Let $[a^{(1)}]$ and $[s^{(1)}]$ be the bit sequences obtained from the word sequences $[\mathbf{a}]$ and $[\mathbf{s}]$, respectively, where $[a^{(1)}]$ contains 1^{st} least significant bit (LSB) of each word of $[\mathbf{a}]$ and similarly $[s^{(1)}]$ contains LSB of each word of $[\mathbf{s}]$. Then by Lemma 2.3.2, the bit sequence $[a^{(1)}]$ has period $T_A = 2^{m_1 n_1} - 1$ and the bit sequence $[s^{(1)}]$ has period $T_S = 2^{m_2 n_2} - 1$. If $[z^{(1)}]$ is the bit sequence generated by the bit-oriented shrinking generator, where $[s^{(1)}]$ is the control bit sequence, then by Theorem 3.1.2, the period of the shrunken sequence is $\dfrac{T_A}{gcd(T_A, T_S)} \cdot 2^{m_2 n_2 - 1}$.

Let $\mathbf{B}$ be a sequence constructed from the word sequences $[\mathbf{a}]$ and $[\mathbf{s}]$, which contains pairs of words i.e., $\mathbf{B} = \{(\mathbf{s}_0, \mathbf{a}_0), (\mathbf{s}_1, \mathbf{a}_1), \ldots\}$. Again, consider another sequence $\mathbf{C} = \{(s_0^{(1)}, \mathbf{a}_0), (s_1^{(1)}, \mathbf{a}_1), \ldots\}$ formed by the bit sequence $[s^{(1)}]$ and the word sequence $[\mathbf{a}]$. Then, if the shrinking generator concept is applied on both $\mathbf{B}$ and $\mathbf{C}$, they will produce the same word sequence $[\mathbf{z}]$. In particular, $[\mathbf{z}] = \{\mathbf{a}_{k_0}, \mathbf{a}_{k_1}, \ldots\}$, where k_i is the i^{th} odd number in the word sequence $[\mathbf{s}]$. Again, $[z^{(1)}]$ contains LSB of each word of $[\mathbf{z}]$ and $[z^{(1)}]$ has period $\dfrac{T_A}{gcd(T_A, T_S)} \cdot 2^{m_2 n_2 - 1}$. Therefore, the period of the word sequence $[\mathbf{z}]$ is $\dfrac{T_A}{gcd(T_A, T_S)} \cdot 2^{m_2 n_2 - 1}$. □

Remarks 3.1.4. If O_S and E_S are the number of odd numbers and even numbers in a full

period of the word sequence [**s**], respectively, then by the primitivity property of MRMM, $O_S = (E_S + 1)$ in a span of one period. This implies, $O_S = 2^{m_2 n_2 - 1} = \frac{T_S+1}{2}$. So the period of the shrunken word sequence [**z**] is $T_A O_S$.

Corollary 3.1.5. In the case of a primitive LFSR-based shrinking generator, both the word size m_1 and m_2 are equal to 1. If the periods of both LFSRs are relatively prime, then the period of the shrinking generator is $(2^{n_1} - 1)2^{n_2-1}$.

Since in most computer systems, the word size is either 64, 32, or 16. We may assume that in general, it is of the form 2^k for some positive integer k. So, we may let $m_1 = 2^{k_1}$ and $m_2 = 2^{k_2}$ for some positive integers k_1 and k_2. The following theorem gives both the lower and upper bound of linear complexity of the bit sequence generated by the shrunken word-based generator.

Theorem 3.1.6. *Under the conditions of Theorem 3.1.3 and if* $gcd(T_A, T_S) = 1$*, then the linear complexity LC of the shrunken sequence* [$\boldsymbol{z}$] *satisfies the following inequality,*

$$m_1{}^2 n_1 \tfrac{(T_S+1)}{4} < LC \leq m_1{}^2 n_1 \tfrac{(T_S+1)}{2}$$

Proof. Suppose the MRMM used in [**a**] has word size $m_1 = 2^{k_1}$, thus the bit period of the shrunken sequence is $m_1(2^{m_1 n_1} - 1)2^{m_2 n_2 - 1}$. Therefore, by [15, Theorem 2], LC satisfies the following inequality.

$$m_1{}^2 n_1 2^{m_2 n_2 - 2} < LC \leq m_1{}^2 n_1 2^{m_2 n_2 - 1}$$

This completes the proof. □

3.2 Self-shrinking MRMMs

In this section, we introduce word-oriented self-shrinking generator similar to bit-oriented self-shrinking generator (SSG) [36]. Here, we would like to show one new result on bit-oriented SSG. In [36], it is proven that if the period of an SSG of order N is P, then

$P \geq 2^{\lfloor \frac{N}{2} \rfloor}$ and P divides 2^{N-1}. In the following lemma, we show that the period is exactly equal to 2^{N-1}.

Theorem 3.2.1. *The period of the bit sequence produced by a self-shrinking primitive LFSR of order N over $\mathbb{F}_2$ is equal to 2^{N-1}.*

Proof. First, we will show that the SSG can be implemented using a shrinking generator with two LFSRs. Consider $[s] = \{s_0, s_1, \ldots\}$ be a bit sequence produced by a primitive LFSR of order N. Then by definition of the SSG, for $k = 0, 1, \ldots$, the bit s_{2k+1} is added to the bitstream of the shrunken sequence if $s_{2k} = 1$. This means the sequence $\{s_0, s_2, \ldots\}$ acts as control bit, and the sequence $\{s_1, s_3, \ldots\}$ defines the sequence being controlled. Now, consider a shrinking generator where both the LFSRs are the same as the LFSR of the SSG and these two LFSRs are initialized with the states $\{s_0, s_2, \ldots, s_{2N-2}\}$ and $\{s_1, s_3, \ldots, s_{2N-1}\}$, respectively. Then, this shrinking generator will produce the same bit sequence generated by the SSG.

Now, we are in a position to calculate the period of the SSG using the result of the shrinking generator. The bit sequence of the SSG is reproduced by a shrinking generator with two LFSRs having the same original feedback function. So, by Theorem 3.1.2, the period of the shrinking generator is $\frac{T_S}{gcd(T_S, T_S)} \cdot 2^{|N|-1}$, where T_S is the period of the sequence S. Therefore, the period of the SSG is $2^{|N|-1}$. □

Now, we introduce word-oriented SSG. For this, let $[\mathbf{s}] = \{\mathbf{s}_0, \mathbf{s}_1, \ldots\}$ be a sequence of words with period P. Now arrange the words of $[\mathbf{s}]$ as pairs of words i.e., $\{(\mathbf{s}_0, \mathbf{s}_1), (\mathbf{s}_2, \mathbf{s}_3), \ldots\}$. If the period P is even, then there will be $\frac{P}{2}$ pairs, otherwise P pairs will be generated before the repetition of pairs occurs. Now, we define a new word sequence $[\mathbf{z}] = \{\mathbf{z}_0, \mathbf{z}_1, \ldots\}$ as follows. For $i = 0, 1, \ldots$, if $\mathbf{s}_{2i}$ is odd, then $\mathbf{s}_{2i+1}$ is added into the sequence $[\mathbf{z}]$ otherwise discard both $\mathbf{s}_{2i}$ and $\mathbf{s}_{2i+1}$. The self-shrinking MRMM concept is shown in Fig. 3.2. Since we are interested in MRMM with a large period, only primitive MRMM(m, n) is considered. In the following theorem, a result pertaining to the period of the self-shrinking generator is provided.

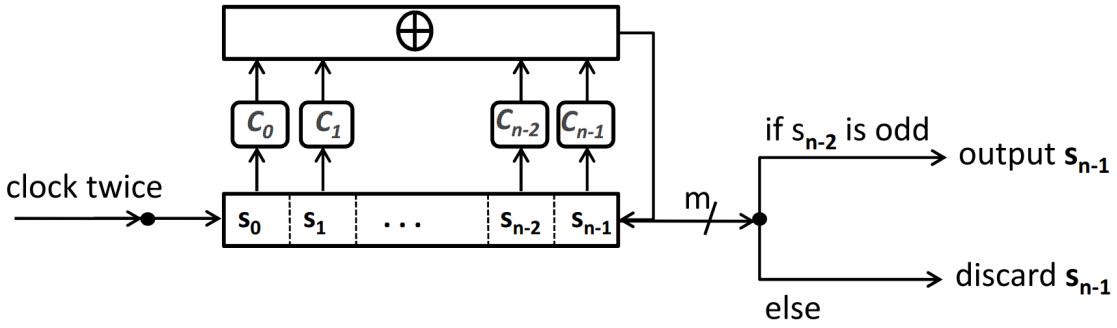

Figure 3.2: The self shrinking MRMM of order n over $\mathbb{F}_{2^m}$

Theorem 3.2.2. *The word sequence produced by a self-shrinking primitive MRMM of order n over $\mathbb{F}_{2^m}$ is periodic and its period divides 2^{mn-1}. Moreover, the generated shrunken sequence is balance.*

Proof. Suppose, $[\mathbf{s}] = \{\mathbf{s}_0, \mathbf{s}_1, \ldots\}$ be the word sequence generated by a primitive MRMM(m, n) and $[\mathbf{z}]$ is the self-shrinking sequence of $[\mathbf{s}]$. So the period of $[\mathbf{s}]$ is $P = (2^{mn} - 1)$ which is odd. This implies, the initial pair i.e., $(\mathbf{s}_0, \mathbf{s}_1)$ will be repeated after P pairs of words. So the word sequence generated by the self-shrinking MRMM is periodic. Note that each word $\mathbf{s}_i$ for $i = 0, 1, \ldots, (P-1)$, occurs once as the first component of a unique pair in the first P pairs. In particular, if i is even then $\mathbf{s}_i$ occurs in $(\frac{i+2}{2})^{th}$ pair, otherwise it occurs in $(\frac{P+i+2}{2})^{th}$ pair. Again, by the properties of primitive MRMM, the word sequence $[\mathbf{s}]$ will have 2^{mn-1} odd and $(2^{mn-1} - 1)$ even m-bit words in its period. This implies the period of $[\mathbf{z}]$ divides 2^{mn-1}.

Again, due to the properties of primitive MRMM, for fixed m-bit odd number $\mathbf{a}$ and for every m-bit $\mathbf{b}$, the pair $(\mathbf{a}, \mathbf{b})$ occurs equal number of times in $[\mathbf{s}]$ in a span of two period. Therefore, the sequence $[\mathbf{z}]$ is balance. □

Corollary 3.2.3. In the case of primitive LFSR of length n, $m = 1$ therefore, the period of self-shrinking sequence divides 2^{n-1}.

Let the m-sequence $[\mathbf{s}] = \{\mathbf{s}_0, \mathbf{s}_1, \ldots\}$ be generated by a primitive MRMM(m, n). Suppose the sequence $[\mathbf{z}] = \{\mathbf{z}_0, \mathbf{z}_1, \ldots\}$ be generated from $[\mathbf{s}]$ using self-shrinking concept. Let $[z^{(j)}] =$

$\{z_0^{(j)}, z_1^{(j)}, \ldots, \}$ be the j^{th} bit sequence, where $z_k^{(j)}$ is the j^{th} bit of m-bit word $\mathbf{z}_k$. So, $\mathbf{z}_k = (z_k^{(m)}, \ldots, z_k^{(2)}, z_k^{(1)}) \in \mathbb{F}_2^m$.

The following theorem gives a lower bound on the period of a shrunken sequence generated by a primitive self-shrinking MRMM. An almost similar technique is used to derive the bound as given in [36, Theorem 2].

Theorem 3.2.4. *In case of a primitive MRMM(m, n), the period P of $[z^{(j)}]$ satisfies*

$$P \geq 2^{m\lfloor \frac{n}{2} \rfloor}. \tag{3.1}$$

Proof. First consider n is even i.e., $n = 2k$ for some $k > 0$. Since the generator MRMM is primitive, by Lemma 2.3.2 the first LSB sequence $[z^{(1)}]$ produces maximum length sequence with period $(2^{mn} - 1)$, where every nonzero mn-bit word appears exactly once when scanning the sequence with a window of length $mn = 2mk$ over the full period. Again, due to the maximum length property, the mn-bit pattern $(1, x_1, 1, x_2, \ldots, 1, x_{mk})$ appears in the original sequence for every choice of $(x_1, x_2, \ldots, x_{mk})$. This implies that every mk-bit pattern appears in the bit sequence $[z^{(1)}]$ when scanning it with window size mk therefore, the period of $[z^{(1)}]$ is greater or equal to 2^{mk}. Hence, $P \geq 2^{mk}$.

