On Cryptographic Applications of Word Oriented LFSR

Thesis Submitted for

DOCTOR OF PHILOSOPHY

by

Subrata Nandi

Roll No: 156101007

under the supervision of

Dr. Pinaki Mitra and Dr. Srinivasan Krishnaswamy

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

GUWAHATI, ASSAM - 781039

Dedication

I dedicate this thesis to

Didima(Late Srimati Sandhaya Rani Saha) & Asis Jethu(Late Sri Ashis Dutta Choudhury)

For their unconditional love, blessings and constant inspiration

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Guwahati - 781039 Assam India

Certificate

This is to certify that this thesis entitled, **On Cryptographic Applications of Word Oriented LFSR**, being submitted by **Subrata Nandi**, to the Department of Computer Science and Engineering, **IITG**, for partial fulfilment of the award of the degree of **P.hD** , is a bonafide work he carried out under my supervision and guidance. The thesis, in my opinion, is worthy of consideration for the award of the degree of **Doctor of Philosophy** in accordance with the regulation of the institute. I know it has not been submitted elsewhere for the degree award.

....
Dr. Pinaki Mitra
Associate Professor
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Email: pinaki@iitg.ac.in
Thesis Supervisor

....
Dr. Srinivasan Krishnaswamy
Associate Professor
Department of Electronics and Electrical Engineering
Indian Institute of Technology Guwahati
Email: srinikris@iitg.ac.in
Thesis Supervisor

Date:
Place: Guwahati

Declaration

I certify that:

- The work in this thesis is original and has been done by myself and under the general supervision of my supervisor.
- The work reported herein has not been submitted to any other Institute for any degree or diploma.
- Whenever I have used materials (concepts, ideas, text, expressions, data, graphs, diagrams, theoretical analysis, results, etc.) from other sources, I have given due credit by citing them in the text of the thesis and giving their details in the references. Elaborate sentences used verbatim from published work have been identified and quoted.
- I also affirm that no part of this thesis can be considered plagiarism to the best of my knowledge and understanding. I take complete responsibility if any complaint arises.
- I am fully aware that my thesis supervisor is not able to check for any possible plagiarism within this submitted work.

Date:
Place: Guwahati

Subrata Nandi

Acknowledgement

Firstly, I want to express my gratitude to my mentors, *Dr. Pinaki Mitra*(Pinaki Sir) and *Dr. Srinivasan Krishnaswamy*(Srini Sir). I owe a great deal to their unwavering kindness, concern, competent advice, experience, and insight. Whenever I was struggling with any problem during my PhD studies, they were there for me every step of the way. I want to mention my mentor, Dr. Srinivasan Krishnaswamy(Srini Sir), who has worked closely with this thesis. Without his help, I could not start working on this project and ended up doing the same. His PhD thesis was quite helpful in pointing me toward solutions for my PhD challenges. My mentor, Pinaki Sir, ability to pose essential questions prompted me to consider the issue carefully. Both of my mentors contributed significantly to my personal development.

I want to acknowledge the members of my Doctoral Committee (DC),*Dr. Sajith Gopalan*,*Dr Vinay Wagh*, *Dr. Deepanjan Kesh* for their essential suggestions to add quality to my work. Thank you, Dr. Vinay Wagh, for allowing me to sit through his Modern Algebra and Galois Theory classes. I also express my gratitude to *Dr. Gaurav Trivedi*, faculty of the EEE department, for supporting me with server computer access for solving a computationally expensive problem.

Further, I express my gratitude to the heads of the Department of Computer Science and Engineering, *Dr Jatindra Kumar Deka*, and *Dr Tamarapalli Venkatesh* during my PhD work for giving me access to departmental facilities and resources. I would also like to thank all the faculty in the CSE department. All the department staff, Nanu Da, Bhrigu Da, Raktajit Da, Naba Da, Gouri and Manajit Da, were amiable when I asked for help.

To talk about my friends at IIT Guwahati, I would like to mention Shounak Chakravorty, who helped me a lot in my initial days at IITG and introduced me to others. A special thanks to Suman, who introduced me to Srini Sir and enabled me to initiate my research. Besides, Arunangshu, Abhijit and Brijesh are the three persons with whom I spend most of my valuable time at IIT Guwahati. As hostel mates in Dibang, we enjoyed countless hours chatting over issues and watching movies, sports, food, etc. I will cherish a healthy friendship with them all my life. Partha Da is one of my friends with whom I spent quality time discussing many ideas for our PhD work. Besides, Anasua, Suman, Atanu, Madhurima, Sisir Da, and Pranab Da were special batchmates and friends at IITG. Further, I would like to thank my Tabla Teacher, *Mr. Dhritigobinda Dutta*, for sharing exciting lessons with me, which made my IITG days very interesting. I want to mention Suman, Partha Da, Uddipana, Abhijeet, Kasturi, and Deepak for being lovely research mates.

I want to express my heartiest respect to my Boromama, Chotomama, Didima, Boromami, Chotomami, Ashish Jethu, and my parents-in-law for their unconditional support throughout my PhD days. I want to thank my cousins Roni and Toni for being supportive. I express special thanks to my wife, *Arpita Mukherjee*, for being an immense source of motivation and strength throughout my journey. Lastly, I owe my deepest regards to my parents, *Mrs. Sima Nandi* and *Mr. Baidyanath Nandi.* This thesis would not have been possible without their support and faith in me. This acknowledgement would never be complete without a special mention of my idol, *Sri Anukulchandra Chakravarty*, who has inspired me deeply to improve my life since my childhood.

Abstract

This thesis deals with the generation of key dependent feedback configurations for LFSRs in Word based Stream Ciphers. In particular, SNOW 2.0 and SNOW 3G are considered. This work looks at methods of replacing the word based LFSR in these ciphers with σ-LFSRs whose feedback configurations depends on the secret key. Further, the security implications of these changes are analysed in detail.

The first contribution of this thesis is the design of the $\sigma-$KDFC scheme. In this scheme, the word based LFSR in a stream cipher is replaced with the σ-LFSR whose state transition matrix has the same characteristic polynomial as the original LFSR. This scheme uses the fact that, there exists a large number $\sigma-$LFSR feedback configurations corresponding to each primitive characteristic polynomials. We demonstrate that the degree of the entries of feedback matrices as polynomials in variables derived from the key is significantly high. This significantly increases the resistance to Algebraic Attacks. Further, this scheme resists those Fast Correlation Attacks where in the feedback equation is taken over an extension field of F_2. However, this scheme gives no added advantage against Fast Correlation Attacks that employ Linear Recurring Relation over $GF(2)$. This makes the scheme vulnerable to attacks like the one given in [1]. This leads to the next contribution of the thesis.

The next task taken up in this work is the design of random z-primitive σ-LFSR configurations. These configurations are random in both in terms of feedback gain and the characteristic polynomial of the state transition matrix. In this scheme, the feedback configuration is hidden using a symmetric matrix that is derived from the key. This symmetric matrix and the feedback gain are used to derive a parameter that is made public. The receiver can retrive the feedback configuration from the public parameter using the secret key. In addition to the attacks that are resisted by σ-KDFC scheme, this scheme also resists attacks like the one given in [1] which employ the LRR over $GF(2)$.

Finally, we discuss methods of doing away with the public parameter in the above mentioned scheme. This avoids the communication cost incurred in sharing the public parameter.

The thesis concludes by summarising the major contributions and discussing scope for future work.

Contents

List of Figures

Notation

Symbol	**Meaning**
$\mathbb{F}_{2^n}$	Finite field of cardinality 2^n
$\mathbb{F}_2^n$	n-dimensional vector space over $\mathbb{F}_2$
$M[i,:]$	The i^{th} row of a matrix M
$M[:,j]$	j^{th} column of a matrix M
$M[i,j]$	(i,j)-th entry of the matrix M
$\mu(M[i,j])$	The minor of $M[i,j]$
$\oplus$	XOR
$+$	Addition or XOR
$M^{n\times n}$	Matrix M with n rows and n columns
M^{-1}	Inverse of a matrix M.
$GF(p)$	Galois Field with elements $\in \{0,1,\cdots,p-1\}$
$M_m(F_2)$	Matrix Ring over F_2
δ	Circular Left shift Operator of a Sequence
$\mathbb{N}$	Natural Number
$\mathbb{R}$	Real Number
$P(x)$	Probability of a random varibale x
v^T	Transpose of a vector v.
$v_1.v_2$	Dot product between two vectors v_1 and v_2 over F_2
$\boxplus$	Addition modulo 2^n
e_1^i	row vector with $i-$th entry is 1, and the other entries are 0.
A^T	Transpose of a matrix A.
GCD(a,b)	Greatest common divisors of a and b
$[v_1, v_2, \ldots, v_n]$	A matrix with columns $v_1, v_2, \ldots, v_n$
$[v_1; v_2; \ldots; v_n]$	A matrix with rows $v_1, v_2, \ldots, v_n$
&&	Bitwise AND Operator
$\vert\vert$	Concatenation Operator

Table 1: Symbol and their meaning

Chapter 1

Introduction

1.1 Word Based Linear Feedback Shift Register

Cryptography is the art of securing communication in the presence of third parties. It involves the use of mathematical algorithms that convert plaintexts (original messages) into ciphertexts (encrypted messages), that are unreadable to anyone without the valid key. It has been used for centuries to protect sensitive information, including military secrets, diplomatic messages, and financial transactions. With the rise of digital communication, cryptography has become increasingly important in protecting online information. It is used in various applications such as secure messaging, online banking, e-commerce, and data storage. It ensures the confidentiality, integrity, and authenticity of digital information. Hash functions and Digital Signatures ensure integrity and authenticity. Any cryptographic scheme has many associated algorithms—for example, Encryption, Decryption, MAC and Digital Signatures. Further, depending upon the nature of key sharing, a cryptographic scheme can be symmetric or asymmetric. A cryptographic scheme that acts on blocks of plaintext data to generate blocks of ciphertexts of the same size is known as a Block cipher. Examples of block ciphers include BlowFish[2], Rijndael[3, 4],Present[5],Simon and SPECK[6], Midori[7] and so on . On the other hand, a stream cipher encrypts a stream of data one element at a time. In a stream cipher, the ciphertext is the XOR sum of a pseudorandom keystream and the plaintext. LFSRs are often used to build pseudorandom keystream generators in stream cipher as they are extremely easy to implement both in hardware and software. Grain[8], Trivium[9], Mickey-2[10], Fruit-80[11], Espresso[12] are a few examples of LFSR based stream cipher. Word-based LFSRs were introduced to enable efficient software implementation on computers with modern word-based processors. Various stream ciphers with word based LFSRs include RC4[13], Sober[14], SNOW 1.0[15], SNOW 2.0[16],Turing[17], SNOW 3G[18], ZUC[19], SNOW V[20], Sosemanuk [21], and SNOW Vi [22].

A σ-LFSR, introduced in [23] is a special word-based LFSR that implements the multiple recursive matrix method for pseudorandom vector generation given in [24]. Here, feedback gains are taken as matrices. A heuristic algorithm to generate a $\sigma-$LFSR with three nonzero gain matrices is proposed in [25]. [26] deals with a special σ-LFSR configuration known where each gain matrix is designed as a scaler multiple of each other. In [23], it is conjectured that the number of primitive $\sigma-$LFSR configurations with b, m-input $m-$output delay blocks is $(\frac{|GL(m,\mathbb{F}_2)|}{2^m-1} \times \frac{\phi(2^{mb}-1)}{mb} \times 2^{m(m-1)(b-1)})$. For the case $m = 1$ and $m = n$, this conjecture is proved in [27]. The proof for the general case is given in [28]. Further, [28] gives a constructive method for enumerating $\sigma-$LFSR configurations. Krishnaswamy et al. The article[29] introduces a special class of σ-LFSR called z-primitive σ-LFSR. This paper also gives the cardinality of the set of z-primitive $\sigma-$LFSRs$(\frac{|GL(m,\mathbb{F}_2)|}{2^m-1})$.

The knowledge of the feedback function plays a critical role in most attacks on LSFR-based stream ciphers. These include Algebraic attacks, Correlation attacks, Distinguishing attacks, Guess and determine attacks, Cache timing attacks etc.([30, 31, 32, 33, 34, 35, 36, 1]). Therefore, hiding the feedback function of the LFSR could potentially increase the security of such schemes. One way of doing this is by using dynamic feedback control. This approach is used in stream ciphers such as K2 ([37]) and A5/1. This converts the deterministic linear recurrence into a probabilistic recurrence. However, key recovery attacks on K2 and A5/1 have been reported in the literature ([38, 39]). In this thesis, we have suggested methods of hiding the feedback configuration of σ-LFSR.

1.2 Contribution

In this thesis, we have suggested methods of hiding the feedback configuration of σ-LFSRs to resist various known plaintext attacks.

1 The first method, called the σ-KDFC (Key Dependent Feedback Configuration), uses the algorithm given in [40, 28]. This method is a key-dependent $\sigma-$LFSR configuration whose characteristic polynomial is a publicly known primitive polynomial. The feedback gains obtained in this method are non-linear functions of the secret key. Further, the number of iterations in this algorithm can be adjusted depending on the available computing power. As an example, we study the interconnection of the σ-KDFC scheme with the finite state machine (FSM) of SNOW-2.0. The randomness of the generated keystream is verified using empirical tests. Further, the security of the scheme against various attacks is analysed.

2 The second scheme proposed in this thesis addresses the vulnerability in σ-KDFC caused by the publicly known characteristic polynomial. This scheme generates a random z-primitive σLFSR configuration masked using the secret key. This masked configuration is declared as a public parameter. The receiver regenerates the feedback configuration

from the public parameter using the secret key. The ability of this scheme to resist attacks that σ-KDFC is vulnerable to is demonstrated. Finally, we have suggested ways of doing away with the public parameter to avoid the cost of sharing the same.

1.3 Thesis Organization

The rest of the thesis is organised as follows

1 Chapter 2 deals with the background information needed to understand the thesis. It covers some basic algebraic definitions and describes several word-based cryptographic schemes. Further, this chapter briefly discusses various attacks on stream ciphers and introduces σ-LFSRs.

2 Chapter 3 introduces, describes and analyses the σ-KDFC scheme. It proposes a method for generating key-dependent feedback configurations (KDFC) for σ-LFSRs. It analyses the security impact of replacing the existing word-based LFSR in SNOW 2.0 with a key-dependent σ-LFSR.

3 Chapter 4 proposes a method of generating a random $z-$primitive $\sigma-$LFSR configuration. Further, it describes a method of concealing this configuration in a public parameter matrix. This chapter also contains a security analysis of the proposed scheme when it is used in SNOW 2.0 and SNOW 3G.