Now for odd n, let $k = (n-1)/2$ therefore, $m(n-1) = 2mk$. Then the $(n-1)m$-bit pattern $(1, x_1, 1, x_2, \ldots, 1, x_{mk})$ appears (twice) when scanning the bit sequence $[z^{(1)}]$. Therefore, the period of $[z^{(1)}]$ is greater than 2^{mk} and hence, $P > 2^{mk}$. This completes the proof.

□

Applying the technique utilized in [36] to compute the lower bound of Linear Complexity (LC) for a self-shrunken maximum length LFSR, we can establish the lower bound on LC for the bit sequence generated by a self-shrunken primitive MRMM, as demonstrated in the following theorem.

Theorem 3.2.5. *The linear complexity LC of the bit sequence produced by a self-shrunken*

maximum length MRMM(m, n) with $m = 2^k$ satisfies

$$LC > 2^{m\lfloor \frac{n}{2} \rfloor + k - 1}. \tag{3.2}$$

Proof. Let P_{bit} be the period of the bit sequence produced by a self-shrunken maximum length MRMM(m, n). As the size of each word is m-bit and period of word sequence is P, so by Theorem 3.2.4 we have $P_{bit} = mP \geq m2^{m\lfloor \frac{n}{2} \rfloor} = 2^{k+m\lfloor \frac{n}{2} \rfloor}$. By Theorem 3.2.2, it is clear that the period P_{bit} of the self-shrunken bit sequence of the given primitive MRMM is of the form 2^a with $a \geq (k + m\lfloor \frac{n}{2} \rfloor)$. Again in case of binary field $\mathbb{F}_2$, it is possible to write $x^{2^a} - 1 = (x - 1)^{2^a}$. As P_{bit} is the period of the self-shrunken bit sequence, the minimal polynomial $f(x)$ of the bit sequence must divide $(x^{P_{bit}} - 1) = (x - 1)^{2^a}$. This implies $f(x)$ is of the form $f(x) = (x - 1)^L$, where L is the linear complexity of the self-shrunken sequence. Using the same logic used in [36, Theorem 3], it can be shown that $L > 2^{a-1}$. This completes the proof. □

Corollary 3.2.6. In the case of primitive LFSR of length n, the word size $m = 1$ therefore, $k = 0$. So the linear complexity of a self-shrunken maximum length LFSR sequence produced by an LFSR of length n is greater than $2^{\lfloor \frac{n}{2} \rfloor - 1}$.

3.3 Experiment and results on self shrinking MRMM

The Theorem 3.2.2 proves that the period P is a factor of 2^{mn-1}, whereas the Theorem 3.2.4 gives the lower bound on P. But, there is no knowledge about the exact period of a primitive self-shrinking MRMM. In the following section, we investigate more on the period of self-shrinking generators.

3.3.1 Experiment on Period

In order to study the exact period, we have calculated the period of several self-shrinking MRMMs experimentally. Interestingly, in our experiment, it is found that every time, the

period P is 2^{mn-1} for a primitive MRMM(m, n). This motivates us to compute the period for primitive MRMMs of lower value of m and n exhaustively. By Remarks 2.3.3, it is easy to realize that for a given primitive self-shrinking MRMM, two word sequences generated from two different nonzero initial states are the same with respect to circular shifting. So we have taken the initial states for the primitive MRMM as the impulse $\{0, 0, \ldots, 1\}$ i.e., $\mathbf{s}_{n-1} = 1$ and the remaining other states are 0. The layout of our experiment is as follows.

- For each primitive polynomial of degree mn, we generate corresponding primitive MRMM of order n with word size m using the construction algorithm for primitive MRMM [5]. Note that for a given word size m and order n, the total number of primitive polynomials of degree mn over $\mathbb{F}_2$ is $N_0 = \frac{\phi(2^{mn}-1)}{mn}$. So by the construction algorithm, we have N_0 primitive MRMMs.

- Then using these N_0 primitive MRMMs, N_0 bitstream files are created with word sequence of length 2^{mn+1} using self shrinking generator concept as explained in Section 3.2. In our experiment, $\mathbf{s}_{n-1}$ is taken as the output word of MRMM in each iteration.

- Computed the period of each word sequence.

In this experiment, as the value of m and n increases, the data size increases exponentially. Hence, we have considered only small values of m and n. Table 3.1 shows the observed period of word sequence generated by the primitive self-shrinking MRMMs for different values of m and n. The first column shows the word size m and the second column is the order of the MRMM n, whereas the third column tells the observed period of the generated word sequence. Interestingly, it is found that all generated word sequences have the same period for a fixed value of m and n. In fact, the period P is exactly equal to 2^{mn-1}. This observation is proved in the following theorem.

Theorem 3.3.1. *The word sequence produced by a self-shrinking primitive MRMM of order n over $\mathbb{F}_{2^m}$ is periodic and its period is 2^{mn-1}.*

Proof. The proof of this result is similar to the proof of Theorem 3.2.1. The word sequence produced by a self-shrinking primitive MRMM of order n over $\mathbb{F}_{2^m}$ can be implemented using

Word size m	Order n	Observed period P
3	3	$256 = 2^8$
3	4	$2048 = 2^{11}$
3	5	$16384 = 2^{14}$
4	3	$2048 = 2^{11}$
4	4	$32768 = 2^{15}$
5	3	$16384 = 2^{14}$

Table 3.1: Period in self-shrinking MRMMs

word based shrinking generator with two equal MRMMs having the same feedback function of the original MRMM of the SSG. Then by Theorem 3.1.3, the period of the self-shrinking primitive MRMM is 2^{mn-1}. □

3.3.2 Results of NIST statistical test suite

In order to study the statistical properties of a bit sequence, randomness tests must be applied to it. Many statistical test suites have been developed to analyze the randomness properties of a bitstream. In our experiment, we have used NIST statistical test suite SP 800-22 Rev.1a [1]. To see the randomness properties of the word sequences generated by primitive self-shrinking MRMMs, we first convert word sequences to bit sequences. In the previous section, it is observed that 2^{mn-1} is the period of the word sequence, so the period of the corresponding bit sequence will be $m2^{mn-1}$ as each word is m-bit wide. Therefore, we have taken the bitstream length of each file as $m2^{mn-1}$. We have investigated the randomness properties as follows:

- The same procedure is used to generate word sequence of length 2^{mn-1} as described earlier in Section 3.3.1. Then each word is converted to m bits so that each file contains $m2^{mn-1}$ bits.

- Then, NIST statistical test suite is performed on all bitstream files. For this experiment, the significant value α is taken as 0.05.

We carry out this experiment for smaller values of (m, n) pair because of memory and time constraints. The list of (m, n) pairs for which NIST statistical test suite are carried out is $\{(3,3), (3,4), (3,5), (4,3), (4,4), (5,3)\}$. The results on each pair of (m, n) are almost similar. So the results of one pair of (m, n) for $m = 5$ and $n = 3$ are provided in Table 3.2. This shows the statistics of all 15 statistical tests of NIST test suite. The first column tells the name of the statistical test of NIST test suite, whereas the second column shows the number of files that pass the corresponding test. Column three tells the percentage of success. In Table 3.2, the value of $m = 5$ and $n = 3$, so the total number of files is 1800 as the total number of primitive polynomials of degree $mn = 15$ over $\mathbb{F}_2$ is 1800.

3.3.3 Observations

From Table 3.2, It is clear that four tests (i.e., Frequency, Linear Complexity, Non-Overlapping Template Matchings, and Cumulative Sums tests) have a passing percentage of 100 i.e., these tests pass always. The reason behind the Frequency test and Linear Complexity test are already proved in Theorem 3.2.2 and Theorem 3.2.5, respectively. However, for the other two tests i.e., Non-Overlapping Template Matchings and Cumulative Sums tests; the reason needs to be investigated.

The second observation from Table 3.2 is that the passing percentage for the Universal test and Approximate Entropy test is 0 and it is true for other above-mentioned (m, n) pairs. The third observation is that in the case of the Runs test and Serial test passing percentage is close to around 50 i.e., the probability of passing these two tests in the case of self-shrinking MRMM is $\frac{1}{2}$.

Name of the test	No. of passed files	% of passed test
Frequency	1800	100.00
BlockFrequency	1219	67.72
Runs	899	49.94
LongestRunOfOnes	191	10.61
Rank	1716	95.33
DiscreteFourierTransform	1018	56.56
NonOverlappingTemplateMatchings	1800	100.00
OverlappingTemplateMatchings	648	36.00
Universal	0	0.00
LinearComplexity	1800	100.00
Serial	899	49.94
ApproximateEntropy	0	0.00
CumulativeSums	1800	100.00
RandomExcursions	337	18.72
RandomExcursionsVariant	371	20.61

Table 3.2: Statistics of self shrinking MRMMs for $m = 5$ and $n = 3$

3.4 Conclusion

With the aim of increasing linear complexity using MRMMs, we extend the idea of bit-oriented shrinking generators and self-shrinking generators to MRMMs. Then, present some results pertaining to their periodicity and statistical properties. The results pertaining to word-oriented shrinking and self-shrinking generators are published as research articles[1].

In the next chapter, we discuss another word-oriented generator and its variant, based on cascade connection.

[1]S. K. Bishoi, K. Senapati and B. R. Shankar, *Shrinking generators based on σ-LFSRs*, Discrete Applied Mathematics, vol. 285, pp. 493-500, 2020.

Chapter 4

Cascade connection of WFSRs

The concept of cascade connection of two FSRs was first introduced in [25]. Let FSR_1 and FSR_2 denote two FSRs with characteristic functions f_1 and f_2, respectively. Consider the cascade connection of the FSR_1 into the FSR_2 is abbreviated as $\text{FSR}(f_1; f_2)$. Then it was shown in [25] that the $\text{FSR}(f_1; f_2)$ produced the same family of sequences as the FSR with characteristic function $f_1 * f_2$. Here the operator $*$ is defined as follows

$$f_1 * f_2(z_0, z_1, \ldots, z_{n+m}) = f_1(f_2(z_0, \ldots, z_m), \ldots, f_2(z_n, \ldots, z_{n+m})), \tag{4.1}$$

for two Boolean functions $f_1(x_0, x_1, \ldots, x_n)$ and $f_2(y_0, y_1, \ldots, y_m)$. Note that $f_1 * f_2$ is a Boolean function of $(n+m+1)$-variables. If both the FSRs are linear, then $f_1 * f_2 = f_2 * f_1$. In $\text{FSR}(f_1; f_2)$, FSR_1 runs in free running mode i.e., only the values of the states of FSR_1 are involved in the feedback value calculation of FSR_1, whereas FSR_2 runs in scrambler mode i.e., the feedback value calculation of FSR_2 uses the values of the states of FSR_2 along with some external values. One example of a cascade connection of FSRs is Grain-128 [16], where FSR_1 is an LFSR and FSR_2 is an NFSR. Hu *et al.* [29] have proved that the period of FSR_1 divides the period of $\text{FSR}(f_1; f_2)$.

Another variant of cascade connection of FSRs is used in TRIVIUM [16] and ACORN [53] ciphers, where all the FSRs run in scrambler mode. In TRIVIUM, three NFSRs are used, whereas six LFSRs are used in ACORN. This section extends the concept of cascade

connection to word-based FSRs where the first FSR runs in free-running mode and the remaining run in scrambler mode. Then, we study their period and different randomness properties.

First, we focus on the cascade system (CS) comprises of two WFSRs only. Suppose WFSR_1 and WFSR_2 are those two WFSRs where WFSR_1 is cascaded into WFSR_2 as depicted in Fig. 4.1.

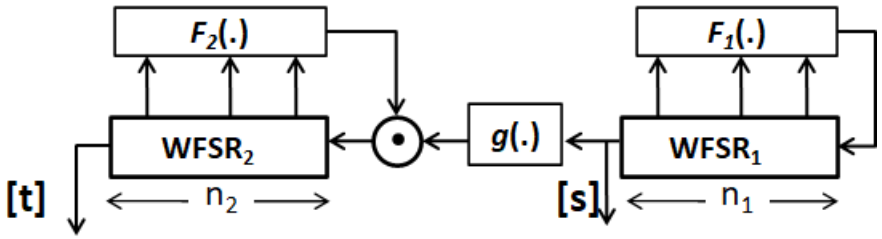

Figure 4.1: The cascade connection of WFSR_1 into WFSR_2.

Suppose the orders of WFSR_1 and WFSR_2 are n_1 and n_2, respectively. Let $\mathbf{S}_0 = (\mathbf{s}_0, \mathbf{s}_1, \ldots, \mathbf{s}_{n_1-1})$ and $\mathbf{T}_0 = (\mathbf{t}_0, \mathbf{t}_1, \ldots, \mathbf{t}_{n_2-1})$ be the initial states of WFSR_1 and WFSR_2, respectively. Suppose $F_1(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n_1-1})$ is the feedback function for WFSR_1, mapping from $\mathbb{F}_{2^m}^{n_1}$ to $\mathbb{F}_{2^m}$. Similarly, let $F_2 : \mathbb{F}_{2^m}^{n_2} \to \mathbb{F}_{2^m}$ be the feedback function for WFSR_2. Let $g : \mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$ be a bijective function. Consider the word sequence $[\mathbf{s}]$ which is generated by WFSR_1 in free running mode and $[\mathbf{t}]$ is generated by the cascade connection of WFSR_1 into WFSR_2. Thus, WFSR_2 runs in scrambler mode and the feedback value of WFSR_2 is calculated as follows

$$\mathbf{t}_{i+n_2} = g(\mathbf{s}_i) \odot F_2(\mathbf{t}_i, \mathbf{t}_{i+1}, \ldots, \mathbf{t}_{i+n_2-1}), \ \textit{for } i \geq 0, \tag{4.2}$$

where the operation $\odot$ can be either $\oplus$ or modular addition. Let P_s and P_t be the periods of the sequences $[\mathbf{s}]$ and $[\mathbf{t}]$, respectively. Then we have the following relation between P_s and P_t.

Theorem 4.0.1. *If WFSR_1 is periodic and WFSR_2 is nonsingular, then $[\boldsymbol{t}]$ is periodic and P_s divides P_t.*

Proof. It is obvious that the sequence $[\mathbf{t}]$ will be *ultimately* periodic and let Q_t be the preperiod of $[\mathbf{t}]$. Then, $\mathbf{t}_i = \mathbf{t}_{i+P_t}$ for any $i \geq Q_t$ therefore, $\mathbf{t}_{i+n_2} = \mathbf{t}_{i+n_2+P_t}$. So by Eq. (4.2), $g(\mathbf{s}_i) + F_2(\mathbf{t}_i, \ldots, \mathbf{t}_{i+n_2-1}) = g(\mathbf{s}_{i+P_t}) + F_2(\mathbf{t}_{i+P_t}, \ldots, \mathbf{t}_{i+P_t+n_2-1})$. But P_t is the period of $[\mathbf{t}]$ and thus, $g(\mathbf{s}_i) = g(\mathbf{s}_{i+P_t})$ for all $i \geq Q_t$. As g is bijective, $\mathbf{s}_i = \mathbf{s}_{i+P_t}$. This implies that $\mathbf{s}_i = \mathbf{s}_{i+P_t}$ for any $i \geq 0$ as $[\mathbf{s}]$ is periodic. But, the period of $[\mathbf{s}]$ is P_s and hence P_s divides P_t.

Suppose, $[\mathbf{t}]$ is not periodic i.e., $Q_t > 0$. Then, $\mathbf{t}_{Q_t+n_2-1} = \mathbf{t}_{Q_t+n_2-1+P_t}$ as P_t is the period of $[\mathbf{t}]$ and $Q_t + n_2 - 1 \geq Q_t$. This implies, $g(\mathbf{s}_{Q_t-1}) \odot F_2(\mathbf{t}_{Q_t-1}, \ldots, \mathbf{t}_{Q_t+n_2-2}) = g(\mathbf{s}_{Q_t-1+P_t}) \odot F_2(\mathbf{t}_{Q_t-1+P_t}, \ldots, \mathbf{t}_{Q_t+n_2-2+P_t})$. Since P_s divides P_t, it implies that $\mathbf{s}_{Q_t-1} = \mathbf{s}_{Q_t-1+P_t}$ therefore, $g(\mathbf{s}_{Q_t-1}) = g(\mathbf{s}_{Q_t-1+P_t})$. Thus, $F_2(\mathbf{t}_{Q_t-1}, \ldots, \mathbf{t}_{Q_t+n_2-2}) = F_2(\mathbf{t}_{Q_t-1+P_t}, \ldots, \mathbf{t}_{Q_t+n_2-2+P_t})$. Again, FSR_2 is nonsingular and by Theorem 2.2.2, F_2 can be expressed as $F_2(\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}) = f_{20}(\mathbf{x}_0) \odot f_{21}(\mathbf{x}_1, \ldots, \mathbf{x}_{n-1})$ with f_{20} is one-one. This results $f_{20}(\mathbf{t}_{Q_t-1}) = f_{20}(\mathbf{t}_{Q_t-1+P_t})$ and $\mathbf{t}_{Q_t-1} = \mathbf{t}_{Q_t-1+P_t}$. Thus, $\mathbf{t}_i = \mathbf{t}_{i+P_t}$ for any $i \geq Q_t - 1$. This is a contradiction to the fact that Q_t is the preperiod of $[\mathbf{t}]$. Hence $Q_t = 0$ and this completes the proof.

□

With $\odot$ as $\oplus$ and the function $g(.)$ as the identity function, the above result was proved in [9, Theorem 9]. In the following section, we study the statistical properties of the bitstream generated by different CSs using different operations of $\odot$ and later analyze the avalanche effect on states of CSs.

4.1 Cascade connection of two MRMMs

Consider two MRMMs, namely $\mathrm{MRMM}_1(m_1, n_1)$ and $\mathrm{MRMM}_2(m_2, n_2)$. Let $M_1(x) = x^{n_1} - C_{n_1-1}x^{n_1-1} - \ldots - C_1x - C_0$ and $M_2(x) = x^{n_2} - D_{n_2-1}x^{n_2-1} - \ldots - D_1x - D_0$ be the matrix polynomial of MRMM_1 and MRMM_2, respectively in $\mathbb{F}_2[x]$. Here C_i and D_j are $m_1 \times m_1$ and $m_2 \times m_2$ matrices, respectively for $0 \leq i \leq n_1 - 1$ and $0 \leq j \leq n_2 - 1$. If T_1 and T_2 are

companion matrices of MRMM_1 and MRMM_2, respectively, then

$$(T_1)_{m_1n_1\times m_1n_1} = \begin{pmatrix} \mathbf{0} & I & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & I & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & I \\ C_0 & C_1 & C_2 & \cdots & C_{n_1-1} \end{pmatrix}_{n_1\times n_1}, \tag{4.3}$$

and

$$(T_2)_{m_2n_2\times m_2n_2} = \begin{pmatrix} \mathbf{0} & I & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & I & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & I \\ D_0 & D_1 & D_2 & \cdots & D_{n_2-1} \end{pmatrix}_{n_2\times n_2}. \tag{4.4}$$

Here $\mathbf{0}$ and I are zero and Identity matrices of dimension $m_1 \times m_1$ in T_1 and, $m_2 \times m_2$ in T_2. Define an $(m_1n_1 + m_2n_2) \times (m_1n_1 + m_2n_2)$ matrix $\widetilde{T}$ as follows

$$\widetilde{T} = \begin{pmatrix} T_1 & 0 \\ B & T_2 \end{pmatrix}, \tag{4.5}$$

where the matrix B is given as

$$(B)_{m_2n_2\times m_1n_1} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \widetilde{I} & \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix}_{n_2\times n_1} .$$

Here the order of all the matrix entries of B is $m_2 \times m_1$. $\widetilde{I}$ is the matrix having all the entries 0 except $min(m_1, m_2)$ entries as 1s along the diagonal. For example, with $m_1 = 2$ and $m_2 = 3$, $\widetilde{I} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}$ and with $m_1 = 3$ and $m_2 = 2$, $\widetilde{I} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$.

It can be checked that $\widetilde{T}$ is the matrix representation of the cascaded system of two MRMMs. Now, for $j \geq 1$, $\sum_{k=0}^{j-1} T_2^k B T_1^{j-k-1}$ is well defined and it can be verified that

$$\widetilde{T}^j = \begin{pmatrix} {T_1}^j & \mathbf{0} \\ \sum_{k=0}^{j-1} T_2^k B T_1^{j-k-1} & {T_2}^j \end{pmatrix}. \tag{4.6}$$

From Eq. (4.5), it is clear that the characteristic polynomial of the matrix $\widetilde{T}$ is the product of the characteristic polynomial of the matrices T_1 and T_2. For any $m_2n_2 \times m_1n_1$ order matrix $B^{'}$, if we define

$$\widetilde{T}^{'} = \begin{pmatrix} T_2 & 0 \\ B^{'} & T_1 \end{pmatrix},$$

then the characteristic polynomials of the matrices $\widetilde{T}$ and $\widetilde{T}^{'}$ are same. Therefore, the cascade connection of MRMM_1 into MRMM_2 produces the same sequences as that of MRMM_2 into

MRMM_1 with respect to circular shifting. This fact is explained in [25] for LFSR. Before finding the period of the matrix $\widetilde{T}$, we prove the following lemma.

Lemma 4.1.1. If m and n are two positive integers with $gcd(m, n) = 1$, then the set $\{kn \mod m\}_{k=0}^{m-1} = \{0, 1, \ldots, m-1\}$.

Proof. Suppose $k_1 n = k_2 n \mod m$ for $0 \leq k_1 \neq k_2 \leq m-1$. This implies $(k_1 - k_2)n = 0 \mod m$ and so $k_1 = k_2$ as $gcd(m, n) = 1$. This is a contradiction. Thus, the elements of the set $\{kn \mod m\}_{k=0}^{m-1}$ are distinct and the desired result follows. □

Theorem 4.1.2. *If the period of $MRMM_1$ and $MRMM_2$ are P_1 and P_2, respectively with $gcd(P_1, P_2) = 1$, then the period of $\widetilde{T}$ is $P_1 P_2$.*

Proof. Let $j = kP_1P_2 + r$ be the period of $\widetilde{T}$, where $0 < r < P_1P_2$ is an integer. Then at least one of the P_1 and P_2 does not divide r and thus, either $T_1^r \neq I$ or $T_2^r \neq I$. But $\widetilde{T}^j = I$, implies $\begin{pmatrix} T_1^j & 0 \\ \sum_{k=0}^{j-1} T_2^k B T_1^{j-k-1} & T_2^j \end{pmatrix} = I$. This is a contradiction as neither of T_1^r and T_2^r is equal to I. Therefore, $j = kP_1P_2$. Now with $j = P_1P_2$ and applying Lemma 4.1.1, we have

$$\begin{aligned}
\sum_{k=0}^{j} T_2^k B T_1^{j-k} &= BT_1^j + T_2 BT_1^{j-1} + \ldots + T_2^{P_2-1} BT_1^{j-(P_2-1)} \\
&\quad + T_2^{P_2} BT_1^{j-P_2} + T_2^{P_2+1} BT_1^{j-(P_2+1)} + \ldots + T_2^{2P_2-1} BT_1^{j-(2P_2-1)} \\
&\quad \vdots \\
&\quad + T_2^{(P_1-1)P_2} BT_1^{j-(P_1-1)P_2} + \ldots + T_2^{P_1P_2-1} BT_1 + T_2^{P_1P_2} B \\
&= BT_1^j + T_2 BT_1^{j-1} + \ldots + T_2^{P_2-1} BT_1^{j-(P_2-1)} \\
&\quad + BT_1^{j-P_2} + T_2 BT_1^{j-(P_2+1)} + \ldots + T_2^{P_2-1} BT_1^{j-(2P_2-1)} \\
&\quad \vdots \\
&\quad + BT_1^{j-(P_1-1)P_2} + \ldots + T_2^{P_2-1} BT_1 + B
\end{aligned}$$

$$
\begin{aligned}
&= \left(B + T_2 B T_1^{-1} + \ldots + T_2^{P_2-1} B T_1^{-(P_2-1)}\right) \\
&\quad \left(T_1^j + T_1^{j-P_2} + \ldots + T_1^{j-(P_1-1)P_2}\right) + B \\
&= \left(\sum_{k=0}^{P_2-1} T_2^k B T_1^{-k}\right)\left(\sum_{k=0}^{P_1-1} T_1^{j-k}\right) + B \\
&= B.
\end{aligned}
$$

Therefore,

$$
\widetilde{T}^{j+1} = \begin{pmatrix} {T_1}^{j+1} & 0 \\ \sum_{k=0}^{j} T_2^k B T_1^{j-k-1} & {T_2}^{j+1} \end{pmatrix} = \begin{pmatrix} T_1 & 0 \\ B & T_2 \end{pmatrix} = \widetilde{T} \text{ and hence the period of } \widetilde{T} = P_1 P_2. \qquad \square
$$

Define the vector $S_k = (\mathbf{s}_k, \mathbf{s}_{k+1}, \ldots, \mathbf{s}_{n_1-1}, \mathbf{t}_k, \mathbf{t}_{k+1}, \ldots, \mathbf{t}_{n_2-1})^T$ for $k \geq 0$. Then $S_k = \widetilde{T} S_{k-1} = \widetilde{T}^k S_0$ for $k > 0$.