4 Chapter 5 suggests methods of removing the public parameter in the scheme described in Chapter 4.

5 Chapter 6 concludes the thesis and suggests areas of future research.

Chapter 2

Preliminaries

This chapter discusses some basic concepts related to this work. The first part of the chapter deals with algebraic preliminaries. The rest of the chapter discusses word-based stream ciphers and attacks on them.

2.1 Algebraic Preliminaries

Given below are some algebraic preliminaries. The definitions and results given in this section are taken from [41, 42, 43].

Definition: 1 ***Binary Operation:*** *A binary operation on a set B is a map $f : B \times B \to B$ which assigns an element of B to each ordered pair of elements in B.*

Definition: 2 ***Group:*** *A nonempty set of elements G constitutes a group under a binary operation '.', if*

1. *$p, q \in G \implies p.q \in G$.(Closure Property)*
2. *$p, q, r \in G \implies p.(q.r) = (p.q).r$(Associative Property)*
3. *There exists an element i such that $p.i = i.p = p \ \forall p \in G$. The element i is called the identity element of the group G.*
4. *For each $p \in G$, there exists an element p^{-1} such that $p.p^{-1} = p^{-1}.p = i$. The element p^{-1} is called the inverse of p.*

A group is denoted by two tuples $(G, .)$ where G refers to the set and . refers to the operation. A group is abelian or commutative if the operation . is commutative i.e. $\forall a, b \in G$, $a.b = b.a$. Given below are a few examples of groups.

a) The set of integers $\mathbb{Z}$,+ is a group under addition.

b) $\{GL(n,\mathbb{R}),*\}$ is the group of all real nonsingular $n \times n$ matrices under matrix multiplication.

c) The set of permutations of a finite set constitutes a group under composition.

Observe that the first example is Abelian while the other two are not.

Definition: 3 ***Subgroup:*** *A nonempty subset K of the group G is a subgroup of G if the following hold,*

1. $\forall p, q \in K \implies p.q \in K$.
2. $p \in K$ *results that* $p^{-1} \in K$.

$SL(n,\mathbb{R})$, the group of real $n \times n$ matrices with determinant 1, is a subgroup of $GL(n,\mathbb{R})$ under matrix multiplication.

Definition: 4 ***Order of a Group:*** *The order of a Group is the number of elements in that Group. The order of G is denoted by $o(G)$.*

Definition: 5 ***Order of an element:*** *Given a group $(G,.)$ the order of $p \in G$, denoted by $o(p)$, is the least positive integer m such that $\underbrace{p.p.\cdots.p}_{m-times} = i$, where i is the identity element of the group G.*

Definition: 6 ***Cyclic Group:*** *A group $(G,.)$ is a cyclic group if $\exists p \in G$ such that $G = \{p^i : i \in \mathbb{Z}\}$, where $p^i = \underbrace{p.p.\cdots.p}_{i-times}$. Here, p is called the generator of the cyclic group.*

A group generated by an element p is denoted by $< p >$.

Example: 1 *The set of integers modulo n, denoted by $\mathbb{Z}_n$, is a cyclic group under addition with generator* 1.

Theorem: 1 *Let $(G,.)$ be a finite cyclic group of order $n > 1$ generated by p. Then p^x is also a generator of the group if and only if x is less than and co-prime to n.*

Corollary: 2 *[41] The total number of generators for a finite cyclic group of order n is $\phi(n)$.*

Definition: 7 ***Ring:*** *A non-empty set R equipped with two operations $+$ and $.$ is said to be a ring $(R,+,.)$ if the following set of axioms is satisfied.*

- *R is an abelian group with respect to $+$.*
- *For all $p, q, r \in R$, the following are satisfied*

  - $(p.q).r = p.(q.r)$ *(Associative)*
  - $(q + r).p = q.p + r.p$ *(Distributive)*

The set of integers $\mathbb{Z}$ is a ring with addition and multiplication. For convenience, the $+$ and $.$ operations are henceforth referred to as addition and multiplication. A ring that contains a multiplicative identity is called a ring with identity. A ring where multiplication is commutative is called a commutative ring.

Definition: 8 *A subset S of a ring R is a subring of R if it is closed under the operations $+$ and $.$ and forms a ring under the same operations.*

Definition: 9 *An ideal J is a subring of a ring R, which satisfies the following property.*

- *$\forall p \in J$ and $r \in R$ we have $pr \in J$ and $rp \in J$.*

An ideal J partitions a ring R into disjoint cosets called residue classes modulo J. Two elements $a, b \in R$ belong to the same residue class modulo J if $a - b \in J$. The residue class containing the element a is denoted by $[a]$. The set of residue classes constitutes a ring where addition and multiplication are inherited from R and are defined as follows.

$$[a] + [b] = [a + b] \tag{2.1}$$

$$[a].[b] = [a.b] \tag{2.2}$$

An ideal J in a commutative ring R is said to be generated by set of elements $p_1, p_2, \ldots p_k \in J$, denoted by $J =< p_1, p_2, p_3, \ldots, p_k >$, if $J := \{a_1p_1 + a_2p_2 + \cdots + a_kp_k | a_1, a_2, \ldots, a_k \in R\}$. An ideal generated by a single element is called a principal ideal. A commutative ring where all ideals are principal ideals is called a principal ideal domain.

Definition: 10 ***Field:*** *A field is a set $\mathbb{F}$ with two binary operations $+$ and $\times$, denoted by $(\mathbb{F}, +, .)$ satisfying the following properties:*

- *$(\mathbb{F}, +)$ is an abelian group with additive identity element 0.*
- *$(\mathbb{F} - \{0\}, .)$ is an abelian group with multiplicative identity $\mathbf{1}$.*
- *The distributive law $p(q + r) = pq + pr$ holds $\forall\ p, q, r \in \mathbb{F}$.*

Example: 2 *Given a prime p, $\{\mathbb{Z}_p, +, .\}$ is a field where $+$ is addition modulo p and $.$ is multiplication modulo p.*

Definition: 11 ***Vector Space:*** *A nonempty set V is said to be a vector space over a field $\mathbb{F}$ if;*

1. *The following operations are associated with V.*
   - *Addition : $V \times V \longrightarrow V$, The addition of $v, w \in V$ is denoted by $v + w$*
   - *Scalar Multiplication : $\mathbb{F} \times V \longrightarrow V$. The scalar multiplication of $a \in \mathbb{F}$ and $v \in V$ is denoted by cv.*
2. *V satisfies the following properties*
   - *V is a commutative group under addition.*
   - *Scalar multiplication is associative with field multiplication.*

$$(cd)v = c(dv) \forall c, d \in \mathbb{F} \text{ and } v \in V$$

   - *Scalar multiplication of a vector with the multiplicative identity of the group gives the same vector.*
   - *$(c + d)v = cv + dv \ \forall c, d \in \mathbb{F}$ and $v \in V$*
   - *$c(v + w) = cv + cw \ \forall v, w \in V$ and $c \in \mathbb{F}$*

Given a vector space V, a set of linearly independent vectors $(v_1, v_2, \ldots, v_n)$ in V whose linear combinations generate the entire vector space is called a basis of the vector space. Given a basis $B = (v_1, v_2, \ldots, v_n)$ the vector $v = a_1v_1 + a_2v_2 + \cdots + a_nv_n$ is represented by the n-tuple $(a_1, a_2, \ldots, a_n)$. The cardinality of all bases of a vector space is the same and is called the dimension of the vector space.

Definition: 12 *If V_1 and V_2 are finite dimensional vector spaces over a field $\mathbb{F}$, then $T : V_1 \to V_2$ is said to be a linear transformation if $T(x + y) = T(x) + T(y)$ and $T(cx) = cT(x) \forall c \in \mathbb{F}$,*

Given a field $\mathbb{F}$, if T is a linear transformation from $\mathbb{F}^m$ to $\mathbb{F}^n$, then for a given basis, it is represented by a matrix $A \in \mathbb{F}^{m\times n}$, i.e., $T(v) = vA$. Further, if $m = n$ then a vector $v \in \mathbb{F}^m$ is said to be cyclic with respect to the transformation T if the set of vectors $v, T(v), T^2(v), \ldots, T^{m-1}(v)$ spans $\mathbb{F}^m$.

Given a field $\mathbb{F}$, any subset of $\mathbb{F}$ which is also a field under the operations of $\mathbb{F}$ is called a subfield of $\mathbb{F}$. If $\mathbb{K}$ is a subfield of $\mathbb{F}$, then $\mathbb{F}$ is said to be an extension of $\mathbb{K}$. Such an extension is denoted by $\mathbb{F}/\mathbb{K}$. Given an extension $\mathbb{F}/\mathbb{K}$, the field $\mathbb{F}$ is a $\mathbb{K}$-vector space. The dimension of this vector space is called the degree of field extension.

Definition: 13 *The characteristic of a field $\mathbb{F}$, $char(\mathbb{F})$, is the smallest positive integer p such that $\underbrace{1 + 1 + \cdots + 1}_{p-times} = 0$ where 1 is the multiplicative identity of $\mathbb{F}$.*

Theorem: 3 *[43] The characteristic of a field $\mathbb{F}$, $char(\mathbb{F})$, is either 0 or prime p.*

A field with finite cardinality is called a finite field. Such a field always has a prime characteristic (Corollary 1.45 in [42]). Let $\mathbb{F}_q$ be the finite field of order q. The set $\mathbb{F}_q \setminus \{0\}$ constitutes a cyclic multiplicative group denoted by $\mathbb{F}_q^*$ (Theorem 2.8 in [42]). Given any prime p, the set $\mathbb{Z}_p$ is a field where the addition and multiplication operations are inherited from integers. Any other finite field having characteristic p is an extension of $\mathbb{Z}_p$. Further, the cardinality of such a finite field is p^n where n is the degree of field extension (Theorem in [42]).

Definition: 14 *A polynomial $f \in \mathbb{F}[x]$ is said to be irreducible over $\mathbb{F}$ if it cannot be written as a product of two non-constant polynomials in $\mathbb{F}[x]$ i.e. if $m, n \in \mathbb{F}[x]$ and $f = mn$ then either m or n is a constant.*

Theorem: 4 *[43] Let $f(x) \in \mathbb{F}[x]$ be an irreducible polynomial of degree n over the field F and let K be the extension field $\mathbb{F}[x]/ < p(x) >$. Let $\alpha = x \pmod{p(x)} \in K$. Then the elements $\{1, \alpha, \alpha^2, \alpha^3, \cdots, \alpha^{n-1}\}$ are a basis for K as a vector space over F, so the degree of the extension if n, i.e., $[K : \mathbb{F}] = n$. Hence*

$$K = \{a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_{n-1}\alpha^{n-1} | a_0, a_1, \cdots, a_{n-1} \in F\}$$

consists of all polynomials of degrees less than n in α.

Theorem: 5 *[41] The finite field of cardinality q^n, $\mathbb{F}_{q^n}$, is isomorphic to $\mathbb{F}_q^n$.*

Thus, each element of $\mathbb{F}_{q^n}$ can be represented as an element in $\mathbb{F}_q^n$ and vice versa.

Lemma: 6 *[42] Every $a \in \mathbb{F}_{p^n}$ satisfies the equation $a^{p^n} = a$.*

The above lemma implies that the order of any non-zero element of $\mathbb{F}_{p^n}$ in $\mathbb{F}_{p^n}^*$ divides $p^n - 1$

Definition: 15 *Given $q = p^n$ and $\alpha \in \mathbb{F}_{q^m}$, the elements $\alpha, \alpha^q, \alpha^{q^2}, \cdots, \alpha^{q^{m-1}}$ are called conjugates of α with respect to $\mathbb{F}_{q^m}$.*

Definition: 16 *Given $q = p^n$, $\alpha \in \mathbb{F}_q$ is said to be a primitive element of $\mathbb{F}_q$ if it generates the multiplicative group F_q^*.*

If $\alpha \in \mathbb{F}_q$ is a primitive element, then so are all its conjugates to any subfield of $\mathbb{F}_q$. For $q = p^n$, $\mathbb{F}_q^*$ has $\phi(p^n - 1)$ generators.

Definition: 17 ***Primitive Polynomial*** *A monic irreducible polynomial $f \in \mathbb{F}_q[x]$ having degree n is called a primitive polynomial over $\mathbb{F}_q$ if its roots generate $\mathbb{F}_{q^n}^*$.*

Number of primitive polynomials of degree n over $\mathbb{F}_q$ is $\frac{\phi(q^n-1)}{n}$.

Consider a primitive polynomial $f(x) = x^n + c_{n-1}x^{n-1} + c_{n-2}x^{n-2+\cdots+c_0}$. Let $\alpha \in \mathbb{F}_{q^n}$ be the root of $f(x)$. The set of elements $(1, \alpha, \alpha^2, \ldots, \alpha^{n-1})$ is a basis for $\mathbb{F}_{q^n}$ as a vector space over $\mathbb{F}_q$ [42]. For this basis, the following matrix represents the linear transformation corresponding to multiplication by α.

$$M_\alpha = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ c_0 & c_1 & c_2 & \cdots & c_{n-1} \end{bmatrix} \tag{2.3}$$

A finite field of cardinality $q = p^n$ contains every finite field whose cardinality is a factor of q. The following lemma characterizes the primitive elements of these sub-fields.

Lemma: 7 *Given $m, b \in \mathbb{Z}$ and a prime p, if α is a primitive element of $GF(p^{mb})$, α^z is a primitive element of $GF(p^m)$, where $z = \frac{p^{mb}-1}{p^m-1}$.*

Proof: As α is a primitive element of $GF(2^{mb})$, α generates the cyclic multiplicative group $F^*_{2^{mb}}$. Therefore, $\alpha^{2^{mb}-1} = 1$. Further, the cardinality of the cyclic group generated by α^z, i.e. the order of α^z, is $\frac{2^{mb}-1}{GCD(2^{mb}-1,z)}$. Hence, if $z = \frac{2^n-1}{2^m-1}$,

$$order(\alpha^z) = \frac{2^{mb}-1}{GCD(z, 2^{mb}-1)} \tag{2.4}$$

$$= \frac{(2^m-1)(2^{m(b-1)} + 2^{m(b-2)} + \cdots + 1)}{(2^{m(b-1)} + 2^{m(b-2)} + \cdots + 1)} \tag{2.5}$$

$$= 2^m - 1 \tag{2.6}$$

As, the order of the element α^z is $2^m - 1$, it can be said that α^z is the generator of the subgroup $F^*_{2^m}$, i.e. α^z is a primitive element of $GF(2^m)$, where $z = \frac{2^{mb}-1}{2^m-1}$. $\square$

Corollary: 8 *Let $f(x)$ be a primitive polynomial of degree $n = mb$ over F_2 and α be its root. If $z = \frac{2^{mb}-1}{2^m-1}$, a polynomial $g(x)$ having degree m with root α^z is a primitive polynomial. Further, the polynomial $g(x)$ is given as follows:*

$$g(x) = (x + \alpha^z) * (x + \alpha^{2z}) * \cdots * (x + \alpha^{2^{m-1}z}) \pmod{f(\alpha)} \tag{2.7}$$

Proof: The primitiveness of $g(x)$ follows from Lemma 1. Since mb is the smallest exponent of α that equals 1, (α^z, α^{2z}, α^{2^2z},..., $\alpha^{2^{m-1}z}$) are all distinct. Further, if α^z is a root of $g(x)$, so are α^{2z}, α^{2^2z},..., $\alpha^{2^{m-1}z}$. Therefore, $g(x) = (x+\alpha^z)*(x+\alpha^{2z})*\cdots*(x+\alpha^{2^{m-1}z}) \pmod{f(\alpha)}$. $\square$

2.2 Basic Concepts on Cryptography

Cryptography is a data modification technique that ensures that a sender's message transmitted on a public channel can be retrieved only by the intended recipients [44]. Given below are a few related definitions.