Theorem 4.1.3. *Let $MRMM_1(m_1, n_1)$ and $MRMM_2(m_2, n_2)$ be two primitive MRMMs such that $gcd(m_1 n_1, m_2 n_2) = 1$. If $[\boldsymbol{t}]$ is generated from the cascade connection of $MRMM_1$ into $MRMM_2$ and at least one state of $MRMM_1$ is nonzero, then $Per[t^{(j)}] = (2^{m_1 n_1} - 1)(2^{m_2 n_2} - 1)$ and $L[t^{(j)}] = m_1 n_1 + m_2 n_2$.*

Proof. It is not very hard to show that the periods of MRMM_1 and MRMM_2 are co-prime. Thus, by Theorem 4.1.2, the period of each component bit sequence is $(2^{m_1 n_1} - 1)(2^{m_2 n_2} - 1)$. Again from Eq. (4.5), the characteristic polynomial of $\widetilde{T}$ is the product of characteristic polynomials of T_1 and T_2. Since both the MRMMs are different and primitive, the characteristic polynomials of T_1 and T_2 are different and primitive. So the minimal polynomial and characteristic polynomials of $\widetilde{T}$ are the same. As the degrees of characteristic polynomials of T_1 and T_2 are $m_1 n_1$ and $m_2 n_2$, respectively, by [40, Lemma 1], the linear complexity of each component bit sequence is $m_1 n_1 + m_2 n_2$. This completes the proof. $\square$

Corollary 4.1.4. Suppose LFSR_1 and LFSR_2 are two binary primitive LFSRs of different orders n_1 and n_2, respectively. If $[t]$ is generated from the cascade connection of LFSR_1 into LFSR_2 and at least one state of LFSR_1 is nonzero, then the $Per[t] = (2^{n_1} - 1)(2^{n_2} - 1)$ and $L[t] = n_1 + n_2$.

Proof. The MRMM(m, n) becomes an LFSR with $m = 1$. Therefore, by Theorem 4.1.3 the result follows. □

Till now, the cascade connection of two primitive MRMMs is considered. This notion can be extended to $k \geq 2$ number of MRMMs as shown in Fig. 4.2.

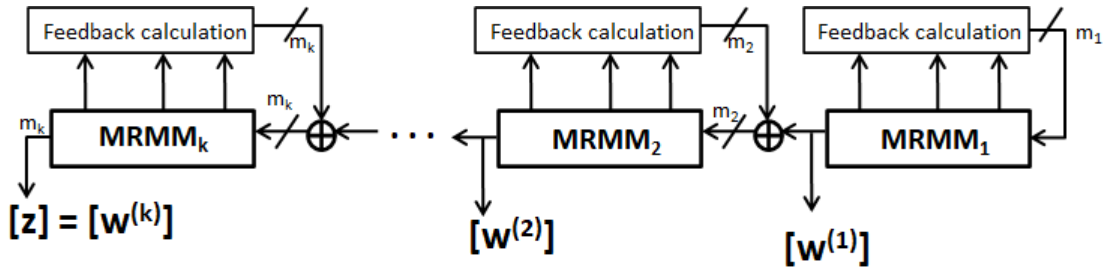

Figure 4.2: The cascade connection of k MRMMs

Theorem 4.1.5. *Consider cascade connection of k primitive MRMMs with periods $P_1, \ldots, P_k$. If $gcd(P_i, P_j) = 1$ for $i \neq j$ and at least one state of MRMM$_1$ is nonzero, then the period of output word sequence $\{\boldsymbol{z}_i\}_{i\geq 0}$ is $P_1 P_2 \ldots P_k$ and the linear complexity is $\sum_{i=1}^{k} log_2(P_i + 1)$.*

Proof. For $j = 1, 2, \ldots, k$, let $\{\mathbf{w}_i^{(j)}\}_{i\geq 0}$ be the word sequence produced after j^{th} MRMM as shown in the Fig. 4.2. Then by applying Theorem 4.1.5 recursively, the period of $\{\mathbf{w}_i^{(j)}\}_{i\geq 0}$ is $P_1 P_2 \ldots P_j$ for $2 \leq j \leq k$ and the linear complexity is $\sum_{i=1}^{k} log_2(P_i + 1)$. □

Till this point, it is assumed that the periods of MRMMs used in the cascade connection are mutually co-prime. In the following, a result is derived when the period is the same.

Theorem 4.1.6. *For a cascade connection of two primitive MRMMs having the same matrix polynomial with period P, the period of the matrix $\widetilde{T}$ as defined in Eq. (4.5) divides $2P$.*

Proof. By Theorem 4.0.1, the period of $\widetilde{T}$ is a multiple of P. Again,

$$\widetilde{T}^{2P+1} = \begin{pmatrix} T_1 & 0 \\ \sum_{k=0}^{2P} T_1^k B T_1^{2P-k} & T_2 \end{pmatrix}.$$

But,

$$\begin{aligned}\sum_{k=0}^{2P} T_1^k B T_1^{2P-k} &= BT_1^{2P} + T_1 BT_1^{2P-1} + \ldots + T_1^{P-1} BT_1^{P+1} + T_1^P BT_1^P \\ &+ T_1^{P+1} BT_1^{P-1} + \ldots + T_1^{2P-1} BT_1 + T_1^{2P} B \\ &= B + T_1 BT_1^{P-1} + \ldots + T_1^{P-1} BT_1 + B \\ &+ T_1 BT_1^{P-1} + \ldots + T_1^{P-1} BT_1 + B \\ &= B.\end{aligned}$$

Therefore, $\widetilde{T}^{2P+1} = \begin{pmatrix} T_1 & 0 \\ B & T_1 \end{pmatrix} = \widetilde{T}$. As $\widetilde{T}$ is invertible, $\widetilde{T}^{2P} = I$ and so the period of $\widetilde{T}$ must divide $2P$. □

4.2 Randomness properties of CS based bitstreams

In this section, we analyze the randomness properties of the bitstreams generated by three MRMM-based CSs. We call CS as CS_1 (or CS_2) when $\oplus$ (or $+$) is used for $\odot$ in Eq. (4.2). The CS_1 and CS_2 are depicted in Fig. 4.3 and Fig. 4.4, respectively. In both cases, g is considered as an identity map. The third cascade system CS_3 is same as CS_1 with $g(.)$ as a nonlinear bijection function S box. The S box is given in Appendix A. For experimental study in each CS, both FSRs are taken primitive MRMMs defined over $\mathbb{F}_{2^{16}}$ i.e., $m = 16$. The order of $MRMM_1$ and $MRMM_2$ is 10 and 7, respectively. The matrix polynomial of $MRMM_1$ and $MRMM_2$ are $x^{10} + R_6 x^8 + LCS_8 x^7 + R_8 x^6 + 62924x^5 + R_9 x^4 + R_4 x^1 + LCS_1$ and $x^7 + R_{11} x^4 + LCS_8 x^3 + L_9 x^2 + R_5 x^1 + LCS_{14}$, respectively. Here R_a is the right shift operator defined as $R_a(X) = X >> a$ whereas L_b is the left shift operator defined as $L_b(X) = X << b$ and the left circular shift of X by c-bit is denoted as $LCS_c(X)$. As $MRMM_1$ runs in free running mode, we avoid all zero-state initialization situations in $MRMM_1$. It is observed that the results of the statistical test suite are similar for several random initializations of the states of $MRMM_2$, including all zero initialization states. This occurs as the states of

$MRMM_1$ are injected into the states of $MRMM_2$ through its feedback calculation. Thus, the states of $MRMM_2$ can be initialized with any random values, but for our experimental set up we always initialize with zero values.

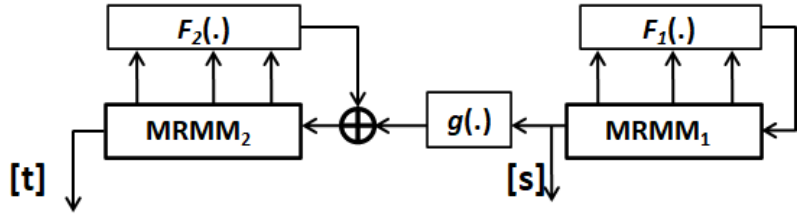

Figure 4.3: CS_1

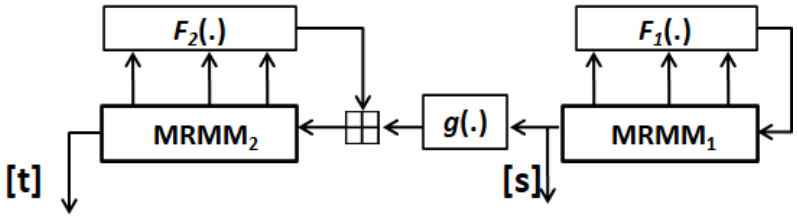

Figure 4.4: CS_2

4.2.1 Comparison of statistical properties of CS_1, CS_2 and CS_3

Many statistical test suites have been developed to study and analyze the randomness properties of a bitstream. In our experiment, we use NIST statistical test suite SP 800-22 Rev.1a [1]. To see the randomness properties of the word sequences generated by CS_1 and CS_2, we first convert word sequence $[\mathbf{t}]$ to bit sequence. By Theorem 4.0.1, $(2^{160} - 1)$ is a factor of the period of the word sequence, so the period of the corresponding bit sequence will be at least $16(2^{160} - 1)$ as each word is 16-bit wide. We investigate the randomness properties of the bitstream of CS_1 as follows:

1. Initialization of the states $\mathbf{S}_0$ and $\mathbf{T}_0$: Initialize all the states of $MRMM_1$ and $MRMM_2$ to $\mathbf{0}$ except $\mathbf{s}_0 = 1$. Run CS_1 for 48 iterations without collecting any bitstream. The reason for the selection of 48 rounds is explained in Section 4.2.2.

2. Then at the end of each iteration, collect 16-bit keystream as $\mathbf{t}_0$. In this experiment,

it generates 100 bitstream files and each file consists of a bitstream of length 1000000. As bitstreams are not collected for initial 48 rounds, CS_1 runs $(62500 \times 100) + 48$ times to generate these 100 files.

3. Then, NIST statistical test suite is performed on all those 100 bitstream files. For this experiment, the significant value α is taken 0.05.

Similar to CS_1, 100 bitstream files are generated for CS_2 and CS_3, then NIST statistical test suite is performed. Note that both CS_1 and CS_2 are the same except for different operations $\odot$ used in Eq. (4.2). The comparative statistical results are provided in Table 4.1. The first column tells the name of each statistical test of the NIST test suite. The second, third, and fourth columns show the number of passed files out of 100 in the case of CS_1, CS_2, and CS_3, respectively.

As [**s**] is generated by a primitive MRMM, it has good randomness properties. But it possesses low linear complexity due to linear structure. These randomness properties will be inherited by CS_1 as [**s**] is XORed in the generation of [**t**]. This fact is reflected in the second column of Table 4.1. It is checked that the linear complexity of all files of CS_1 is $4352 = m^2(n_1 + n_2)$ as expected. Thus, none of the bitstreams of length 1000000 passes the linear complexity test as the expected linear complexity is 500000. Table 4.1 shows that statistical results on the bitstream of CS_1 and CS_2 are very close except for the linear complexity. In the case of the bitstream of CS_2, it is tested that linear complexity is around 500000 for all 100 bitstreams of length 1000000. This is due to use of one modular addition[1]. Due to the presence of a nonlinear S box in CS_3, the bitstreams of CS_3 have also good statistical properties like CS_2. Therefore, both CS_2 and CS_3 have an advantage over CS_1 with respect to randomness property. The S box used in CS_3 is generated by the S box construction method applied in the AES block cipher [38].