Definition: 18 ***Alphabet(A):*** *It is a set of symbols used for encryption. It consists of b bit words,* $A = \{0,1\}^b$*, for some* $b \in \mathbb{Z}_+$.

Let us define the formal notion of the cryptosystem as follows.

Definition: 19 ***Cryptosystem:*** *A cryptosystem or cipher* ***CS*** *is defined by the following five tuple* $\boldsymbol{CS} = \{Pt, Ct, K, Enc, Dec\}$.

- *Pt is the finite set of possible plaintexts.* $Pt = m_1, m_2, \cdots, m_k$, *where* $m_i \in A$.
- *Ct is the finite set of possible ciphertexts.* $Ct = c_1, c_2, \cdots, c_\ell$, *where* $c_i \in A$.
- *K is the keyspace, a finite set of possible keys.* $K = k_1, k_2, \cdots, k_n$, *where* $k_i \in A$
- *The encryption algorithm* $Enc : K \times Pt \mapsto Ct$. *The Key used in encryption is called encryption key* $k_e \in K$.
- *The decryption algorithm* $Dec : K \times Ct \mapsto Pt$. *The Key used in decryption is called decryption key* $k_d \in K$.
- $\forall k_e, k_d \in K$, $Dec(k_d, Enc(k_e, P)) = P$ *holds for all* $P \in Pt$.

The encryption and decryption keys are generated using randomized key generation algorithms with the security parameter as the argument. The encryption algorithm may or may not be randomized, while the decryption algorithm is deterministic. A single secret key is shared between the two parties via a secure channel in a symmetric or private key encryption scheme. This key is used for both encryption and decryption. On the other hand, in a public key or asymmetric key encryption scheme, both parties have their public and private keys. The public keys are publicly declared, while the secret keys are known only to the respective parties. The sender encrypts the message using the receiver's public key, while the receiver decrypts the cipher text using his secret key.

A block cipher is an encryption scheme that encrypts a block of data at a time. Examples of block ciphers include DES. 3-DES, AES-128 and AES-256. On the other hand, a stream cipher is an encryption scheme that encrypts a stream of data one element at a time. Here, the message stream is combined with a pseudorandom number sequence known as the keystream to generate the ciphertext. Examples of stream ciphers include SOSEMANUK and the SNOW series of ciphers.

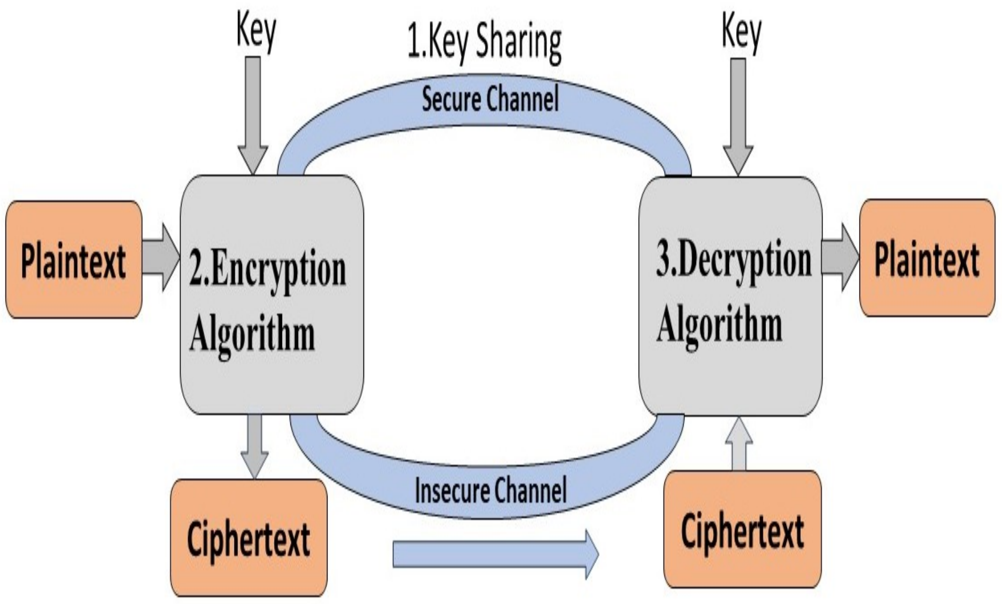


Fig. 2.1: Communication in Symmetric Key Primitive

This thesis deals with stream ciphers that are synchronous and additive. These terms are defined below.

Definition: 20 *A synchronous stream cipher is one in which both the plaintext and the ciphertext are generated independently of the keystream.*

Formally, we describe a synchronous stream cipher by three functions $\{fun_1, fun_2, fun_3\}$ as follows

- $S_{t+1} = fun_1(S_t, K)$, where S_0 is the initial state, K is the key and the function fun_1 evaluates the state S_{t+1} at the $t+1$-th instant using the key and the state S_t.
- $z_t = fun_2(S_t, K)$, where fun_2 evaluates the keystream value z_t at time instant t as a function of the key and the state at that time instant.
- $c_i = fun_3(z_i, m_i)$, where fun_3 is the output function that produces the ciphertext c_i by combining the keystream and the plaintext m_i.

Definition: 21 *A binary additive stream cipher is a synchronous stream cipher in which fun_3 is the XOR function, and the keystream, plaintext, and ciphertext are all binary streams.*

Additive stream ciphers are often constructed using Linear Feedback Shift Registers (LFSRs), which are extremely easy to implement in hardware and software.

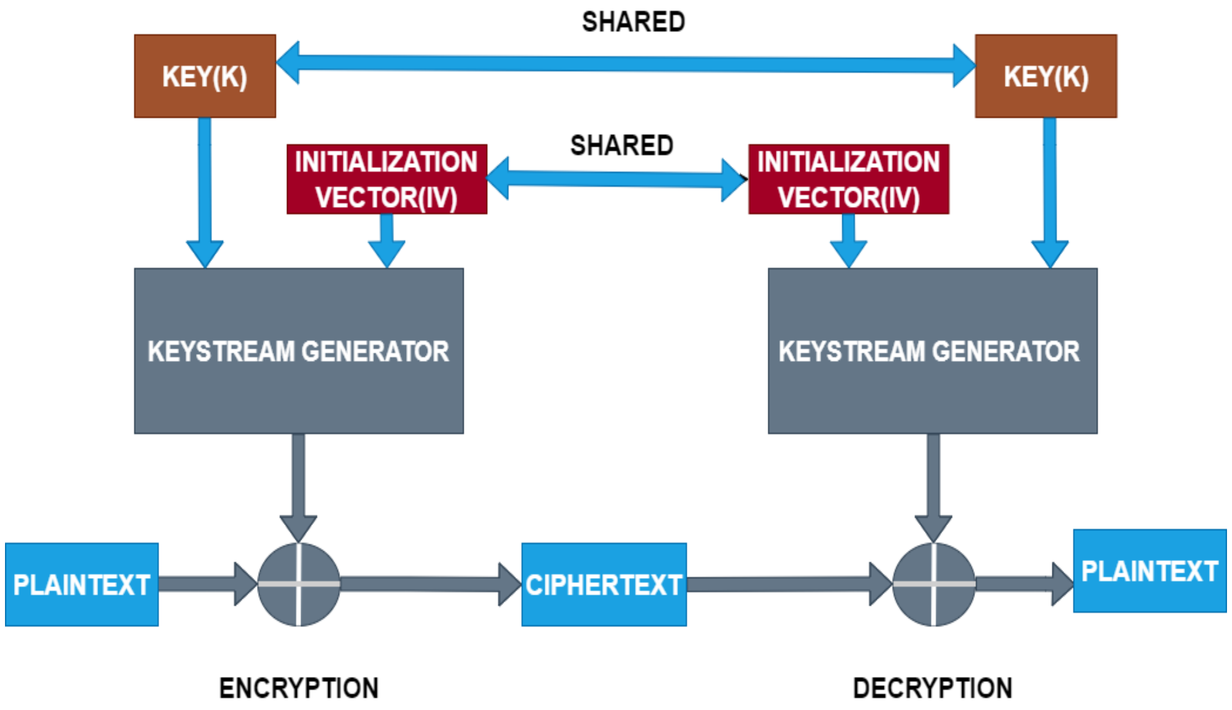


Fig. 2.2: General Model of Additive Stream Cipher

The security of a stream cipher can be assessed by quantifying the ability of a Probabilistic polynomial time(PPT) algorithm to distinguish its keystream from a random sequence. In this context, the computational indistinguishability of two sequences is defined as follows

Definition: 22 ***Computational Indistinguishability :*** *Two sequences $\{S_i\}_{i \in \mathcal{N}}$ and $\{T_i\}_{i \in \mathcal{N}}$ are computationally indistinguishable if for all Probabilistic Polynomial Time(PPT) algorithms B the following holds:*

$$ADV_B^{S,T}(n) = |\underset{y \leftarrow \mathcal{S}_i}{\mathcal{P}}[\mathcal{B}(y) = 1] - \underset{y \leftarrow \mathcal{T}_i}{\mathcal{P}}[\mathcal{B}(y) = 1]| \leq \epsilon(n) \tag{2.8}$$

where $ADV_B^{S,T}$ is the advantage of PPT B, n is the security parameter and $\epsilon : \mathbb{N} \to \mathbb{R}$, is a negligible function i.e. for every positive polynomial $p(.)$, there exists an integer $y > 0$ such that for every $n > y$ and $\epsilon(n) < \frac{1}{p(n)}$.

The security of a stream cipher can be proved by showing that its keystream is indistinguishable from a random sequence. However, since indistinguishability requires every PPT algorithm to have a negligible advantage, establishing it is difficult. Hence, most practical stream ciphers like SNOW series, Sosemanuk, ZUC, etc., are not provably secure. The security of these schemes is instead based on heuristics. These include resistance to various attacks like Algebraic Attacks, Fast Correlation Attacks, Time Memory and Data trade-off attacks, Guess and Determine attacks, Distinguishing Attacks, Differential Attacks, NIST Randomness Tests[45] etc.

2.2.1 LFSRs and $\sigma-$LFSRs

An LFSR is a machine that generates a binary sequence $\{x_0, x_1, x_2, \ldots\}$, which satisfies a relation of the following type.

$$x_N = \sum_{i=0}^{N-1} c_i \times x_{N-i} \quad j \geq N \tag{2.9}$$

Such a relation is called a linear recurring relation (LRR), and the polynomial $f(x) = c_0 + c_1x + \cdots + c_Nx^N$ is called the characteristic polynomial of the LFSR. The degree of an LFSR is the degree of its characteristic polynomial.

As shown in Figure 2.3, an LFSR consists of a set of delay blocks whose outputs are acted upon by a linear feedback function.

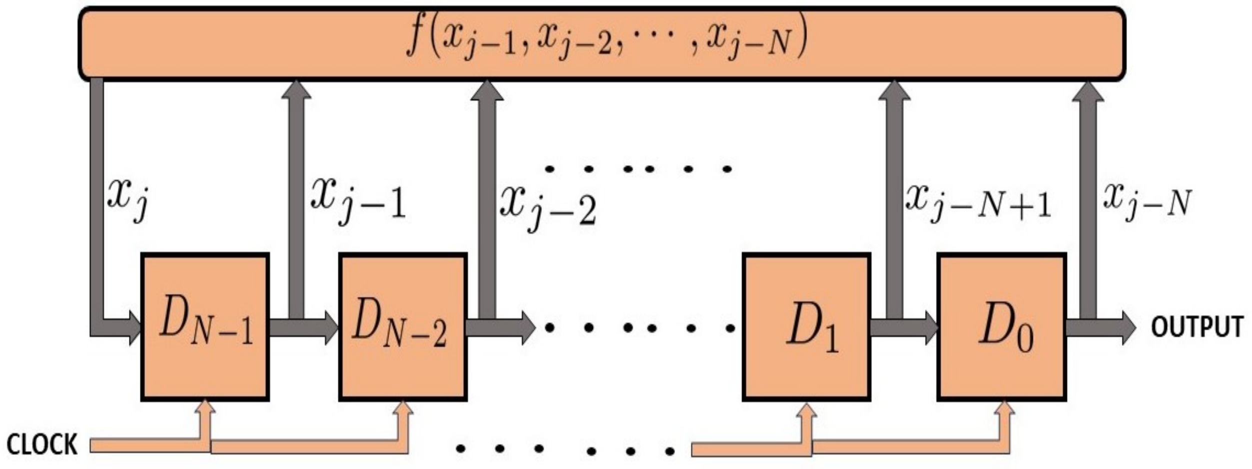


Fig. 2.3: Linear Feedback Shift Register(LFSR)

The state of an LFSR is the set of outputs of its delay blocks at a given time instant. Alternatively, a state vector consists of N consecutive elements of the sequence generated by the LFSR. Two consecutive state vectors $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$ are related by the equation $\mathbf{x}_{k+1} = \mathbf{x}_k M_f$ where M_f is given as follows;

$$M_f = \begin{bmatrix} 0 & 0 & \cdots & 0 & c_0 \\ 1 & 0 & \cdots & 0 & c_1 \\ 0 & 1 & \cdots & 0 & c_2 \\ \vdots & \ddots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 1 & c_{n-1} \end{bmatrix} \tag{2.10}$$

The matrix M_f is called the state transition matrix of the LFSR, and it is the companion matrix of the characteristic polynomial of the LFSR.

Given any periodic sequence, there exists an LFSR that generates it.

Definition: 23 *The minimal polynomial of a sequence is the characteristic polynomial of the LFSR with the least number of delay blocks that generate the sequence.*

The degree of the minimal polynomial of a periodic sequence is called the linear complexity of the sequence.

For a given degree N, the period of a sequence generated by an LFSR is maximum when its characteristic polynomial is primitive[46]. Such a sequence is called an m-sequence; its period is $2^N - 1$. Such sequences have good statistical properties such as balancedness, 2-level autocorrelation, and the span-N property. However, among all sequences having period $2^N - 1$, m-sequences have the least linear complexity. This is undesirable from a cryptographic point of view. Further, the feedback equation of an LFSR can be recovered from the generated sequence in $\mathbb{O}(N^2)$ time by the Barlecamp Massey Algorithm using $2N$ consecutive bits of the sequence.([47]). Therefore, LFSRs are used along with various mechanisms that introduce non-linearity to create secure stream ciphers.

2.2.2 σ-LFSRs

The advent of word-based processors and the need to effectively use them motivated the design of word-based LFSRS. Here, each delay block has multiple inputs and multiple outputs. Such LFSRs are used in stream ciphers like SOSEMANUK, SNOW 1.0, SNOW 2.0 and SNOW-3G.

A σ-LFSR is a special word-based LFSR that implements the following matrix linear recurring relation (MLRR);

$$\mathbf{D_{n+b}} = B_{b-1} * \mathbf{D_{n+b-1}} + B_{b-2} * \mathbf{D_{n+b-2}} + \cdots + B_0 * \mathbf{D_n} \tag{2.11}$$

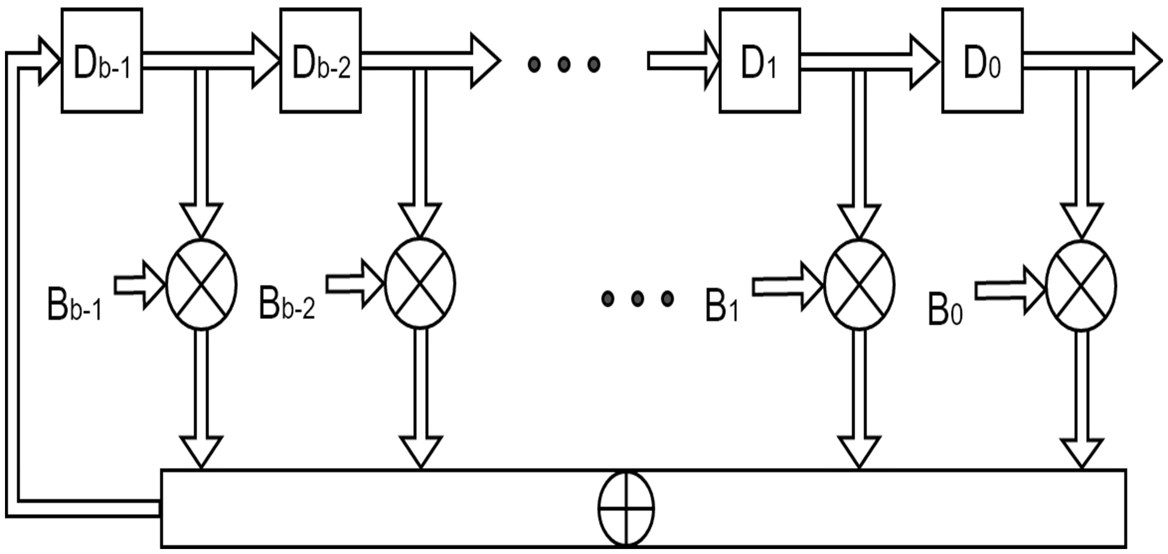


Fig. 2.4: Block Diagram of σ-LFSR

where each $\mathbf{D_i} \in \mathbb{F}_2^m$ and $B_i \in \mathbb{F}_2^{m\times m}$. A σ-LFSR that implements the MLRR given in Equation 2.11 has b. m-input m-output delay blocks. The output of the σ-LFSR is a sequence of vectors in $\mathbb{F}_2^m$. The matrices $B_0, B_1, \cdots, B_{b-1}$ are called the gain matrices of the σ-LFSR and the following matrix is defined as its configuration matrix.

$$C = \begin{bmatrix} 0 & I & 0 & \cdots & 0 \\ 0 & 0 & I & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & I \\ B_0 & B_1 & B_2 & \cdots & B_{b-1} \end{bmatrix} \in \mathbb{F}_2^{mb\times mb} \tag{2.12}$$

where $0, I \in \mathbb{F}_2^{m\times m}$ are the all-zero and identity matrices respectively. We shall refer to the structure of this matrix as the M-companion structure. The characteristic polynomial of the configuration matrix is called the characteristic polynomial of the σ-LFSR.

If $\mathbf{D_n}$ (considered as a row vector) is the output of the σ-LFSR at the n-th time instant, then its **state vector** at that time instant is defined as $\hat{\mathbf{D}}_\mathbf{n} = [\mathbf{D_n}, \mathbf{D_{n+1}}, \ldots, \mathbf{D_{n+b-1}}]^T$. This vector is obtained by stacking the outputs of all the delay blocks at the n-th time instant. The following equation relates two consecutive state vectors:

$$\hat{\mathbf{D}}_\mathbf{n+1} = C\hat{\mathbf{D}}_\mathbf{n} \tag{2.13}$$

The output of the σ-LFSR can be seen as a collection of m scalar sequences emanating from the m outputs of the first delay block. These scalar sequences are the component sequences of the σ-LFSR.

Theorem: 9 *The sequence generated by a $\sigma-$LFSR is periodic if and only if the first gain* *matrix $B_0 \in M_m(F_2)$ is invertible.*

If the sequence generated by a $\sigma-$LFSR is not periodic, it has a pre-period followed by a periodic sequence.

A σ-LFSR is considered primitive if its characteristic polynomial is primitive. Sequences generated by such σ-LFSRs have a maximum period. Each component sequence of a vector sequence generated by a primitive σ-LFSR has the same period as the output sequence of the σ-LFSR. Further, the minimal polynomial of each of these sequences is the same as the characteristic polynomial of the σ-LFSR. These sequences are shifted versions of each other. Given a primitive polynomial $f(x)$, define a map $f_M : GL(mb, \mathbb{F}_2) \mapsto GL(mb, \mathbb{F}_2)$ such that $f_M(Q) = Q \times M \times Q^{-1}$, where M is the companion matrix of primitive polynomial $f(x)$ and $Q \in GL(mb, \mathbb{F}_2)$. The following theorem gives the structure for the matrix Q, which ensures the matrix $f_M(Q)$ is a m-companion form.

Theorem: 10 *[28] Given $n = m \times b$ and a polynomial $f(x)$ of degree n. Let $M \in \mathbb{F}_2^{n\times n}$ be the companion matrix of $f(x)$. The matrix $C = Q \times M \times Q^{-1}$ is in the m- companion matrix form iff Q is an invertible matrix of the form*

$$Q = [v_1; v_2; \cdots ; v_m; v_1 * M; v_2 * M; \cdots ; v_m * M; \cdots ; v_1 * M^{b-1}; v_2 * M^{b-1}; \cdots ; v_m * M^{b-1}]$$

where $v_i \in F_2^{1\times n}$ is row vector.

The map f_M is strictly surjective. Hence, it induces a partition on $GL(mb, \mathbb{F}_2)$ where two matrices belong to the same equivalence class if they produce the same image under the map. The equivalence class of a matrix Q is denoted by $[Q]$. The following lemma characterizes matrices belonging to the same equivalence class.

Lemma: 11 *[28].Suppose, $Q_1, Q_2 \in GL(mb, F_2)$, the condition $f_M(Q_1) = f_M(Q_2)$ iff $Q_1 = Q_2 \times M^i$ for $i \in \{0, 1, 2, \cdots , 2^{mb} - 2\}$.*

As the characteristic polynomial of M is primitive, for $i \neq j$ and $q \neq 0$, $QM^i \neq QM^j$. Therefore, the cardinality of the set $[Q]$, denoted by $|[Q]|$, is $2^{mb} - 1$. Further, every matrix in an equivalence class has a unique first row. As $|[Q]|$ is equal to the number of non-zero vectors in $\mathbb{F}_2^{mb}$, given any non-zero vector $v \in \mathbb{F}_2^{mb}$, there is a unique element in $[Q]$ with first row v. This property is not unique to the first row and is valid for the other rows. Therefore, for every m-companion matrix C with the same characteristic polynomial as M, there exists a unique $Q \in GL(mb, F_2)$ with the following structure such that $C = QMQ^{-1}$:

$$Q = [e_1^n; v_2; \cdots ; v_m; e_1^n * M; v_2 * M; \cdots ; v_{m-1} * M; \cdots ; e_1^n * M^{b-1}; v_2 * M^{b-1}; \cdots ; v_{m-1} * M^{b-1}] \tag{2.14}$$

Thus, the number of m-comapnion matrices with the same primitive characteristic polynomial as M equals the number of matrices Q with the above structure. In [23], this number was conjectured to be the following:

$$N_P = \frac{|GL(m, \mathbb{F}_2)|}{2^m - 1} \times \frac{\phi(2^{mb} - 1)}{mb} \times 2^{m(m-1)(b-1)} \tag{2.15}$$

where ϕ represents Euler's totient.

For the cases where $m = 1$ and $m = n$, this conjecture is proved in [27]. For the case where $m = 2$, this conjecture is proved in [40]. This conjecture is constructively proved in the general case in [48]. Moreover, this proof gives an algorithm for calculating the feedback functions of the corresponding σ-LFSRs.

Definition: 24 ***Matrix state of $\sigma-$LFSR sequence*** *Let $S = \{S(i) \in F_2^m\}_{i\in\mathbb{Z}}$ be a sequence generated by a σ-LFSR with a primitive characteristic polynomial $f(x)$ of degree n. The i-th matrix state of S is a matrix of n consecutive elements of S starting from the i-th element*

$$Mat_S(i) = \{S(i), S(i+1), \cdots, S(i+b-1)\}_{m\times n}$$

The dimension of a $\sigma-$LFSR sequence S is the rank of any of its matrix states $Mat_S(i)$.

Theorem: 12 *Consider a σ-LFSR with b, m-input m-output delay blocks with configuration matrix $C = Q \times M \times Q^{-1}$ where the structure of Q is as given in Equation 2.14 and M is the companion matrix of the characteristic polynomial of the σ-LFSR. The first m rows of the matrix Q constitute a matrix state of the sequence generated by the σ-LFSR.*

Proof: For $1 \leq i \leq mb$, let the i-th column of Q be denoted by c_i. Further, for some $1 \leq i \leq mb$, let c_i be a state vector of the σ-LFSR. The next state vector of the σ-LFSR is $C \times c_i$. This vector is computed as follows

$$C \times c_i = (Q \times M \times Q^{-1}) \times c_i = Q \times M \times (Q^{-1} \times c_i) = Q \times M \times e_1^i = Q \times e_1^{i+1} = c_{i+1} \tag{2.16}$$

Thus, the next state vector is the next column of Q. Therefore, the n columns of Q are n consecutive state vectors of the σ-LFSR. Consequently, these vectors' first m rows constitute m consecutive outputs of the σ-LFSR. Hence, the first m rows of the matrix Q are a matrix state of the sequence generated by the σ-LFSR. $\square$

2.2.3 z-primitive σ-LFSR

z-primitive σ-LFSRs are special σ-LFSRs that are defined as follows:

Definition: 25 *Let S be a primitive σ-LFSR with b, m-input m-output delay blocks. Let its distance vector be $D = (d_1, \cdots, d_{m-1})$. If all the elements of the distance vector are divisible by $z = \frac{2^{mb}-1}{2^m-1}$, i.e. $z|d_i \; \forall \; 1 \leq i \leq m-1$, then S is called a z-primitive σ-LFSR.*

The following theorem gives the cardinality of the z-primitive σ-LFSRs set.

Theorem: 13 *[29] The number of $z-$primitive $\sigma-$LFSR of having b, m-input m-output delay blocks is $\frac{|GL_m(GF(2))|}{2^m-1} \times \frac{\phi(2^{mb}-1)}{mb}$, where $|GL_m(GF(2))|$ is the total $m \times m$ invertible matrices over $GF(2)$ and $\frac{\phi(2^{mb}-1)}{mb}$ is the number of primitive polynomials of degree mb over $GF(2)$.*

Given m and b, the set of z-primitive σ-LFSR configurations is a subset of primitive σ-LFSR configurations. For the case, $b = 1$, both the sets have the same cardinality (This follows from Theorem 1 and 6.3.1 in [28]). Therefore, when $b = 1$, every σ-LFSR configuration is a z-primitive σ-LFSR configuration.

Definition: 26 *Given $m, b \in \mathbb{Z}$ and a primitive polynomial f with degree mb, The z-set for the 3-tuple (m, b, f), denoted by z^f_{mb}, is the set of distance vectors of z-primitive σ-LFSRs having b, m-input m-output delay blocks and characteristic polynomial f.*

Note that there is a one-to-one correspondence between the set of distance vectors and the set of σ-LFSRs.

Theorem: 14 *[29] Let α and α^z be roots of primitive polynomials $f(x)$ and $g(x)$ having degrees mb and m respectively. The following map ϕ from z^f_{mb} and $z^g_m$is a bijection.*

$$\begin{aligned} \phi \colon z^f_{mb} &\longrightarrow z^g_m \\ (d_0, d_1, \cdots, d_{m-1}) &\mapsto (\frac{d_0}{z}, \frac{d_1}{z}, \cdots, \frac{d_{m-1}}{z}) \end{aligned}$$

If α and α^z are roots of primitive polynomials $f(x)$ and $g(x)$ having degrees mb and m respectively, the above theorem proves that there is one-to-one correspondence between z-primitive σ-LFSRs having b, m-input m-output delay blocks and characteristic polynomial $f(x)$ and σ-LFSR configurations with a single m-input m-output delay block and characteristic polynomial $g(x)$ (This is because every primitive σ-LFSR configuration is a z-primitive σ-LFSR configuration when $b = 1$.)

2.3 Properties of Cryptographic Boolean Functions

In this section, we briefly discuss the properties of cryptographic boolean functions.