[1]It is known that if $a = \sum_{i=0}^{m-1} a_i 2^i$, $b = \sum_{i=0}^{m-1} b_i 2^i$ and $c = a + b = \sum_{i=0}^{m-1} c_i 2^i$, with $a_i, b_i, c_i \in \{0, 1\}$, then $c_0 = a_0 \oplus b_0$, and for $1 \leq k < m$, $c_k = a_k \oplus b_k \oplus \sum_{i=0}^{k-1} a_i b_i \prod_{j=i+1}^{k-1} (a_j \oplus b_j)$. As the value of word size m is 16, the nonlinear degree of the expression of c_{15} will be 16.

Name of the test	No. of passed files CS_1	No. of passed test CS_2	No. of passed test CS_3
Frequency	95	95	97
BlockFrequency	96	99	99
Runs	98	94	97
LongestRunOfOnes	98	97	94
Rank	93	94	92
DiscreteFourierTransform	92	90	94
NonOverlappingTemplateMatchings	100	100	100
OverlappingTemplateMatchings	93	95	94
Universal	94	97	94
LinearComplexity	0	100	100
Serial	98	94	97
ApproximateEntropy	97	93	95
CumulativeSums	95	96	96
RandomExcursions	60	59	64
RandomExcursionsVariant	60	58	62

Table 4.1: Comparative statistical results of CS_1 and CS_2

4.2.2 Avalanche analysis on state registers of CS_1, CS_2 and CS_3

In this section, we analyze the avalanche effect on the states of three CSs. For the avalanche analysis study, we kept the same MRMMs used in the previous section for statistical analysis. After initialization of $\mathbf{S}_0$ and $\mathbf{T}_0$, we compute experimentally the number of iterations needed to achieve avalanche property, i.e., 50% bit changes in the states of each of three CSs separately. It is observed that number of iterations to achieve 50% avalanche in the case of CS_1 and CS_2 is very close for any random initialization of $\mathbf{S}_0$ and $\mathbf{T}_0$. However, depending upon the total weight of the initial states of CS, the number of iterations varies to achieve 50% avalanche effect as shown in Table 4.2 and Table 4.3. The steps of the experiment to generate Table 4.2 are as follows.

1. Set $count = 0$.

2. Initialize the states of MRMM_1 and MRMM_2: $\mathbf{s}_0 = \textbf{ini-s}_0 = 1$ and $\mathbf{s}_k = \textbf{ini-s}_k = \mathbf{0}$ for $0 < k \leq n_1 - 1$. Here $\textbf{ini-s}_k$ is the initial state of $\mathbf{s}_k$. Initialize $\mathbf{t}_k = \textbf{ini-s}_k = \mathbf{0}$ for $0 \leq k \leq n_2 - 1$

3. Run MRMM_1 for one iteration. Here one iteration means it calculates the feedback value fb_1 of MRMM_1, then shift the states i.e., $\mathbf{s}_k = \mathbf{s}_{k+1}$ for $0 \leq k < n_1 - 1$ and $\mathbf{s}_{n_1-1} = fb_1$.

4. Run MRMM_2 for one iteration i.e., it calculates the feedback value fb_2 of MRMM_2, then shift the states i.e., $\mathbf{t}_k = \mathbf{t}_{k+1}$ for $0 \leq k < n_2 - 1$ and $\mathbf{t}_{n_2-1} = fb_2 \odot \mathbf{s}_0$.

5. Calculate $weight = \sum_{k=0}^{n_1-1} Wt(\mathbf{s}_k \oplus \textbf{ini-s}_k) + \sum_{k=0}^{n_2-1} Wt(\mathbf{t}_k \oplus \textbf{ini-t}_k)$. Here $Wt(\boldsymbol{x})$ is the hamming weight of the word $\boldsymbol{x}$.

6. If $\mid weight - \frac{m(n_1+n_2)}{2} \mid \geq 3$, increase *count* by 1 and go to Step 3. Otherwise, return *count*. Here 3 is the approximation value of 1% of the total size of memory states i.e., $m(n_1 + n_2)$.

	CS_1		CS_2		CS_3	
Iter. No.	Wt. of states	avalanche effect %	Wt. of states	avalanche effect %	Wt. of states	avalanche effect %
0	3	1.10	3	1.10	9	3.31
1	3	1.10	3	1.10	17	6.25
5	7	2.57	7	2.57	47	17.28
10	18	6.62	18	6.62	66	24.26
15	35	12.87	36	13.24	71	26.10
20	62	22.79	62	22.79	94	34.56
30	119	43.75	120	44.12	123	45.22
40	124	45.59	120	44.12	124	45.59

Table 4.2: Avalanche effects vs iteration numbers when all states except $\mathbf{s}_0$ are $\mathbf{0}$

Table 4.2 displays the total weight of the states and the percentage of total states size of CS_1, CS_2, and CS_3, respectively with respect to the iteration number. Here the percentage

	CS_1		CS_2		CS_3	
Iter. No.	Wt. of states	avalanche effect %	Wt. of states	avalanche effect %	Wt. of states	avalanche effect %
0	23	8.46	23	8.46	17	6.25
1	46	16.91	46	16.91	34	12.50
2	68	25.00	68	25.00	50	18.38
3	90	33.09	83	30.51	68	25.00
4	104	38.24	107	39.34	80	29.41
5	125	45.96	124	45.59	97	35.66
6	144	52.94	144	52.94	115	42.28

Table 4.3: Avalanche effects vs iteration numbers when all states of $MRMM_1$ are 0xFFFF and all states of $MRMM_2$ are **0**

of avalanche effect is calculated as (*total weight of the states of CS)/(total size of CS in bits)* $\times$ *100*. In this case, the total size of CS in bits is 272. It is observed experimentally that both CS_1 and CS_2 take almost the same number of iterations to achieve 50% avalanche effect for the same nonzero random initialization of states.

Table 4.3 shows that around 6 iterations are needed to achieve the desired effect. In this case, data are generated after all states of $MRMM_1$ are initialized with the hex value 0xFFFF i.e., all bits of $MRMM_1$ are 1, and all states of $MRMM_2$ are kept 0. It is also noticed that if the states of $MRMM_1$ and $MRMM_2$ are balanced, then CS takes around 9 iterations to achieve 50% avalanche effect. Except $\mathbf{s}_0$, if all states of CS_1 and CS_2 are in zero states, both CS_1 and CS_2 achieve the avalanche property after 48 iterations. Therefore, during the study of statistical properties of CS_1 and CS_2 in Section 4.2.1 we have collected bitstream after 48 iterations.

4.2.3 Cascade connection of MRMM and LFG

In the previous section, we have studied cascade systems comprised of MRMMs only. In this section, we focus on two more cascade systems. One is CS_4 consists of two LFGs and the other is CS_5 where one FSR is MRMM and the other is LFG. Suppose both the LFGs

used in CS_4 are two primitive additive LFGs where LFG_1 is cascaded into LFG_2. If $[\mathbf{t}]$ is a nonzero sequence generated by this cascade system, then we have the following result.

Theorem 4.2.1. *Let the order of LFG_1 and LFG_2 be n_1 and n_2, respectively having common word size m. If $gcd(n_1, n_2) = 1$, then period of $[\boldsymbol{t}]$, $Per[\boldsymbol{t}] = (2^{n_1} - 1)(2^{n_2} - 1)2^{m-1}$.*

Proof. Suppose $[\mathbf{s}]$ is the word sequence generated by LFG_1. Then $[\mathbf{s}]$ is periodic as LFG_1 is primitive and $Per[\mathbf{s}] = (2^{n_1} - 1)2^{m-1}$. Again the feedback function $f_2(.)$ of LFG_2 is nonsingular therefore, $(2^{n_1} - 1)2^{m-1}$ divides $Per[\mathbf{t}]$ by Theorem 4.0.1. Let $[t^{(1)}]$ be the bit sequence consisting of the first least significant bit (LSB) of each word of $[\mathbf{t}]$. Then, it can be seen that $[t^{(1)}]$ is generated by the cascade connection of two primitive LFSRs. Thus $Per[t^{(1)}] = (2^{n_1} - 1)(2^{n_2} - 1)$ by [9, Corollary 14]. This implies $(2^{n_1} - 1)(2^{n_2} - 1)2^{m-1}$ divides $Per[\mathbf{t}]$ as $Per[t^{(1)}]$ divides $Per[\mathbf{t}]$.

Again, it is obvious that $Per[\mathbf{t}]$ divides $lcm((2^{n_1} - 1)2^{m-1}, (2^{n_2} - 1)2^{m-1})$ as both the LFGs are primitive. As $gcd(n_1, n_2) = 1$, $lcm((2^{n_1} - 1)2^{m-1}, (2^{n_2} - 1)2^{m-1}) = (2^{n_1} - 1)(2^{n_2} - 1)2^{m-1}$. This proves the desired result. □

Since the period of CS_4 is $(2^{n_1} - 1)(2^{n_2} - 1)2^{m-1}$, the lower bound for linear complexity of any nonzero bitstream of CS_4 is $n_1 n_2 2^{m-2}$. In CS_5 LFG cascades into MRMM. Then using similar arguments, the following theorem can be proved.

Theorem 4.2.2. *Let the order of LFG and MRMM be n_1 and n_2, respectively having common word size m. If $gcd(n_1, mn_2) = 1$, then $Per[\boldsymbol{t}] = (2^{n_1} - 1)(2^{mn_2} - 1)2^{m-1}$.*

4.2.4 Cryptanalysis of word-based CSs

In the previous two sections, we have studied cascade systems comprised of MRMMs and LFGs, then discussed some of their statistical properties. In this section, we discuss a cryptanalytic attack and its complexity. We start with a general method for reconstructing the original sequence from a known portion of the output word sequence. Assume that $[\mathbf{t}]$ is the known output word sequence of the cascade system of Eq. (4.2) where the function $g(.)$ is the Identity function. Thus, $\mathbf{t}_{i+n_2} = \mathbf{s}_i \odot F_2(\mathbf{t}_i, \mathbf{t}_{i+1}, \ldots, \mathbf{t}_{i+n_2-1})$, for $i \geq 0$. Our

aim is to reconstruct the initial states of both the FSRs, $\mathbf{S}_0$ and $\mathbf{T}_0$. Since the procedure of cryptanalytic attack is similar for all discussed cascade systems comprised of MRMMs and LFGs except CS_3, we only focus on CSs of LFGs i.e., CS_4. Suppose LFG_1 is cascaded into LFG_2 in the CS and both LFGs are primitive. Let $[t^{(1)}]$ be the bit sequence consisting of the first LSB of each word of $[\mathbf{t}]$. Then, $[t^{(1)}]$ is a bit sequence generated by a cascade system of two primitive LFSRs. If n_1 and n_2 are the order of LFG_1 and LFG_2, then from any $2(n_1+n_2)$ consecutive bits of $[t^{(1)}]$, it is possible by Berlekamp-Massey algorithm [35], to get the feedback polynomials of both LFSRs and therefore, for LFGs. Now from Eq. (4.2), $\mathbf{s}_i = (\mathbf{t}_{n_2+i} - F_2(\mathbf{t}_i, \mathbf{t}_{i+1}, \ldots, \mathbf{t}_{n_2-1+i}))\ mod\ 2^m$, for $i \geq 0$ and so the initial states $\mathbf{s}_0, \ldots, \mathbf{s}_{n_1-1}$ can be computed, once $[\mathbf{t}]$ is known.

Since $\mathbf{t}_0, \ldots, \mathbf{t}_{n_2-1}$ are assumed to be part of the secret key, they should not be used as keystream words. Let us assume the keystream words $\{\mathbf{t}_{n_2+i}\}_{i\geq 0}$ are known. Then using the Berlekamp-Massey algorithm on $[t^{(1)}]$, it is again possible to find the feedback polynomial of both the LFGs. Now $\mathbf{s}_{n_2+i}$ can be computed as $\mathbf{s}_{n_2+i} = (\mathbf{t}_{2n_2+i} - F_2(\mathbf{t}_{n_2+i}, \ldots, \mathbf{t}_{2n_2-1+i}))$ $mod\ 2^m$, for $i \geq 0$. Further, $\mathbf{s}_{n_2+n_1-1} = F_1(\mathbf{s}_{n_2-1}, \mathbf{s}_{n_2}, \ldots, \mathbf{s}_{n_2+n_1-2})$ and $F_1(.)$ is linear, thus $\mathbf{s}_{n_2-1}$ can be calculated from it. Using the same procedure, it is possible to retrieve all initial states of LFG_1. Here also same $2(n_1+n_2)$ number of consecutive words of $[\mathbf{t}]$ is sufficient to mount this attack. This attack is summarized as follows.

Algorithm 4 : Attack on LFG-based cascade systems

Input: The output word sequence $[\mathbf{t}]$.
Output: Recover the initial states of LFGs.