Definition: 27 ***Boolean Function:*** *An n variable Boolean function is a map from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. [49]*

Definition: 28 ***Hamming Weight:*** *The Hamming Weight of a binary sequence S, $HW(S)$, is the number of set bits (1s present in S.*

Definition: 29 ***Hamming Distance:*** *The Hamming Distance of two same lengths sequences S_1 and S_2, denoted by $HD(S_1, S_2)$, is defined as in the number of positions they differ.*

$$HD(S_1, S_2) = HW(S_1 \oplus S_2)$$

Definition: 30 ***Truth Table Representation:*** *For a function f, the ordered 2^n tuple $T_f = (f(\alpha_0), f(\alpha_1), \ldots, f(\alpha_{2^n-1}))$ is called truth table of f. Each function can be uniquely described by its truth table T_f.*

Definition: 31 ***Algebraic Normal Form Representation***

An n-variable Boolean function f can be written in the Algebraic Normal Form (ANF) representation as follows [50].

$$f = \underset{I \in P(l)}{\oplus} a_I \left(\prod_{i \in I} x_i \right) = \underset{I \in P(n)}{\oplus} a_I x^I \tag{2.17}$$

where, $P(n)$ denotes the power set of $N = \{1, ..., n\}$ and f belongs to the ring $\mathbb{F}_2[x_0, x_1, \ldots, x_{n-1}]/ < x_0^2 + x_0, x_2^2 + x_2, \ldots, x_{n-1}^2 + x_{n-1} >$.

The algebraic degree of the Boolean function f with the algebraic normal form representation given in Equation 2.17 is defined as $\{max|T||a_T \neq 0\}$, where $|T|$ represents the cardinality of the set T.

Definition: 32 ***Walsh Hadamard Transformation(WHT)*** *Walsh Hadamard Transformation is a map from $V_n \to \mathbb{Z}$, which is defined as follows*

$$W_f(u) = \sum_{x \in \mathbb{F}_2^n} f(x)(-1)^{<u.x>}$$

For the sign function, it can be written as

$$W_{\hat{f}}(u) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus <u.x>}$$

Numbers $W_{\hat{f}}(u)$ are called Walsh Hadamard coefficients of a Boolean function f.

It explains the difference between the number of places $f(x) == l_u(x)$ and the number of places $f(x) \neq l_u(x)$ where $l_u(x) =< u.x >$ is a linear function where $x \in \mathbb{F}_{\not{2}}{}^n$. We define bias($\epsilon$) as $Pr(f(x) == l_u(x)) - \frac{1}{2} = \frac{|W_f(u)|}{2^{n+1}}$. It is used to estimate f(x) from the linear function $l_u(x)$, which is very important in Correlation attacks.

Definition: 33 ***Balancedness:*** *A boolean function is said to be balanced if* $HW(T_f) = 2^{n-1}$.

It implies $W_f(x) = 0 \ \forall x \in V_n$.

Affine Function An affine function $L_{\alpha,c}$is defined as

$$L_{\alpha,c} : \mathbb{V}_n \to \mathbb{F}_2$$

where $L_{\alpha,c}(x) = \alpha.x + c = \sum_{i=1}^{n}(\alpha_i * x_i) + c$ and $\alpha, x \in \mathbb{V}_n$ and $c \in \mathbb{F}_2$. if $c = 0$ then $L_{\alpha,0}(x) = \alpha * x$ is a linear function. We consider A_n as a set of n variable affine functions.

Definition: 34 ***Nonlinearity*** *[51] The nonlinearity of a n-variable boolean function f is the minimum hamming distance from* A_n.

$$nl(f) = \min_{\forall l \in A_n} HD(f, l)$$

Nonlinearity can be expressed with Walsh spectra of f as

$$nl(f) = 2^{n-1} - \frac{1}{2} \max_{u \in \mathbb{V}_n} \left| W_{\widehat{f}}(u) \right|$$

The largest Walsh coefficient of a boolean function is the key value to determine the nonlinearity. This helps to find the nearest affine function of a Nonlinear boolean function.

Example: 3 *[51] let* $f(x_1, x_2, x_3) = 1 + x_1 + x_2 + x_2x_3 + x_1x_2x_3$ *is a boolean function on* $\mathbb{V}_n$. *So it's truth table is* $[1, 1, 0, 1, 0, 0, 1, 1]$ *and walsh coefficients are* $[-2, 2, 2, -2, -2, 2, -6, -2]$. *So maximum value of walsh coefficients is 6 ,* $\left| W_{\widehat{f}}(\alpha_6) \right| = 6$. *The nearest linear function of f will be* $l_{\alpha_6,1} =< \alpha_6.x > +1 = 1 + x_1 + x_2$. *We can check easily that* $d(f, l_{\alpha_6,1}) = 1$

Definition: 35 ***Correlation Immunity[49]:*** *A n variable boolean function* $f(x_1, x_2, \cdots, x_n)$ *is called correlation immune of order l,* $1 \le l \le n$*, if for any fixed subset of k values the probability, given the output* $f(x_1, x_2, \cdots, x_n)$*, is always* $\frac{1}{2^k}$. *In other way, f is correlation immune of order l for any l subset of i.i.d* $(x_{i_1}, x_{i_2}, \cdots, x_{i_l})$ *if the following is true*

$$I((x_{i_1}, x_{i_2}, \cdots, x_{i_l}); f(x_1, x_2, \cdots, x_n)) = 0$$

,where $I(x; y)$ *is the mutual information.*

In addition to that, f is called m-resilinet if f is balanced. An important property of m-resilient boolean function to Walsh coefficient is as follows

$$W_f(x) = 0, \forall x, 1 \le HW(x) \le l$$

Definition: 36 ***Algebraic Immunity[52]:*** *If* $f : \mathbf{F}_2^n \rightarrow \mathbb{GF}(2)$ *is* $n-$*variable boolean function, the Algebraic Immunity of* f*, AI(f), is the lowest degree annihilator function of* f*,* g *such that* $f * g = 0$ *or* $(1 + f) * g = 0$*. A boolean function with optimal algebraic immunity (*$\lceil \frac{n}{2} \rceil$*) is said to be one of the good cryptographic characteristics.*

Definition: 37 ***Strict Avalanche Criterion[49]:*** *A boolean funtion f(x) with n variable satisfies the SAC(k) iff* $f(x) \oplus f(x + a)$ *is balanced for* $1 \leq HW(a) \leq k$*.*

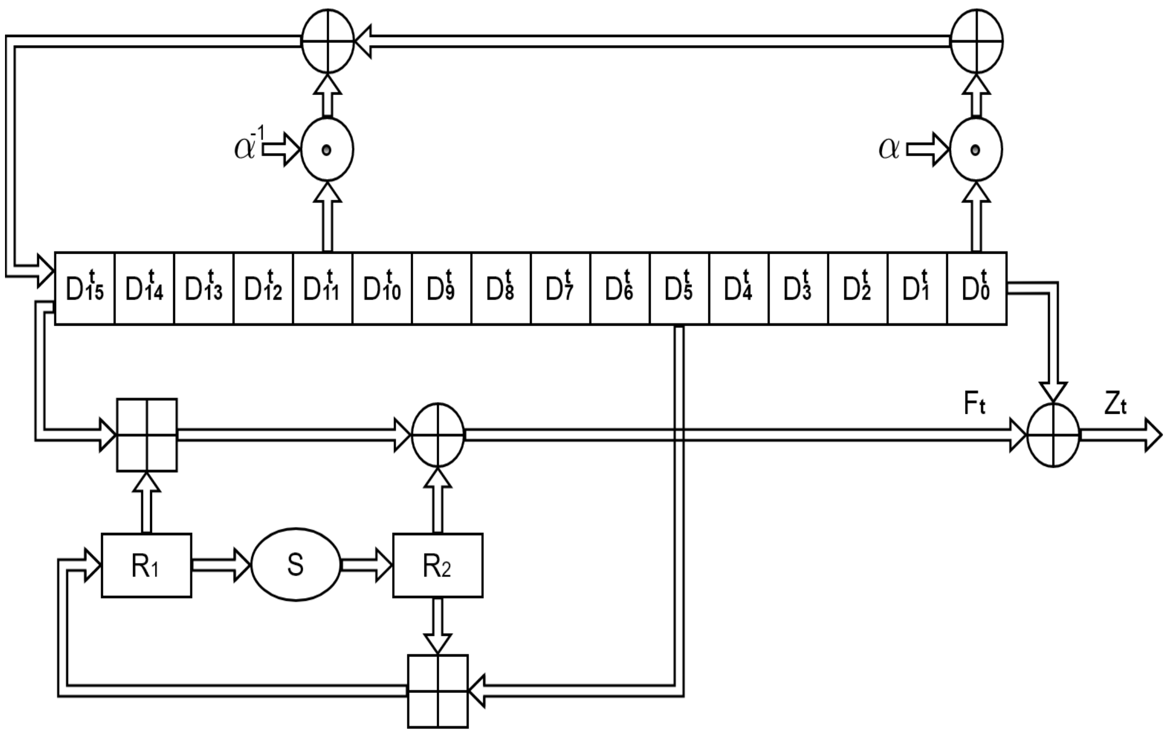
SAC property explains that if we change one bit of the input, the output will change with probability $\frac{1}{2}$. Moreover, An S-Box with a high SAC value resists the differential cryptanalysis attack.

2.3.1 S-Boxes

An (n, m) S-Box(Substitution Box) f is a vectorial boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ where each component boolean function $(f_1, f_2, \cdots, f_n)$ is represented as $f_i : \mathbb{F}_2^n \rightarrow \mathbb{GF}(2)$ is a nonlinear boolean function. In cryptography, $S - box$ is used for substitution, obscuring the relation between key and ciphertext(Confusion Property).

2.4 SNOW 2.0

The SNOW series of word-based stream ciphers was first introduced in [53]. This version of SNOW is known as SNOW 1.0. This was vulnerable to a linear distinguishing attack and a guess and determine attack [54]. SNOW 2.0 (Adopted by ISO/IEC standard IS 18033-4) was introduced later in [16] as a modified version of SNOW 1.0. The block diagram of SNOW 2.0 is shown in figure 2.5. The cipher works in two phases: the initialization stage and the keystream generation phase.



α^{-1}
α
D^t_{15} D^t_{14} D^t_{13} D^t_{12} D^t_{11} D^t_{10} D^t_{9} D^t_{8} D^t_{7} D^t_{6} D^t_{5} D^t_{4} D^t_{3} D^t_{2} D^t_{1} D^t_{0}
F_t
Z_t
R_1
S
R_2

Fig. 2.5: The block diagram of SNOW 2.0

In figure 2.5, $+$ and $\boxplus$ represent bit wise XOR (addition in the field $GF(2)$) and integer addition modulo 2^{32} respectively. As shown in figure 2.5, the keystream generator in SNOW 2.0 consists of an LFSR and an FSM (Feedback State Machine). The LFSR consists of 16 delay blocks $D_i \in F_2^{32}$. It implements the following linear recurring relation:

$$D_{15}^{t+1} = \alpha^{-1} D_{11}^t + D_2^t + \alpha D_0^t.$$

where α is the root of the following primitive polynomial

$$G_S(x) = (x^4 + \beta^{23}x^3 + \beta^{245}x^2 + \beta^{48}x + \beta^{239}) \in \mathbb{F}_{2^8}[X]$$

where β is the root of the following primitive polynomial.

$$H_S(x) = x^8 + x^7 + x^5 + x^3 + 1 \in \mathbb{F}_2[X]$$

The FSM contains two 32-bit registers, R_1 and R_2. These registers are connected using an S-Box made using four AES S-boxes. This S-box serves as the source of nonlinearity. The output of the FSM at time t, F_t satisfies the following equation

$$F_t = (D_5^t \boxplus R_1^t) \oplus R_2^t, t \geq 0 \tag{2.18}$$

For a 256- bit key and a 128-bit initialization vector (IV), SNOW 2.0 is initialized as follows:

$$(D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7) = (k_1 \oplus IV_1, k_2, k_3, k_4 \oplus IV_2, k_5, k_6, k_7, k_8)$$
$$(D_8, D_9, D_{10}, D_{11}, D_{12}, D_{13}, D_{14}, D_{15}) = (k_1 \oplus 1, k_2 \oplus 1, k_3 \oplus 1, k_4 \oplus 1, k_5 \oplus 1, k_6 \oplus 1, k_7 \oplus 1, k_8 \oplus 1)$$

where, **1** represents the value $2^{32}-1$, Key(K)=$\{k_1, \cdots, k_8\}$,$k_i \in F_2^{32}$ and Initialization Vector(IV) = $\{IV_1, IV_2, IV_3, IV_4\}$ where, $iv_i \in F_2^{32}$. For a 128-bit key and a 128-bit initialization vector (IV), SNOW 2.0 is initialized as follows:

$$\begin{aligned}(D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7) &= (k_1 \oplus 1, k_2 \oplus 1, k_3 \oplus 1, k_4 \oplus 1, k_1, k_2, k_3, k_4)\\ (D_8, D_9, D_{10}, D_{11}, D_{12}, D_{13}, D_{14}) &= (k_1 \oplus 1, k_2 \oplus 1 \oplus IV_4, k_3 \oplus 1 \oplus IV_3, k_4 \oplus 1, k_1 \oplus IV_1, k_2, k_3)\\ D_{15} &= k_4 \oplus IV_1\end{aligned}$$

runs for 32 clock cycles. In each clock cycle, the state of SNOW 2.0 is updated as follows

$$D_{15}^{t+1} = \alpha^{-1} D_{11}^t + D_2^t + \alpha D_0^t + F_t \tag{2.19}$$

Further, the keystream at time t, z^t follows the equation $z^t = (D_{15}^t \boxplus R_1^t) \oplus R_2^t$. In SNOW 2.0, the registers R_1 and R_2 updated as follows

$$\begin{aligned} R_1^{t+1} &= D_5^t \boxplus R_2^t \\ R_2^{t+1} &= S(R_1^t) \end{aligned} \tag{2.20}$$

where S is a S-Box over F_2^{32}, composed of four parallel AES S-boxes[3](S_1) followed by the AES MixColumn transform matrix (denoted by $N_1 \in \mathbb{F}_{2^8}^{4\times4}$). In order to define N_1 over $\mathbb{F}_{2^8}$ for field multiplication, the polynomial $y^8 + y^4 + y^3 + y + 1 \in F_2[y]$ is used. The working procedure of S is as follows:

$$\begin{bmatrix} b_0^{t+1} \\ b_1^{t+1} \\ b_2^{t+1} \\ b_3^{t+1} \end{bmatrix} = N_1 \times \begin{bmatrix} S(b_0^t) \\ S(b_1^t) \\ S(b_2^t) \\ S(b_3^t) \end{bmatrix} = \begin{bmatrix} Y & Y+1 & 1 & 1 \\ 1 & Y & Y+1 & 1 \\ 1 & 1 & Y & Y+1 \\ Y+1 & Y & 1 & 1 \end{bmatrix} \begin{bmatrix} S(b_0^t) \\ S(b_1^t) \\ S(b_2^t) \\ S(b_3^t) \end{bmatrix} \tag{2.21}$$

where, $b = (b_0^t, b_1^t, b_2^t, b_3^t)$, $b_i \in F_2^8$, be the input to S at time t and the output at $(t+1)$ is $(b_0^{t+1}, b_1^{t+1}, b_2^{t+1}, b_3^{t+1}) \in \mathbb{F}_2^{32}$.

2.5 SNOW 3G

SNOW 3G keeps all the functionality of SNOW 2.0 while adding a third register R_3 and a transformation $SBox2$ to the FSM.

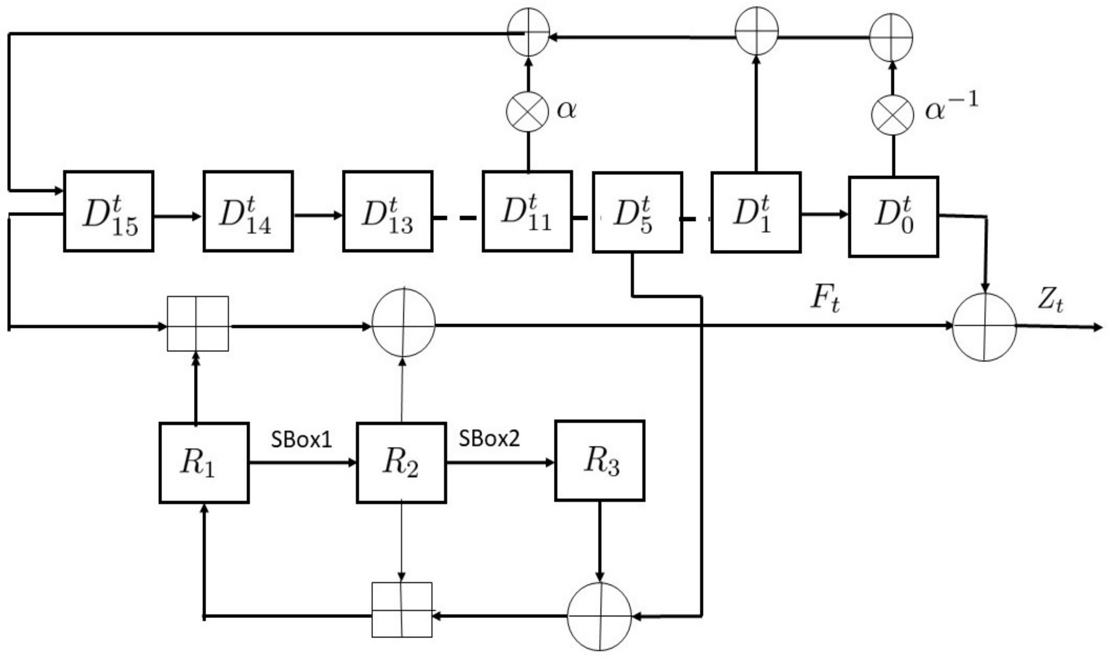


Fig. 2.6: Building Block of SNOW 3G Cipher

Let the outputs of the registers in the FSM of SNOW3G at time instant t be denoted by R_1^t, R_2^t and R_3^t respectively. Let the output of the FSM at time instant t be denoted by F_t. The following equations govern the FSM:

$$\begin{aligned}
F_t &= (D_{15}^t \boxplus {R_1}^t) \oplus {R_2}^t, t \geq 0 \\
R_1^{t+1} &= (D_5^t \oplus R_3^t) \boxplus R_2^t \\
R_2^{t+1} &= SBox1(R_1^t) \\
R_3^{t+1} &= SBox2(R_2^t)
\end{aligned}$$

$SBox1$ in the above equation is the same as the S-Box of SNOW 2.0, while $SBox2$ is another bijection over $GF(2^{32})$, based on the Dickson polynomial.

2.6 Known Plaintext Attacks(KPA) on Stream Ciphers:

A known-plaintext attack (KPA) is a cryptanalysis technique wherein, given access to the plaintext and the corresponding ciphertext, the attacker tries to retrieve secret information like the secret key. In this section, we discuss some KPAs on Stream Cipher.

2.6.1 Algebraic Attacks:

An Algebraic attack [55, 56, 57] is a known plaintext attack on a stream cipher. In an algebraic attack, the cryptosystems' key is retrieved by solving a system of multivariate polynomial equations over a finite field. The degree of these polynomial equations is reduced by multiplying them with low-degree 'annihilators'. This attack is used against various models like the nonlinear combiner, the nonlinear filter generator and the nonlinear combiner with memory. Consider a cryptosystem that uses an LFSR with a k-bit state vector. Such an LFSR is updated by a linear update function $L : \mathbb{F}_2^k \to \mathbb{F}_2^k$. Let the initial state of the LFSR be $S^0 = \{s_0, s_1, \ldots, s_{k-1}\}$. At the t-th clock, the keystream output is given by $z_t = f(S^t)$, where f is a nonlinear function. $S^t = L^t(S^0)$ denotes the state when the linear function L is operated t-times on the initial state S^0. The problem is to recover the initial state $S^0 = \{s_0, s_1, \ldots, s_{k-1}\}$. In a known plaintext attack, l keystream bits(say,$z_{k_1}, z_{k_2}, \ldots, z_{k_l}$) are known to the adversary. These are used to generate a system of equations of degree deg(f) as follows:

$$\begin{aligned} f(L^{k_1}(S^0)) &= z_{k_1} \\ f(L^{k_2}(S^0)) &= z_{k_2} \\ &\vdots \\ f(L^{k_l}(S^0)) &= z_{k_l} \end{aligned}$$

The complexity of solving the system of equations increases if the degree of the nonlinear functions f is high. An algebraic attack involves generating low-degree equations using some weakness in the internal structure of the nonlinear functions. The main idea[52] is to find polynomials g such that the degree of either gf or $g(1 + f)$ is significantly lower than f. These polynomials are multiplied with the abovementioned equations to generate lower-degree equations. These equations are then solved using techniques like Linearization, Extended Linearization(XL), SAT Solver, Grobner Basis, etc.

The article [30] explains an Algebraic Attack on SNOW 2.0. This attack first attempts to break a modified scheme where the $\boxplus$ operator is approximated by $\oplus$. It solves 2^{17} quadratic equations using the Linearization technique. The time complexity of solving this equation is 2^{51}. Further, the article[31] describes an algebraic attack that uses linearly independent nonlinear equations generated from $\boxplus$. The time complexity of this attack is 2^{294}.

2.6.2 Fast Correlation Attacks:

A correlation attack is a known-plaintext attack extensively used against LFSR-based stream ciphers. Such attacks utilize the correlation between the output of the keystream and that of the LFSR. Here, the effect of the non-linearity is modelled as a noise added to the output of the LFSR. In other words, the keystream generation equation is considered as $z^t = s^t + e^t$, where z^t and s^t are the value of the keystream and the output of the LFSR at the t-th time instant. e^t is the binary noise that models the nonlinear function. The idea of launching a correlation attack by exploiting the bias in e^t was introduced in [58]. Here, given a keystream of length N, a guess of the initial state is used to generate the first N bits of the LFSR sequence. The two sequences are then added (EXORed), and the bias of the resulting sequence is checked. The guess is considered potentially correct if this bias is above a certain threshold. The time complexity of this simple algorithm is $N2^n$, where N is the size of the keystream and n is the size of the LFSR. The article[59] proposes a fast correlation that avoids the exhaustive search of the initial state. Here, parity check equations remove the noise e^t from z^t and recover the LFSR sequence s^t. Such schemes are limited by their requirement of a low number of LFSR taps and a significantly high bias. To circumvent this problem, the one-pass algorithm [60] was developed and successfully applied on various stream ciphers [34, 35]. Here, the n bit initial state of the LFSR is divided into two parts, of m bits and $n - m$ bits, where m-bits of the LFSR are guessed through an exhaustive search and $n - m$ bits are found using parity check techniques.

An FCA can be seen as a problem of decoding an $[N, n]$-linear code [61]. If a^t is the t-th column of the generator matrix, parity check equations are given by $e^t =< S^0, a^t > \oplus z^t$. The code size is reduced by replacing pairs of columns of the generator matrix whose last m entries are common by their sum(XOR). This procedure reduces the number of bits to be estimated to $n - m$. This, however, increases the noise in decoding. In [62], this technique is generalized using the generalised birthday problem[62]. Moreover, using the Fast Walsh Hadamard transformation(FWHT) to verify the correct guess reduces the exhaustive search complexity from $N2^n$ to $N + 2^n$. This process is iteratively used to find the initial state S_0.

The feedback polynomial of the LFSR plays a pivotal role in FCA. Several FCAs against SNOW 2.0 and SNOW 3G are reported in the literature. The first FCA on SNOW 2.0[34] had a time complexity of $2^{212.38}$. It uses the one-pass algorithm along with a generalised birthday attack. The attack in [35] considers the LFSR in SNOW 2.0 as an LFSR over F_{2^8}, which implements a linear recurring relation of degree 64. Here, Wagner's k-tree approach, described in [62], is used to create parity check equations. The time complexity of this attack is $2^{164.5}$, around 2.49 times better than the one in [34]. However, this attack depends on the LFSR's feedback equation (expressed as an equation over $F_{2^{32}}$). A better correlation $2^{14.411}$ for the FSM approximation equation in SNOW 2.0 is found using a linear mask search [63]. It recovers the key of SNOW 2.0 with a time complexity of $2^{163.91}$. To determine the state of the LFSR on SNOW3G, [64] outlines a vectorized linear approximation attack with a bias value of 2^{-40} and time complexity of 2^{177}. Another attack model is put forth by [1]. It is based on a modified Wagner K-tree technique and a linear approximation of a composition function. This attack has a time complexity of $2^{216.86}$ for SNOW 2.0 and $2^{222.33}$ for SNOW 3G. This technique uses the linear recurring relation of degree 512 that the output of the LFSR satisfies.

2.6.3 Distinguishing Attack

In a distinguishing attack, the keystream generated by a pseudo-random number generator (PRNG) is distinguished from a random sequence of the same length. This is done by finding a relation satisfied by the keystream bits with a higher probability than those of a random sequence. A higher bias reduces the time and memory complexity of such attacks. One way of finding such a relation is by using linear masks. This method is commonly used in attacks on SNOW 2.0 [32, 33, 34]. It adapts the linear cryptoanalysis method given in [65] to stream ciphers.

2.6.4 Guess and Determining Attack

In a Guess and Determine Attack, the attacker aims to estimate the values of a minimum number of variables using which the complete sequence can be constructed. For SNOW 2.0, this includes the values of the outputs of the delay blocks of the LFSR and the outputs of the registers of the FSM at some time instant. This is done by guessing some of the values and determining the rest of them using system equations. If the sequence generated using these estimates matches the output of the key-stream generator, then the guesses are deemed correct. Otherwise, a fresh set of guesses are considered. The set of variables whose values are guessed is the basis for the attack. For both SNOW 2.0 and KDFC-SNOW, these variables take their values from $\mathbb{F}_2^{32}$. Hence, if the basis size is k, then the probability of a correct guess is 2^{-32k}. Thus, on average, one needs $\mathcal{O}(2^{-32k})$ attempts to make a correct guess. Therefore, the problem is finding a basis for the minimum possible size. A systematic Vitterbi-like algorithm is given in [66]. The complexity of this attack was found to be 2^{265}([66]) for SNOW2.0. The complexity of this attack was reduced to 2^{192} in ([67]) by incorporating a couple of auxiliary equations.

2.6.5 Cache Timing Attack

It is a kind of side-channel analysis attack where the adversary accesses the cache memory before or after the generation of each keystream bit. In schemes like SNOW 2.0 (and other schemes in the SNOW series), multiplication in the finite field is performed using look-up tables corresponding to the non-zero constants in the feedback equation. Further, another look-up table is used for the implementation of the S-BOX. These implementations are cache-friendly. However, the adversary can extract secret information about the LFSR by monitoring the cache access. According to the attack model of [36, 68], the adversary uses two synchronous oracles:

1 KEYSTREAM(J) It returns the $J-$th keystream block.

2 SCA-KEYSTREAM(J) It returns the unordered list of cache accesses done while creating the $J-$th keystream.

In SNOW 2.0 and SNOW 3G there are multiplications over $\mathbb{F}_{2^{32}}$ that are done in every clock cycle viz. multiplication by α and α^{-1}. These multiplications are implemented as follows

$$\alpha * x = (x << 8) \oplus T_1[x_3] \tag{2.22}$$

$$\alpha^{-1} * x = (x >> 8) \oplus T_2[x_0] \tag{2.23}$$

where $x = (x_3||x_2||x_1||x_0) \in F_{2^{32}}, \quad x_i \in F_{2^8}$ and T_1, T_2 are two 8×32 tables. Thus, from the list of cache accesses, the adversary can extract 8-bits of information, viz. the first four bits of S_t and the last four bits of S_{t+11}. The adversary then gathers all linear equations formed from these bits for (512/8=64) clock cycles. The state of the LFSR can then be found by solving these linear equations.

2.7 Summary

This chapter studies mathematical concepts and algorithms to understand known-plaintext attacks on the cyphers SNOW 2.0 and SNOW 3G. We list out all the attacks in the following table. We aim to hide the cypher's feedback polynomial to withstand known plaintext attacks. In the following chapters, we give methods to hide the feedback polynomial.

Attack Name	Versions	Time Complexity
Algebraic Attack	Billet et. al.,2005[30]	2^{51}
	Courtois et al., 2008[31]	2^{294}
Distinguishing Attack	Watanabe et. al.,2003[32]	Keystream Required: 2^{225}
	Maximov et. al.,2005[69]	Keystream Required: 2^{205}
	Nyberg et. al.,2006[33]	Keystream Required: 2^{174}
Fast Correlation Attack	Lee et. al.,2008[34]	$2^{204.38}$
	Zhang et. al.,2015[35]	$2^{164.15}$
	Todo et. al.,2018[70]	$2^{162.91}$
	Yang et. al.,2019[64]	2^{177}(SNOW 3G)
	Gong et. al,.2020[1]	$2^{162.86}$,$2^{222.32}$(SNOW 3G)
Guess and Determine Attack	Ahmadi et. al,,2009[66]	2^{265}
	Nia et. al, 2014[67]	2^{192}
Cache Timing Attack	Leander et al.,2009[36]	2^{32}

Chapter 3

Key Dependent Feedback Configuration Scheme

This chapter proposes and evaluates a method for generating key-dependent feedback configurations (KDFC) for σ-LFSRs. σ-LFSRs with such configurations can be applied to any stream cipher that uses a word-based LFSR. A configuration generation algorithm uses the secret key(K) and the Initialization Vector (IV) to generate a new feedback configuration after the initialization round. It replaces the older known feedback configuration. The keystream is generated from this new feedback configuration and the FSM. We have mathematically analysed the feedback configurations generated by this method. As a test case, we have applied this method to SNOW 2.0 and have studied its impact on resistance to algebraic attacks. Besides resisting algebraic attacks, SNOW 2.0 can also withstand other attacks like Distinguishing Attacks, Fast Correlation Attacks, Guess and Determining Attacks and Cache Timing Attacks. Further, we have also tested the generated keystream for randomness and briefly described its implementation and the challenges involved.