1: If $[\mathbf{t}] \neq 0$, then collect $[t^{(1)}]$, the first LSB of each word of $[\mathbf{t}]$. Apply the Berlekamp-Massy algorithm on $[\mathbf{t}]$ and get the feedback polynomial of LFGs. Using Eq. (4.2), calculate the initial states of LFGs.
2: Else, All states of LFGs are $\mathbf{0}$.

This attack is applicable as long as the bits of $[t^{(1)}]$ satisfy the linear recurring relation of order (n_1+n_2) or less. One of the easy solutions is to use a nonlinear bijective function $g(.)$ to destroy the linear structure of $[t^{(1)}]$, for example, the use of an S box. Thus, this cryptanalytic attack is not applicable to CS_3. In the case of an LFG-based cascade system, simply rotation by a nonzero value will also destroy the linear relation in the 1st LSB of $[\mathbf{t}]$.

4.3 Conclusion

In summary, this chapter presents several word-oriented cascade systems comprised of two FSRs only. The concept can be extended to more than two FSRs. We discuss three different MRMM-based cascade systems and study their periodicity and randomness properties, then present avalanche analysis on the states of all three cascade systems. We compute the periodicity of cascade systems of MRMM and LFG. Lastly, a cryptanalytic attack on the cascade systems is discussed along with its countermeasure. The results pertaining to generalized WFSR and cascade generators are submitted to journals[2,3].

In the next chapter, we discuss another two word-oriented generators.

[2]S. K. Bishoi, K. Senapati and B. R. Shankar, *Bitstream generators using Multiple-Recursive Matrix Methods*, Sept 2022, doi: https://doi.org/10.21203/rs.3.rs-2105578/v1.

[3]S. K. Bishoi, K. Senapati and B. R. Shankar, *Generalized word-oriented feedback shift registers.*

Chapter 5

Word-oriented alternating step generators and nonlinear combination generators

In this section, we discuss two more WFSRs based on alternating step generators and nonlinear combination generators, respectively.

5.1 Alternating step generators

Similar to shrinking and self-shrinking generators [8], the alternating step generator (ASG) [27] is highly appealing due to its conceptual simplicity. The ASG is a sequence generator composed of three FSRs A, B, and C as illustrated in Fig. 5.1. Suppose the bit sequences produced by FSRs A, B, C are $[a], [b], [c]$, respectively. In ASG, three generators are interconnected. In i^{th} cycle, B is clocked (stepped) once and C is not clocked if $a_i = 1$, otherwise B is not clocked and C is clocked once. Here, A is called the control register whereas B and C are called generating registers. Let $[z]$ be the output bit sequence of the ASG. In this section, we study three variants of ASGs named ASG_1, ASG_2, and ASG_3, respectively. In all those three cases, A is an FSR producing a de Bruijn sequence, whereas FSRs B and C

are considered as follows

(1) ASG_1: FSRs B and C are two different primitive LFSRs.

(2) ASG_2: Between FSRs B and C, one generates a de Bruijn sequence and the other is a primitive LFSR.

(3) ASG_3: Both B and C produce de Bruijn sequences.

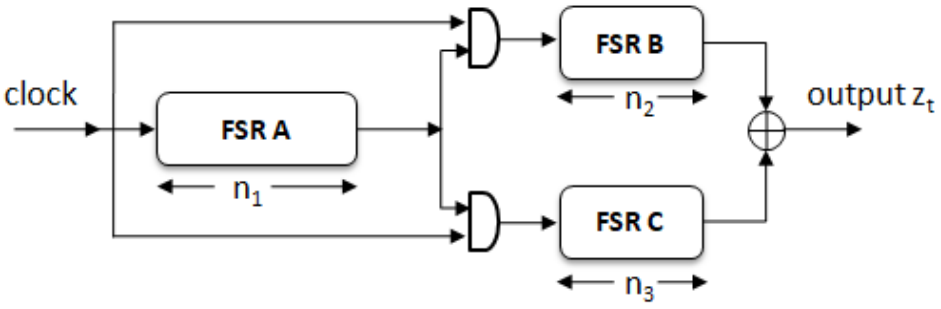

Figure 5.1: Alternating step generator

5.1.1 ASG_1

This case is discussed in Gunther [27, Theorem 1] and the result is as follows

Proposition 5.1.1. Suppose $[a]$ is a de Bruijn sequence of span n_1, $[b]$ and $[c]$ are two primitive sequences of orders n_2 and n_3, respectively. If $gcd(n_2, n_3) = 1$, then $Per[z]$ and $L[z]$ satisfy the following:

(i) $Per[z] = 2^{n_1}(2^{n_2} - 1)(2^{n_3} - 1)$

(ii) $(n_2 + n_3)2^{n_1-1} < L[z] \leq (n_2 + n_3)2^{n_1}$

5.1.2 ASG_2

Without loss of generality, assume that B is the FSR producing a de Bruijn sequence and C is a primitive LFSR. Then the following result tells about the periodicity and linear complexity of the bit sequence generated by this modified bitstream generator.

Proposition 5.1.2. Suppose $[a]$ and $[b]$ are two de Bruijn sequences of spans n_1 and n_2, respectively and $[c]$ is a maximal sequence of order n_3, then $Per[z]$ and $L[z]$ satisfy the following:

(i) $Per[z] = 2^n(2^{n_3} - 1)$, where $n = max(n_1, n_2 + 1)$

(ii) $2^{n-1} + n_3 2^{n_1 - 1} < L[z] \leq 2^n + n_3 2^{n_1}$

Proof. The output sequence $[z]$ can be expressed as $[z] = [\tilde{b}] \oplus [\tilde{c}]$, where $[\tilde{b}]$ and $[\tilde{c}]$ are generated by the sub-generators whose component sequences are $[a], [b]$ and $[a], [c]$, respectively [30]. For the sequence $[\tilde{b}]$, it repeats whenever the states of A and B return to their respective initial states. As $[a]$ and $[b]$ are de Bruijn sequences, $Per[a] = 2^{n_1}$ and $Per[b] = 2^{n_2}$. If $n_1 \leq n_2$, then $Per[\tilde{b}] = 2^{n_2+1}$ as A has to be clocked for 2^{n_2+1} times to get 2^{n_2} number of 1s. Otherwise, if $n_1 > n_2$, then $Per[\tilde{b}] = 2^{n_1}$. Therefore, $Per[\tilde{b}] = 2^n$, where $n = max(n_1, n_2 + 1)$. Again, $Per[\tilde{c}] = 2^{n_1}(2^{n_3} - 1)$ as proved in [30, Lemma 1]. Thus, $Per[z] = lcm(2^n, 2^{n_1}(2^{n_3} - 1)) = 2^n(2^{n_3} - 1)$.

As $Per[\tilde{b}] = 2^n$, by [13, Theorem 1] the minimal polynomial of $[\tilde{b}]$ is $(1 + x)^\alpha$ for $2^{n-1} < \alpha \leq 2^n$ and so $L[\tilde{b}]$ satisfies $2^{n-1} < L[\tilde{b}] \leq 2^n$. Also the minimal polynomial of $[\tilde{c}]$ is $J(x)^\beta$ for $2^{n_1-1} < \beta \leq 2^{n_1}$, where $J(x)$ is a minimal polynomial of degree n_3. Thus by [30, Lemma 4], $n_3 2^{n_1-1} < L[\tilde{c}] \leq n_3 2^{n_1}$. Again the minimal polynomial of $[z]$ is $(1 + x)^\alpha J(x)^\beta$ of the degree $(\alpha + n\beta)$ [32, Theorem 6.57], and the desired result follows. □

5.1.3 ASG$_3$

Proposition 5.1.3. Suppose $[a], [b]$ and $[c]$ are three de Bruijn sequences of spans n_1, n_2 and n_3, respectively, then $Per[z]$ and $L[z]$ satisfy the following:

(i) $Per[z] = 2^n$, where $n = max(n_1, n_2 + 1, n_3 + 1)$

(ii) $2^{n-1} < L[z] \leq 2^n$

Proof. In Proposition 5.1.2, It is shown that $Per[\tilde{b}] = 2^{n'}$, where $n' = max(n_1, n_2 + 1)$. Similarly, $Per[\tilde{c}] = 2^{n''}$, where $n'' = max(n_1, n_3 + 1)$. Therefore, $Per[z] = 2^n$, where

$n = max(n^{'}, n^{''}) = max(n_1, n_2+1, n_3+1)$. Then by [13, Theorem 1], we have $2^{n-1} < L[z] \leq 2^n$. □

5.1.4 Observations and word-based ASG

In the above three cases of ASGs, it is clear that $Per[z]$ of ASG_1 is greater than that of ASG_2 and ASG_3. Also, the lower bound for $L[z]$ is more in case of ASG_1, if n_1 is greater than n_2 and n_3. This motivates us to extend the bit-oriented ASG_1 to word-oriented ASG (W-ASG). In W-ASG the controlling FSR A is a de Bruijn sequence as in the ASG and the generating FSRs B and C are two primitive MRMMs. Consider B is $\text{MRMM}_2(m_2, n_2)$ whereas C is $\text{MRMM}_3(m_3, n_3)$. If $m = min(m_2, m_3)$, then in each cycle i, W-ASG produces m-bit $\mathbf{z}_i$, where $\mathbf{z}_i$ is Xor of m-bit output word of B and m-bit output word of C. As described earlier, let $[z^{(j)}] = \{\mathbf{z}_0^{(j)}, \mathbf{z}_1^{(j)}, \ldots, \}$ be the j^{th} bit sequence, for $1 \leq j \leq m$. Now we have the following result.

Proposition 5.1.4. In W-ASG,if $gcd(m_2n_2, m_3n_3) = 1$, then for $1 \leq j \leq m$

(i) $Per[z^{(j)}] = 2^{n_1}(2^{m_2n_2} - 1)(2^{m_3n_3} - 1)$

(ii) $(m_2n_2 + m_3n_3)2^{n_1-1} < L[z^{(j)}] \leq (m_2n_2 + m_3n_3)2^{n_1}$

Proof. Suppose $[\mathbf{b}]$ and $[\mathbf{c}]$ are word sequences generated by MRMM $_2(m_2, n_2)$ and MRMM $_3(m_3, n_3)$, respectively. Then $[z^{(j)}]$ is the bit sequence produced by the ASG($[a]$, $[b^{(j)}]$, $[c^{(j)}]$). Therefore, by Proposition 5.1.1 the desired result follows.

□

Example 5.1.5. Consider the bit-oriented ASG(A,B,C), where A is an FSR producing a binary de Bruijn sequence 1111010010110000 of period 16, whereas B and C are primitive LFSRs with characteristic polynomials $f_1(x) = x^{16} + x^{15} + x^{14} + x^{13} + x^{11} + x^{10} + x^6 + x^5 + x^3 + x^2 + 1$ and $f_2(x) = x^{15} + x^{13} + x^{11} + x^9 + x^8 + x^7 + x^6 + x^2 + 1$, respectively. Then after initializing B and C with nonzero states, it is experimentally computed that the period of ASG(A,B,C) is $2^4(2^{16} - 1)(2^{15} - 1)$ and the linear complexity is 124 which satisfy the constraints given in Proposition 5.1.1.

Using the polynomial $f_1(x)$ and $f_2(x)$ as input to construction algorithm for primitive MRMM [5], compute $\text{MRMM}_2(4,4)$ and $\text{MRMM}_3(5,3)$. In this example, $m = min(m_1, m_2) = 4$. So W-ASG produces a 4-bit number in each round, whereas ASG produces 1-bit only. Thus, W-ASG generates bitstream at a faster rate, especially in processors. It is verified through C-programming that both the period and linear complexity of all 4 component-bit sequences are the same. If we consider the word sequence as a bit sequence, then it is observed that both the period and the linear complexity of W-ASG are m times that of bit-oriented ASG.

5.2 Nonlinear combination generators

The nonlinear combination generator is another technique to destroy the linearity in the case of LFSR-based ciphers. Numerous works have been reported on the problem of controlling the linear complexity of sequences based on nonlinear combinations. In the case of products of the LFSR-sequences [24, 31], the product of the linear complexities of two sequences is the upper bound to the linear complexity of their product sequence. If the linear complexity of the product sequence is equal to the upper bound, then it is called that the product sequence attains the maximum linear complexity. Here, the combiner function is $f(x_1, x_2) = x_1x_2$. The idea behind the bit-oriented nonlinear combination generator is to use a nonlinear Boolean function $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2$ to destroy the linearity inherent in LFSRs [53]. The nonlinear combination generator is depicted in Fig. 5.2. In every iteration, it takes n inputs from n independent LFSRs, and using the Boolean nonlinear combiner function f, it produces 1-bit output. We generalize these bit-oriented nonlinear combination generators to word-oriented nonlinear combination generators as shown in Fig. 5.3.

A Boolean function in n variables is a function $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2$ [53]. One way to represent a Boolean function is to express the function in terms of Boolean variables. When a Boolean function is written as

$$f(x_1, x_2, \ldots, x_n) = c_0 \bigoplus_{1 \le i \le n} c_i x_i \bigoplus_{1 \le i < j \le n} c_{ij} x_i x_j \oplus \ldots \oplus c_{1,\ldots,n} x_1 x_2 \ldots x_n, \tag{5.1}$$

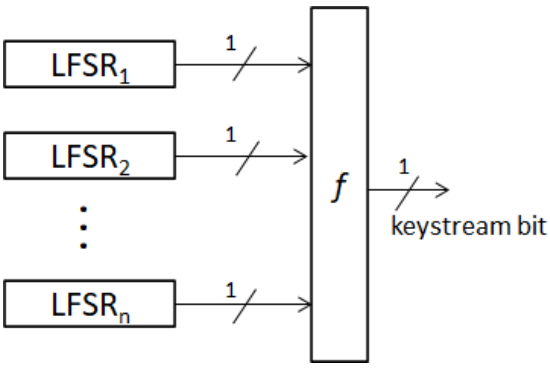

Figure 5.2: Bit-oriented nonlinear combination generator

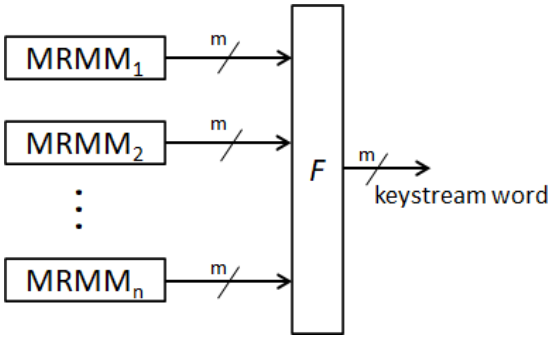

Figure 5.3: Word-oriented nonlinear combination generator

it is called the algebraic normal form representation of f, where $c_0, c_i, c_{ij}, \ldots, c_{1,\ldots,n}$ are coefficients in $\mathbb{F}_2$. Now, for a Boolean function $f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2$, we define $F : \mathbb{F}_{2^m}^n \longrightarrow \mathbb{F}_{2^m}$ as follows:

$$F(X_1, X_2, \ldots, X_n) = c_0 \underset{1\leq i\leq n}{\oplus} c_i X_i \underset{1\leq i<j\leq n}{\oplus} c_{ij} X_i X_j \oplus \ldots \oplus c_{1,\ldots,n} X_1 X_2 \ldots X_n \tag{5.2}$$

Here the product among the variables $X_1, X_2, \ldots, X_k$, for $2 \leq k \leq n$ is the Hadamard product. We call F as the vectorial form of f and f as the scalar form of F. With the function F, the concept of a bit-oriented nonlinear combination generator can be extended to a word-oriented nonlinear combination generator.

Let $\text{MRMM}_k(m_k, n_k)$ be the k^{th} MRMM and $[\mathbf{s}_k]$ be the word sequence generated by $\text{MRMM}_k(m_k, n_k)$ for $1 \leq k \leq n$. Let $m = min(m_1, m_2, \ldots, m_n)$ and $F : \mathbb{F}_{2^m}^n \longrightarrow \mathbb{F}_{2^m}$ be the nonlinear combining function as defined in the Eq. (5.2) with f as its scalar form. The function F takes m bit word from each $\text{MRMM}_k(m_k, n_k)$ as input and produces m bit word as output.

Similar to all the LFSRs in bit-oriented nonlinear combination generators, it is considered that all the MRMMs are running in free-running mode. Let $\{\mathbf{s}_{k,0}, \mathbf{s}_{k,1}, \ldots, \mathbf{s}_{k,n_k-1}\}$ be the initial states of $\text{MRMM}_k(m_k, n_k)$. In each iteration i, each MRMM_k generates a word $\mathbf{s}_{k,i}$. Consider $\mathbf{t}_{k,i}$ is the m lsb of $\mathbf{s}_{k,i}$. If $\mathbf{z}_i = F(\mathbf{t}_{1,i}, \mathbf{t}_{2,i}, \ldots, \mathbf{t}_{n,i})$ is the i^{th} keystream word, then the following result on the periodicity and the linear complexity of the sequences $[z^{(j)}], 1 \leq j \leq m$

is obtained.

Theorem 5.2.1. *Suppose $MRMM_k(m_k, n_k)$ for $1 \leq k \leq n$ are n primitive MRMMs and let $m_1n_1, m_2n_2, \ldots, m_nn_n$ be pairwise co-prime and greater than 2, then $Per[z^{(j)}] = \prod_{i=1}^{n}(2^{m_in_i} - 1)$ and $L[z^{(j)}] = f(m_1n_1, m_2n_2, \ldots, m_nn_n)$ for $1 \leq j \leq m$.*

Proof. Let $[t_k^{(j)}]$ be the j^{th} lsb bit sequence obtained from the word sequence $[\mathbf{t}_k]$, then for $1 \leq j \leq m$ and $1 \leq k \leq n$, the relation between the bit sequences $[z^{(j)}]$ and $[t_k^{(j)}]$ is as follows:

$$z_i^{(j)} = f(t_{1,i}^{(j)}, t_{2,i}^{(j)}, \ldots, t_{n,i}^{(j)}), \tag{5.3}$$

By Lemma 2.3.2, each nonzero bit sequence $[t_k^{(j)}]$ attains maximal period and therefore primitive. Thus, by [23, Theorem 5] and [45, Theorem 3] the desired result follows. □

In the case of combination generators, the constituent LFSRs and MRMMs should be primitive to ensure good statistical properties of their output sequences. Again, the keystream sequence must have high linear complexity to counter the attack due to the Berlekamp-Massey algorithm. Thus, the combining function f or F should have a higher algebraic degree. However, the combination generators in the case of LFSRs and MRMMs are vulnerable to both correlation attack [49] and fast correlation attack [36]. To make these attacks infeasible, the feedback polynomials should not be sparse, and the combining function should have a high resiliency (correlation-immune and balance) order. But, there is a trade-off between the correlation-immunity order and the algebraic degree of a Boolean function due to Siegenthaler bound.

The results pertaining to the word-oriented alternate step generators and nonlinear combination generators are submitted to journal[1] .

[1]S. K. Bishoi, K. Senapati and B. R. Shankar, *Bitstream generators using Multiple-Recursive Matrix Methods*, Sept 2022, doi: https://doi.org/10.21203/rs.3.rs-2105578/v1

Chapter 6

Conclusion

Along with good statistical properties, a cryptographically secure bitstream generator must have a large period and high linear complexity. This thesis has explored the possibility of increasing linear complexity and we have introduced a few new techniques to achieve the same. They are shrinking generator, self-shrinking generator, cascade generator, alternating step generator, and nonlinear combination generator using MRMMs. We present some results pertaining to the periodicity and statistical properties of the bitstream generated by self-shrinking primitive MRMMs. Since both period and linear complexity are large in the case of the above word-oriented generators, they could be used as one of the primitives in the design of symmetric-key crypto algorithm, especially when the crypto algorithm is to be implemented in some processor.

Since both Xorshift RNG and TSR are a particular class of MRMM, all the above results pertaining to MRMM are held for them also. We have shown that the period is exponential, whereas maximal linear complexity is polynomial.

6.1 Future Directions

- In the case of the word-oriented shrinking generator and self-shrinking generator, only the function $f(x)$ is even or odd is used. In future work, we plan to incorporate other

functions and study the randomness properties of the generated bitstream especially periodicity and linear complexity.

- Study on nonlinear Xorshift random number generators.
- Parallel computation using jump LFG.
- More study and analysis on generalized WFSRs.

WFSRs are useful as they take advantage of word-based processors and significantly improved the throughput of sequence generation. In this thesis, we have given a general expression for WFSRs. Table 6.1 shows some well-known FSRs are shown as the special case of WFSR by putting different restrictions in Eq. (2.1). Column 1 of Table 6.1 tells the name of the FSR. Columns 2 and 3 give the value of the degree of feedback function F and the word size m, respectively. Column 4 tells which operation is used for $\sum$ in Eq. (2.1) and column 5 says, the multiplication operation used when $deg(F) > 1$.

FSR	$deg(F)$	m	$\sum$	$\prod$
LFSR	1	1	$\oplus$	
NLFSR	> 1	1	$\oplus$	&
LFG	1	> 1	$+$	
MRMM	1	> 1	$\oplus$	

Table 6.1: Special cases of different WFSRs

Appendix A

S box

S[256] = [0x9A, 0x85, 0xAF, 0xBC, 0x00, 0xAB, 0x89, 0xC1, 0x6D, 0x7F, 0xB8, 0x1C, 0x13, 0x30, 0x37, 0xA6, 0xDB, 0x71, 0xD2, 0x54, 0x31, 0x32, 0xD9, 0xFC, 0xE4, 0x99, 0xCF, 0x15, 0xF6, 0x34, 0x84, 0xBF, 0x3A, 0xAA, 0x6F, 0xB3, 0xBE, 0xEE, 0xFD, 0xD3, 0x4F, 0x23, 0xCE, 0x9B, 0x3B, 0xCC, 0xA9, 0x04, 0xA5, 0xF2, 0x1B, 0xC3, 0x8A, 0xF3, 0x5D, 0x16, 0xAC, 0x47, 0x77, 0x11, 0x2F, 0x1E, 0x08, 0x2E, 0xCA, 0x09, 0x38, 0x40, 0xDA, 0xE7, 0xB4, 0xE6, 0x88, 0xED, 0xA0, 0xD1, 0x29, 0xE3, 0x3E, 0xC5, 0x70, 0x58, 0x46, 0x91, 0x0A, 0xAD, 0x1A, 0xA1, 0x4A, 0xCD, 0x0B, 0x9F, 0xB9, 0x20, 0xD5, 0x42, 0x05, 0xF1, 0x14, 0x75, 0xE0, 0xAE, 0x36, 0xC6, 0x92, 0x8E, 0x94, 0x26, 0x79, 0xB5, 0xDC, 0xB6, 0x81, 0x3C, 0x74, 0xC4, 0xD6, 0x19, 0xE5, 0xA8, 0xFA, 0x12, 0xD8, 0xDE, 0x69, 0x49, 0x7A, 0x44, 0xB2, 0xD0, 0xE9, 0xF0, 0xCB, 0x56, 0x4D, 0x07, 0xBA, 0x97, 0x24, 0xE2, 0x8D, 0x59, 0xA4, 0xA3, 0x93, 0x86, 0x21, 0xF8, 0x3D, 0x83, 0x3F, 0x28, 0x43, 0xA7, 0x9C, 0x98, 0x72, 0x7D, 0x8F, 0xC8, 0xEF, 0x2B, 0x41, 0x90, 0xF4, 0xEC, 0x1F, 0xF9, 0x68, 0x51, 0x01, 0x0C, 0x60, 0x62, 0xBD, 0x5A, 0x48, 0x52, 0x8B, 0x67, 0xE8, 0xFE, 0xA2, 0x53, 0xB1, 0xC2, 0xC7, 0x96, 0x87, 0x22, 0x4C, 0x45, 0x55, 0x4B, 0x95, 0xEB, 0xDD, 0xFF, 0xD7, 0x5E, 0x1D, 0x9D, 0x80, 0x6A, 0x76, 0xD4, 0x0E, 0x4E, 0x9E, 0x50, 0x2A, 0x82, 0x27, 0x7C, 0x7E, 0x61, 0x6B, 0x6E, 0x0D, 0xB7, 0x03, 0x5C, 0x8C, 0xFB, 0x17, 0x2D, 0x73, 0xC0, 0x57, 0x18, 0x0F, 0xDF, 0x06, 0x33, 0xE1, 0x7B, 0x25, 0x66, 0x39, 0x78, 0x10, 0x6C, 0x64, 0xF7, 0xBB, 0xB0, 0x02, 0x35, 0x63, 0x5F, 0xC9, 0x2C, 0xEA, 0x65, 0xF5, 0x5B]

References

[1] L. Bassham, A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, N. Heckert and J. Dray, *A statistical test suite for random and pseudo random number generators for cryptographic applications*, Special Publication (NIST SP) - 800-22 Rev 1a, 2010, [Online], available: https://csrc.nist.gov/publications/detail/sp/800-22/rev-1a/final.