The rest of this chapter is organized as follows. Section 3.1 examines σ-KDFC and its time complexity. Section 3.2 analyses the algebraic degree of the elements of the feedback function generated by σ-KDFC. Section 3.3 discusses the interconnection of σ-KDFC with the FSM of SNOW and its security against various cryptographic attacks. Section 3.4 concludes the chapter.

3.1 σ-KDFC

As mentioned in Chapter 2, in the case of σ-LFSRs, many possible feedback configurations have the same characteristic polynomial. For a given primitive polynomial, the number of such configurations is given by

$$N_P = \frac{|GL(m, \mathbb{F}_2)|}{2^m - 1} \times \frac{\phi(2^{mb} - 1)}{mb} \times 2^{m(m-1)(b-1)}. \tag{3.1}$$

where 3.1, $GL(m, \mathbb{F}_2)$ is the general linear group of non-singular matrices $\in \mathbb{F}_2^{m \times m}$, and ϕ represents Euler's totient function. This has been constructively proved in [48]. This proof gives an algorithm for calculating such feedback functions for a primitive polynomial. Before proceeding to the construction of a key-dependent feedback configuration, we briefly describe this generation algorithm.

Algorithm 1 Invertible Matrix Generation Algorithm

Input:

1. A full rank matrix stored in $M \in \mathbb{F}_2^{m \times m}$.
2. A set of $(n - m)$ primitive polynomials of degrees $\{m, m + 1, \cdots, n\}$ stored in an array L. The corresponding companion matrices are $P_{L(i)}$ for $m \leq i \leq n$. These polynomials can be arbitrarily selected from the lists of primitive polynomials, which are available in the literature.
3. A set of random binary vectors $R_i \in \mathbb{F}_2^{1 \times n}, i \in \{0, \cdots, n - m\}$,(These are the $(n - m)$ random vectors generated in the first step of the initialization process).

```
1: procedure FIND_INV(M,L,R)
2:     $Y \leftarrow M$
3:     $d \leftarrow Dimension(M)$
4:     $t \leftarrow 0$
5:     while $t \leq (n - m)$ do
6:         $c \leftarrow Y[(t \pmod{m});]$
7:         $Val \leftarrow Lin\_solver(c, P_{L[t]})$
8:         $Y \leftarrow Y \times Val$
9:         $Y[:, d + t - 1] \leftarrow R_t$
10:        $Y[c, d + t - 2] \leftarrow 0$
11:        $Y[c, d + t - 1] \leftarrow 1$
12:        $t \leftarrow t + 1$
13:    end while
14: end procedure
```

Note that at every step, the size of c vector increases. Therefore the time taken for each iteration of Algorithm 1 increases with each iteration. To circumvent this problem, a few of these iterations could be run offline in a server and the resulting Y matrix could be made public. Let this matrix be denoted by Y_{init}. The remaining iterations can be done during the initialization phase of the keystream generator. When this is done, the variable Y in Algorithm 1 will be initialized as Y_{init}, and the variable t will be initialized as $k \pmod{m}$, where k$(m < k < n)$ is the number of iteration done offline. Further, the number of random vectors generated in the first step will now be $n - m - k$. The array L will contain primitive polynomials with degrees $(m + k, \cdots, n)$. The for loop will run $n - m - k$ times.

In Algorithm 1, the $Val \leftarrow Lin\_solver(c, P_{L[t]})$ is calculated using the Algorithm 2 as follows.

Algorithm 2 Find the power of A, c, such that $c \times Val = e^1_{|c|}$

Input:

1. $c \in \mathbb{F}_2^{1\times n}$.
2. Companion matrix $A \in \mathbb{F}_2^{n\times n}$ of a primitive polynomial of degree n over $GF(2)$.

```
1: procedure LINEAR_SOLVER(c,A)
2:     $M \leftarrow \begin{bmatrix} c \\ c \times A \\ c \times A^2 \\ \vdots \\ c \times A^{n-1} \end{bmatrix}_{n\times n}$
3:     Solve the equation $y \times M = e^n_1$.
4:     $l \leftarrow y[0] \times I + y[1] \times M \cdots y[n-1] \times M^{n-1}$.
5:     Return $l$.
6: end procedure
```

Algorithm 2 is used as a subroutine to find out the matrix l. Basically the l matrix is used to convert $c \times l = e^n_1$, where $l = A^d$, where $d \in \{1, \cdots, 2^n - 1\}$. As A is a companion matrix of a primitive polynomial, it is full rank. This results $\{c \times A^i\}$ for $i \in \{0, 1, \cdots, n-1\}$ as a basis vector for $\mathbb{F}_2^n$. This space is called Krylov subspace [71].

Algorithm 3 **Configuration Matrix Generation**

Input:

1. Companion matrix of a primitive polynomial of degree n, P_z.
2. $Y \in \mathbb{F}_2^{m \times n}$ from Algorithm 1.

```
1: procedure CONFIG_GEN($P_z, Y$)
2:     $P_z \leftarrow P_{L[n-m-1]}$
3:     $Q \leftarrow \begin{bmatrix} Y[0:,] \\ Y[1:,] \\ \vdots \\ Y[m-1:,] \\ Y[0:,] \times P_z \\ Y[1:,] \times P_z \\ \vdots \\ Y[m-1:,] \times P_z \\ \vdots \\ Y[0:,] \times P_z^{b-1} \\ Y[1:,] \times P_z^{b-1} \\ \vdots \\ Y[m-1:,] \times P_z^{b-1} \end{bmatrix}$
4:     $C \leftarrow Q \times P_z \times Q^{-1}$
5:     Return $C$.
6: end procedure
```

Algorithm 3 is used to generate the $m-$companion matrix for a given primitive polynomial P_z. Theorem 10 supports the construction of the matrix Q and the configuration matrix C.

Below is an example demonstrating the working of Algorithm -1 and Algorithm-3 for the case $m = 4, b = 2$.

Example: 4 *The characteristic polynomial of the LFSR is assumed to be $x^8 + x^4 + x^3 + x^2 + 1$, a primitive polynomial of degree 8. Here,*

Ist Iteration: *In this iteration the matrix M is the companion matrix of the primitive polynomial $x^4 + x + 1$.*

$$\begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}_{4\times 4} \xrightarrow{\times M^i} \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix} \mapsto \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix}_{4\times 5}$$

2nd Iteration: *In this iteration the matrix M is the companion matrix of the primitive polynomial $x^5 + x^2 + 1$.*

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{pmatrix}_{4\times 5} \xrightarrow{\times M^i} \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}_{4\times 6}$$

3rd Iteration:*In this iteration the matrix M is the companion matrix of the primitive polynomial $x^6 + x + 1$.*

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \end{pmatrix} \xrightarrow{\times M^i} \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

4th Iteration:*In this iteration the matrix M is the companion matrix of the primitive polynomial $x^7 + x + 1$.*

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} \xrightarrow{\times M^i} \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

(In the above iterations, $A \xrightarrow{\times M^i}$ means multiplying all the rows of the matrix A with the matrix M^i, and the colour red indicates that the number has been randomly sampled). At the end of the above iterations, we get the following matrix Y.

$$Y = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}_{4\times 4}$$

The corresponding matrix Q is as follows.

$$Q = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}_{8\times 8}$$

The following matrix P is the companion matrix of the primitive polynomial $x^8 + x^4 + x^3 + x^2 + 1$.

$$P = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}_{8\times 8}$$

This results in the following matrix C

$$C = Q * P * Q^{-1} = \left(\begin{array}{cccc|cccc} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}\right)_{8\times 8}$$

$$C = \left(\begin{array}{c|c} 0 & I \\ \hline B_0 & B_1 \end{array}\right)$$

In the above example, B_0 and B_1 are the two gain matrices computed using Algorithm 1,3.

Note that in every iteration of Step 3 in Algorithm 1, $m - 1$ random numbers are appended to the rows of the matrix Y. Some of these numbers are derived from the secret key in the proposed scheme. Consequently, the derived feedback configuration depends on the secret key. We now proceed to look at this configuration in detail.

To create a keystream generator from the proposed σ-LFSR configuration, it can be connected to a Finite state machine, which introduces non-linearity. Figure 3.1 shows the schematic of the proposed scheme and its interconnection with an FSM. The scheme has an initialization phase wherein the feedback configuration of the σ-LFSR is calculated by running Algorithm 1. To reduce the time taken for initialization, Algorithm 1 is precomputed till k iterations of step 3 and the resulting matrix Y is made public. The number k can be chosen depending on the computational capacity of the machine that hosts the σ-LFSR. The feedback configuration is calculated by running the remaining part of the algorithm in the initialization phase. In this

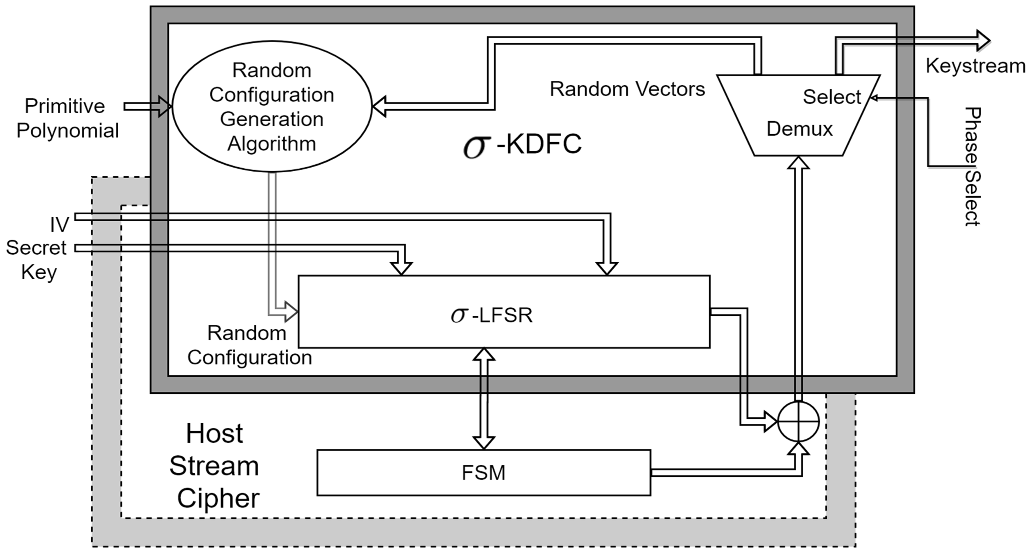


Fig. 3.1: The Schematic of σ-KDFC

phase, there is no keystream generated at the output. The following subsection explains the initialization phase in detail.

3.1.1 The Initialization Phase

During the initialization phase, the σ-LFSR has a publicly known feedback configuration. Further, the pre-calculated matrix $Y \in \mathbb{F}_2^{m\times(m+k)}$, and the primitive polynomials $p_{m+k+1}(x), p_{m+k+2}(x), \ldots, p_{mb}(x)$ are also publicly known. The initial state of the σ-LFSR is derived from the secret key and the IV. (as is normally done in word-based stream ciphers like SNOW). The σ-LFSR is run along with the FSM for $mb - m - k$ clock cycles. This generates $mb - m - k$ vectors in $\mathbb{F}_2^m$. This corresponds to the $mb - m - k$ remaining iterations in Algorithm-1.

The remaining part of Algorithm 1 is now run. In each iteration, the binary numbers appended to the rows of the matrix Y are the entries of the corresponding vector. More precisely, in the i-th iteration of Step 3, for $t \neq i + k \mod m$, the t-th row of Y is appended with the t-th entry of the i-th vector that was generated.

The feedback gains of the σ-LFSR are now set according to the configuration matrix generated by Algorithm 3.

Once the feedback gains are set, the σ-LFSR is run along with the FSM. The first b vectors are discarded, and the keystream starts from the $b + 1$-th vector. The reason for doing this is that the initial state of the σ-LFSR with the new configuration is generated by the publicly known feedback configuration used in the initialization process.

The algorithm for generating the configuration matrix can be applied for all values of m and b. Therefore, the above-described KDFC scheme can be used with any existing word-based stream cipher, irrespective of the size and number of delay blocks.

Time Complexity of the Initialization Phase

Algorithm-**??** involves solving a system of linear equations in less than mb variables. This can be done with time complexity of $\mathcal{O}((mb)^3)$ using Gaussian elimination. The time complexity of Algorithm-1 has $mb-m-k$ iterations. Therefore, if k is chosen such that $mb-m-k$ is $\mathcal{O}(1)$, then the overall time complexity is $\mathcal{O}((mb)^3)$. In Algorithm-3, the matrix C can be calculated by solving the linear system of equations $CQ = QP_{mb}$ for C. Matrix Inversion in Algorithm-**??** has a time complexity of $\mathcal{O}((mb)^3)$. Thus, the time complexity of the initialization phase is $\mathcal{O}((mb)^4)$.

3.2 Algebraic Analysis of σ-KDFC

The entries of the feedback matrices, $B_0, B_1, \ldots, B_{b-1}$, calculated by the procedure given in the previous section, are functions of the matrix Y generated in Step 3 of Algorithm 1. The entries of Y are, in turn, non-linear functions of the initial state of the σ-LFSR.

Note that the last row of Y is always e_1^n. Let the first $m-1$ rows of Y be $v_1, v_2, \ldots, v_{m-1}$. Let $\mathcal{U}$ be the set of variables denoting these rows' entries. Therefore,

$$B_k(i,j) = f_{k(i,j)}(\mathcal{U}) \text{ for } 0 \leq k \leq b-1 \text{ and } 1 \leq i,j \leq m \tag{3.2}$$

where $f_{k(i,j)}$s are polynomial functions.

The algebraic degree of the configuration matrix, denoted by Θ, is defined as follows

$$\Theta\left(C_{\mathcal{S}}\right) = \max_{k,i,j}\left(|f_{k(i,j)}(\mathcal{U})|\right) \tag{3.3}$$

Θ can be considered a measure of the algebraic resistance of σ-KDFC. We now proceed to find a lower bound for Θ.