[2] M. Ben-Or, *Probabilistic algorithms in finite fields*. In Proc. 22nd IEEE Symp. Foundations of Computer Science, 1981, pp. 394-398.

[3] T. Beth and F. Piper, *The stop-and-go-generator*, in Proc. of EUROCRYPT 84, Springer, LNCS, vol. 209, 1985, pp. 88-92.

[4] E. Biham, R. J. Anderson and L. R. Knudsen. *Serpent: A New Block Cipher Proposal*. In Fast Software Encryption, FSE 1998, number 1372 in LNCS, Springer-Verlag, 1998, pp. 222-238.

[5] S. K. Bishoi, H. K. Haran and S. U. Hasan, *A note on the multiple-recursive matrix method for generating pseudorandom vectors*, Discrete Applied Mathematics, vol. 222, 2017, pp. 67-75.

[6] S. K. Bishoi and V. Matyas, *Investigating results and performance of search and construction algorithms for word-based LFSRs, σ-LFSRs*, Discrete Applied Mathematics, vol. 243, 2018, pp. 90-98.

[7] S. K. Bishoi and S. N. Maharana, *Xorshift RNG from primitive polynomial*, Theoretical and Applied Informatics, vol. 29, 2017, pp. 1-13.

[8] S. K. Bishoi, K. Senapati and B. R. Shankar, *Shrinking generators based on σ-LFSRs*, Discrete Applied Mathematics, vol. 285, 2020, pp. 493-500.

[9] S. K. Bishoi, K. Senapati and B. R. Shankar, *Bitstream generators using Multiple-Recursive Matrix Methods*, 2022, doi: https://doi.org/10.21203/rs.3.rs-2105578/v1.

[10] R. P. Brent, *On The Periods of Generalized Fibonacci Recurrences*, Mathematics of Computation, Vol. 63, Number 207, 1994, pp. 389-401.

[11] J. Brillhart, D. H. Lehmer, J. L. Selfridge, B. Tuckerman and S. S. Wagstaf, Jr. *Factorization of* $b^n \pm 1, b = 2, 3, 5, 6, 7, 10, 11, 12$ *up to high powers*, 2nd ed., Contemp. Math., Vol. 22, Amer. Math. Soc., Providence, RI, 1988.

[12] D. Cantor and E. Kaltofen, *On fast multiplication of polynomials over arbitrary algebras*, Acta. Inform. 28, 1991, pp. 693-701.

[13] A. H. Chan, R. A. Games and E. L. Key, *On the Complexities of de Bruijn Sequences**, Journal of Combinatorial Theory, Series A, 33, 1982, pp. 233-246.

[14] M. K. Chetry, S. K. Bishoi and V. Matyas, *When lagged fibonacci generators jump*, Discrete Applied Mathematics, vol. 267, 2019, pp. 64-72.

[15] D. Coppersmith, H. Krawczyk and Y. Mansour, *The Shrinking Generator*, Advances in Cryptology - CRYPTO '93, LNCS 773, 1994, pp. 22-39.

[16] *Cryptographic competitions eSTREAM: the ECRYPT Stream Cipher Project*, http://Competitions.cr.yp.to/estream.html.

[17] P. Ekdahl and T. Johansson. *A New Version of the Stream Cipher SNOW*. In Selected Areas in Cryptography, SAC 2002, number 2595 in LNCS, Springer-Verlag, 2003, pp. 47-61.

[18] N. Ferguson, D. Whiting, B. Schneier, J. Kelsey, S. Lucks and T. Kohno. *Helix: Fast Encryption and Authentication in a Single Cryptographic Primitive*. In Fast Software Encryption, FSE 2003, number 2887 in LNCS, 2003, pp. 330–346.

[19] H. Fredricksen, *A survey of full length nonlinear shift register cycle algorithms*, SIAM Review, vol. 24, no. 2, 1982, pp. 195-221.

[20] S. Gao and D. Panario, *Tests and construction of Irreducible Polynomials over Finite Fields*, Foundations of Computational Mathematics, F. Cucker and M. Shub (EDs.), Springer, 1997, pp. 346-361.

[21] S. Gao, J. Von zur Gathen, D. Panario and V. Shoup, *Algorithms for Exponentiation in Finite Fields*, J. Symb. Comput. 29, 2000, pp. 879-889.

[22] S. R. Ghorpade, S. U. Hasan and M. Kumari, *Primitive polynomials, Singer cycles, and word oriented linear feedback shift registers*, Design Codes and Cryptography, Vol. 58, No. 2, 2011, pp. 123-134.

[23] J. D. Golić, *On the linear complexity of functions of periodic GF(q) sequences*, IEEE Transactions on Information Theory, Vol. 35, No. 1, 1989, pp. 69-75.

[24] R. Göttfert and H. Niederreiter, *On the minimal polynomial of the product of linear recurring sequences*, Finite Fields and Their Applications, 1(2), 1995, pp. 204-218.

[25] D. H. Green and K. R. Dimond, *Nonlinear product-feedback shift registers*, Proc. of the Institution of Electrical Engineers, Vol.117, No.4, 1970, pp. 681-686.

[26] S. W. Golomb, *Shift Register Sequences*, Revised Edition, Aegean Park Press, Laguna Hills, 1982

[27] C. G. Gunther, *Alternating step generators controlled by de Bruijn sequences*, Advances in Cryptology-EUROCRYPT '87 (LNCS 304), 1988, pp. 5-14.

[28] S. U. Hasan, D. Panario and Q. Wang, *Nonlinear vectorial primitive recursive sequences*, Cryptography and Communications, 10, 2018, pp. 1075-1090.

[29] H.G. Hu, G. Gong *Periods on Two Kinds of Nonlinear Feedback Shift Registers with Time Varying Feedback Functions*, International Journal of Foundations of Computer Science, Vol. 22, No. 06, 2011, pp. 1317-1329.

[30] A. A. Kanso, *The Alternating Step(r, s) Generator*, SECI02, Tunis, Sep. 2002.

[31] E. L. Key, *An analysis of the structure and complexity of nonlinear binary sequence generators*, IEEE Trans. Inform. Theory, vol.IT-22, 1976, pp. 732-736.

[32] R. Lidl and H. Niederreiter, *Finite Fields*, Encyclopedia of Mathematics and its Applications 20, Cambridge University Press, Cambridge, 1997.

[33] G. Marsaglia, *Xorshift RNGs*. Journal of Statistical Software, 8, 14, 2003, pp. 1-9. http://www.jstatsoft.org.

[34] G. Marsaglia and L. Tsay, *Matrices and the structure of Random Number Sequences*. Linear Algebra and its Applications 67, 1985, pp. 147-156.

[35] J. L. Massey, *Shift Register Synthesis and BCH Decoding*, IEEE Transactions on Information Theory, Vol. IT-15, 1969, pp. 122-127.

[36] W. Meier and O. Staffelbach, *The self-shrinking generator*. Proc. of Advances in Cryptology, EuroCrypt 1994, Springer-Verlag, 1998, pp. 205-214. .

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

[38] National Institute of Standards and Technology, 2001. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf.

[39] National Institute of Standards and Technology, 1999. https://csrc.nist.gov/CSRC/media/Publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf.

[40] H. Niederreiter, *The multiple-recursive matrix method for pseudorandom number generation*, Finite Fields Appl. 1, 1995, pp. 3-30.

[41] H. Niederreiter, *Pseudorandom vector generation by the multiple-recursive matrix method*, Math. Comp. 64, 1995, pp. 279-294.

[42] H. Niederreiter, *Improved bound in the multiple-recursive matrix method for pseudorandom number and vector generation*, Finite Fields Appl. 2, 1996, pp. 225-240.

[43] B. Preneel, *Introduction to the Proc. of the Second Workshop on Fast Software Encryption*, in: Lecture Notes in Comput. Sci., vol. 1008, Springer, Berlin, 1995, pp. 1-5.

[44] R. L. Rivest, M. J. B. Robshaw, and Y. L. Yin, *RC6 as the AES*. In AES Candidate Conference, 2000, pp. 337-342.

[45] R. A. Rueppel and O. J. Staffelbach, *Products of linear recurring sequences with maximum complexity*, IEEE Transactions on Information Theory, 33, 1987, pp. 124-131.

[46] B. Schneier, "*Applied Cryptography*", John Willey & Sons, New York, 1996.

[47] B. Schneier, J. Kelsey, D. Whiting, D. Wagner and N. Ferguson. *Comments on Twofish as an AES Candidate*. In AES Candidate Conference 2000, pp. 355-356.

[48] C. E. Shannon, *Communication Theory of Secrecy Systems*, Bell System Technical Journal, 28(4), 1949, pp. 656-715.

[49] T. Siegenthaler, *Decrypting a class of stream ciphers using ciphertext only*, IEEE Transactions on Computers, 34, 1985, pp. 81-85.

[50] D. R. Stinson, *Cryptography Theory and Practice*. Chapman & Hall/CRC, Third Edition, 2006.

[51] B. Tsaban and U. Vishne, *Efficient feedback shift registers with maximal period*, Finite Fields Appl. 8, 2002, pp. 256-267.

[52] S. S. Wagstaf, Jr., *Update 2.6 to the Second Edition of Factorization of* $b^n \pm 1$, 1993.

[53] H. Wu, *ACORN: a lightweight authenticated cipher (v3)*. Candidate for the CAESAR Competition (2016), 2020.

[54] G. Zeng, W. Han and K. He, *Word-oriented feedback shift register: σ-LFSR*, 2007, [Online], available: http://eprint.iacr.org/2007/114.

List of research articles (published/ communicated) arising out of the thesis

Peer Reviewed International Journals

- Bishoi, S.K., Senapati, K., Shankar, B.R.: *Shrinking generators based on σ-LFSRs*, Discrete Applied Mathematics, vol. 285, pp. 493-500, 2020.

Communicated research papers

- Bishoi, S.K., Senapati, K., Shankar, B.R.: *Bitstream generators using Multiple-Recursive Matrix Methods*, Sept 2022, doi: https://doi.org/10.21203/rs.3.rs-2105578/v1
- S. K. Bishoi, K. Senapati and B. R. Shankar, *Generalized word-oriented feedback shift registers*, https://eprint.iacr.org/2023/949/.

Susil Kumar Bishoi, MSc, MTech

skbishoi@gmail.com skbishoi.cair@gov.in
Contact No.: 8431580194

Employment History

2005 – Till date **Scientist.** Centre for Artificial Intelligence and Robotics, Defence Research and Development Organization, C V Raman Nagar, Bangalore.

Education

2000 – 2002 **M.Sc. Mathematics, Utkal University, Bhubaneswar**

2008 – 2010 **M.Tech. Computational Science, SERC, Indian Institute of Science, Bangalore**

2015 – 2016 **M.S. Information Security, Masaryk University, Czech Republic**

Research Publications

- Bishoi, S.K., Haran, H.K., Hasan, S.U.: *A note on the multiple-recursive matrix method for generating pseudorandom vectors*, Discrete Applied Mathematics, vol. 222, pp. 67-75, 2017.
- Bishoi, S.K., Maharana, S.N.: *Xorshift RNG from primitive polynomial*, Theoretical and Applied Informatics, vol. 29, pp. 1-13, 2017.
- Bishoi, S.K., Matyas, V., *Investigating results and performance of search and construction algorithms for word-based LFSRs, σ-LFSRs*, Discrete Applied Mathematics, vol. 243, pp. 90-98, 2018.
- M.K. Chetry, S.K. Bishoi and V. Matyas, *When lagged fibonacci generators jump*, Discrete Applied Mathematics, vol. 267, pp. 64-72, 2019.
- S. K. Bishoi, K. Senapati and B. R. Shankar, *Shrinking generators based on σ-LFSRs*, Discrete Applied Mathematics, vol. 285, pp. 493-500, 2020.

Communicated research papers

- Bishoi, S.K., Senapati, K., Shankar, B.R.: *Bitstream generators using Multiple-Recursive Matrix Methods*, Sept 2022, doi: https://doi.org/10.21203/rs.3.rs-2105578/v1
- S. K. Bishoi, K. Senapati and B. R. Shankar, *Generalized word-oriented feedback shift registers*

Skills

Languages Strong reading, writing and speaking competencies for English, Hindi, Odiya, Benglali.

Coding C, Python, R, LaTeX, Matlab, Sage