The matrix Q generated in Step 4 of Algorithm 1 is given as follows

$$Q = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{m-1} \\ e_1^n \\ v_1 P_{mb} \\ \vdots \\ v_{m-1} P_{mb} \\ e_1^n P_{mb} \\ \vdots \\ v_1 P_{mb}^{b-1} \\ \vdots \\ v_{m-1} P_{mb}^{b-1} \\ e_1 P_{mb}^{b-1} \end{bmatrix} \tag{3.4}$$

where P_{mb} is the companion matrix of the publicly known primitive characteristic polynomial of the σ-LFSR. The configuration matrix C is generated by the formula $C = Q \times P_{mb} \times Q^{-1}$. Since Q is an invertible boolean matrix, the determinant of Q is always 1. Therefore, $Q^{-1} = Q^{(a)}$ where $Q^{(a)}$ is the adjugate of Q. Moreover, since the elements of Q belong to $\mathbb{F}_2$, the co-factors are equal to the minors of Q. The rows of Q can be permuted to get the following matrix Q_P

$$Q^P = \begin{bmatrix} e_1^n \\ e_1^n P_{mb} \\ \vdots \\ e_1 P_{mb}^{b-1} \\ v_1 \\ v_1 P_{mb} \\ \vdots \\ v_1 P_{mb}^{b-1} \\ \vdots \\ v_{m-1} \\ v_{m-1} P_{mb} \\ \vdots \\ v_{m-1} P_{mb}^{b-1} \end{bmatrix} \tag{3.5}$$

The matrix Q_P can be decomposed as follows into four sub-matrices Q_1, Q_2, Q_3 and Q_4:

$$Q_P = \left[\begin{array}{c|c} Q_1 & Q_2 \\ \hline Q_3 & Q_4 \end{array}\right] \tag{3.6}$$

where $Q_1 \in \mathbb{F}_2^{b\times(mb-b)}$ is the all zero matrix and the matrices $Q_2 \in \mathbb{F}_2^{b\times b}, Q_3 \in \mathbb{F}_2^{mb-b\times mb-b}$ and $Q_4 \in \mathbb{F}_2^{mb-b\times b}$ are as follows

$$Q_2 = \begin{pmatrix} 0 & \cdots & 0 & 1 \\ 0 & \cdots & 1 & * \\ \vdots & \vdots & & \\ 1 & \cdots & * & * \end{pmatrix} \tag{3.7}$$

$$Q_3 = \begin{pmatrix} v_{1,1} & v_{1,2} & \cdots & v_{1,mb-b} \\ v_{1,2} & v_{1,3} & \cdots & v_{1,mb-b+1} \\ \vdots & \vdots & \cdots & \\ v_{1,b} & v_{1,b+1} & \cdots & v_{1,mb} \\ v_{2,1} & v_{2,2} & \cdots & v_{2,mb-b} \\ v_{2,2} & v_{2,3} & \cdots & v_{2,mb-b+1} \\ \vdots & \vdots & \cdots & \vdots \\ v_{2,b} & v_{2,b+1} & \cdots & v_{2,mb} \\ \vdots & \vdots & \cdots & \vdots \\ v_{m-1,1} & v_{m-1,2} & \cdots & v_{m-1,mb-b} \\ v_{m-1,2} & v_{m-1,3} & \cdots & v_{m-1,mb-b+1} \\ \vdots & \vdots & \cdots & \vdots \\ v_{m-1,b} & v_{m-1,b+1} & \cdots & v_{m-1,mb} \end{pmatrix} \tag{3.8}$$

$$Q_4 = \begin{pmatrix} v_{1,mb-b+1} & \cdots & v_{1,mb-1} & v_{1,mb} \\ v_{1,mb-b+2} & \cdots & v_{1,mb} & * \\ \vdots & \vdots & \cdots\vdots & \\ v_{1,mb+1} & \cdots & * & * \\ v_{2,mb-b+1} & \cdots & v_{2,mb-1} & v_{2,mb} \\ v_{2,mb-b+2} & \cdots & v_{2,mb} & * \\ \vdots & \vdots & \cdots & \vdots \\ v_{2,mb+1} & \cdots & * & * \\ \vdots & \vdots & \cdots & \vdots \\ v_{m-1,mb-b+1} & \cdots & v_{m-1,mb-1} & v_{m-1,mb} \\ v_{m-1,mb-b+2} & \cdots & v_{m-1,mb} & * \\ \vdots & \vdots & \cdots & \vdots \\ v_{1,mb+1} & \cdots & * & * \end{pmatrix} \tag{3.9}$$

Where $*$s are linear combinations of the entries of the previous row. Note that Q^{-1} can be got by permuting the rows of Q_P^{-1}. Since Q_P is invertible, $det(Q_P) = det(Q_3) = 1$.

Let Γ_k be the set of polynomial functions of $\mathcal{U}$ variables with degree k. We now proceed to analyse some of the minors of Q_P.

Lemma: 15 *For* $1 \leq j \leq (mb - b)$, $\mu(Q_P[b, j])) \in \Gamma_{mb-b}$

Proof: For two matrices A and B with the same number of rows, let $[AB]_{p,q}$ be the matrix which is got by removing the p^{th} column from A and appending the q^{th} column of B to A. For $i = b$ and $1 \leq j \leq (mb - b)$, $\mu(Q_P[i, j]))$ is given by:

$$\mu(Q_P[i, j]) = det([Q_3Q_4]_{j,1}) \tag{3.10}$$

Recall that, for a binary matrix $M \in \mathbb{F}_2^{mb\times mb}$, its determinant is given by the following formula,

$$det(M) = \sum_{f\in S_{mb}} \prod_{1\leq i\leq n} M(i, f(i)) \tag{3.11}$$

where S_{mb} is the set of permutations on $(1, 2, \ldots, mb)$. Observe that the diagonal elements of $([Q_3Q_4])_{j,1}$ are distinct $v_{i,k}$s. Their product corresponds to the identity permutation in the determinant expansion formula for $[Q_3Q_4])_{j,1}$. The resultant monomial has degree $mb - b$. Further, this monomial will not occur due to any other permutation. Hence $det([Q_3Q_4]_{j,1})$ is always a polynomial of degree $mb - b$. □

Lemma: 16 *If $1 \leq i \leq b$ then*

$$\mu(Q_P[i,j]) = \begin{cases} det(Q_3) & i+j = mb+1 \\ 0 & i+j >= mb+1 \end{cases} \tag{3.12}$$

Proof: Observe that, for $1 \leq i \leq b$ and $i+j = mb+1$, the $Q_P[i,j]$s are the anti-diagonal elements of Q_2. The minors of these elements are all equal to the determinant of Q_3. As we have already seen, the invertibility of Q_P implies that this determinant is always 1. Therefore, $\mu(Q_P[i,j]) = 1$ when $i+j = mb+1$

Note that, for $1 \leq i \leq b$ and $i+j > mb+1$, the $Q_P[i,j]$s elements of Q_2 that are below the anti-diagonal. If the row and column corresponding to such an element are removed from Q_P, then the first $b-1$ rows of the resulting matrix are always rank deficient. Therefore, the determinant of this matrix is always 0. Therefore, $\mu(Q_P[i,j]) = 0$. □

Lemma: 17 *If $b+1 \leq i \leq mb$ and $1 \leq j \leq n-b$, then $\mu(Q_P[i,j]) \in \Gamma_{mb-b-1}$.*

Proof: Observe that the elements of Q_P considered in this lemma are elements of the sub-matrix Q_3. Therefore, $\mu(Q_P[i,j])$, for the range of i and j considered, is nothing but the determinant of the sub-matrix of Q_3 got by deleting the i^{th} row and j^{th} column of Q_3. The diagonal elements of such a sub-matrix are distinct $v_{i,j}s$. Their product will result in a monomial of degree $mb-b-1$. This corresponds to the identity permutation in the determinant expansion formula given by Equation 3.10. Observe that no other permutation generates this monomial. Hence, the minor will always have a monomial of degree $mb-b-1$. Therefore, $\mu(Q_P[i,j] \in \Gamma_{mb-b-1}$. □

Lemma: 18 *If $b+1 \leq i \leq n$ and $mb-b+1 \leq j \leq mb$, then $\mu(Q_P[i,j]) = 0$.*

Proof: The elements of Q_P considered in this lemma are elements of the submatrix Q_4. Whenever the row and column corresponding to such an element are removed from Q_P, the rows of the submatrix Q_2 become linearly dependent. Therefore, the first b rows of the resultant matrix are always rank deficient. Consequently, $\mu(Q_P[i,j]) = 0$.

□

For a given matrix A with polynomial entries, let $\Theta(A)$ be the maximum degree among all the entries of A. As there are $mb-b$ rows in Q_P with variable entries, $\Theta(Q_P^{-1}) \leq mb-b$. Therefore, we get the following as a consequence of Lemma 15.

$$\Theta(Q^{-1}) = \Theta(Q_P^{-1}) = mb-b \tag{3.13}$$

Recall that the configuration matrix $\mathcal{C}$ is given by $QP_{mb}Q^{-1}$. We now use the above-developed machinery to calculate $\Theta(\mathcal{C})$.

Theorem: 19 $\Theta(\mathcal{C}) \geq mb - b$.

Proof: Observe that the gain matrices $B_0, B_1 \cdots, B_{b-1}$ appear in the last m rows of $C_{\mathcal{S}}$. These rows are generated by multiplying the last m rows QP_{mb} with Q^{-1}. The last m rows of QP_{mb} are as follows

$$\begin{pmatrix} 0 & 0 & \cdots & 1 & * & \cdots & * & * \\ v_{1,b+1} & v_{1,b+2} & \cdots & v_{1,mb} & * & \cdots & * & * \\ v_{2,b+1} & v_{2,b+2} & \cdots & v_{2,mb} & * & \cdots & * & * \\ \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ v_{m-1,b+1} & v_{m-1,b+2} & \cdots & v_{m-1,mb} & * & \cdots & * & * \end{pmatrix} \tag{3.14}$$

The element $C[mb-m+1, mb-m+1]$ is got by multiplying the $(mb-m+1)$-th row of QP_{mb} with the $(mb-m+1)$-th column of Q^{-1}. Note that the $(mb-m+1)$-th column of Q^{-1} is equal to the b-th column of Q_P^{-1}. This column has the following form due to Lemma's 15 and 16.

$$Q^{-1}[:, mb-m+1] = (P_1, P_2, \cdots, P_{mb-b}, 1, 0, \cdots, 0)^T \tag{3.15}$$

where $P_1, P_2, \cdots, P_{mb-b} \in \Gamma(mb-b)$. Therefore,

$$\begin{aligned} C[mb-b+1, mb-b+1] &= (\underbrace{0, 0, \cdots, 1}_{(mb-b)\text{ entries}}, *, \cdots, *, *) \\ &\times (P_1, P_2, \ldots, P_{mb-b}, 1, 0, \cdots, 0)^T \\ &= P_{mb-b} \end{aligned}$$

Hence, it is proved that $\Theta(C) \geq mb - b$. $\square$

Example: 5 *Consider a primitive $\sigma-$LFSR with 4, 2-input 2-output delay blocks i.e. $m = 2$ and $b = 4$. Therefore $n = mb = 8$. The primitive polynomial for the companion matrix P_z is $f(x) = x^8 + x^4 + x^3 + x^2 + 1$. The corresponding matrix Q_P has the following submatrices.*

$$Q_1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{3.16}$$

$$Q_2 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \tag{3.17}$$

$$Q_3 = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ x_2 & x_3 & x_4 & x_5 \\ x_3 & x_4 & x_5 & x_6 \\ x_4 & x_5 & x_6 & x_7 \end{pmatrix} \tag{3.18}$$

$$Q_4 = \begin{pmatrix} x_5 & x_6 & x_7 & x_8 \\ x_6 & x_7 & x_8 & x_1 + x_3 + x_4 + x_5 \\ x_7 & x_8 & x_1 + x_3 + x_4 + x_5 & x_2 + x_4 + x_5 + x_6 \\ x_8 & x_1 + x_3 + x_4 + x_5 & x_2 + x_4 + x_5 + x_6 & x_3 + x_5 + x_6 + x_7 \end{pmatrix} \tag{3.19}$$

Therefore,

$$Q_P^{-1}[:,4] = [Q^{-1}[:,7] = (P_1, P_2, P_3, P_4, 1, 0, 0, 0)$$

where

$P_1 :\ x_2x_4x_6x_8 + x_2x_4x_7 + x_2x_5x_8 + x_2x_6 + x_3x_6x_8 + x_3x_7 + x_4x_5x_6 + x_4x_6 + x_4x_8 + x_5.$

$P_2 :\ x_1x_4x_6x_8 + x_1x_4x_7 + x_1x_5x_8 + x_1x_6 + x_2x_3x_6x_8 + x_2x_3x_7 + x_2x_4x_5x_8 + x_2x_4x_6x_7 + x_2x_5x_6 + x_2x_5x_7 + x_3x_4x_8 + x_3x_5x_6 + x_3x_5x_8 + x_3x_6x_7 + x_4x_5 + x_4x_7$

$P_3 :\ x_1x_3x_6x_8 + x_1x_3x_7 + x_1x_4x_5x_8 + x_1x_4x_6x_7 + x_1x_5x_6 + x_1x_5x_7 + x_2x_3x_5x_8 + x_2x_3x_6x_7 + x_2x_4x_5x_7 + x_2x_4x_6 + x_2x_4x_8 + x_2x_5x_6 + x_2x_6x_8 + x_2x_7 + x_3x_4x_7 + x_3x_4x_8 + x_3x_5x_7 + x_3x_5 + x_4x_5 + x_4x_6$

$P_4 :\ x_1x_3x_5x_8 + x_1x_3x_6x_7 + x_1x_4x_5x_7 + x_1x_4x_6 + x_1x_4x_8 + x_1x_5x_6 + x_2x_3x_5x_7 + x_2x_3x_6 + x_2x_4x_7 + x_2x_5x_8 + x_2x_5 + x_2x_6x_7 + x_3x_4x_6 + x_3x_4x_7 + x_3x_8 + x_4x_5$

Here, $C_{\mathcal{S}}[7,7]$, is equal to P_4 which is a polynomial of degree 4.

3.3 Integration with SNOW 2.0

In this subsection, we refer to SNOW 2.0, which is already discussed in the second chapter and then discuss as a case study to show how σ-KDFC can be applied to an LFSR-based cipher stream. We refer to the resulting cipher as KDFC-SNOW.

3.3.1 KDFC-SNOW:

In the proposed modification, we replace the LFSR part of SNOW 2.0 by a σ-LFSR having 16, 32-input 32-output delay blocks. The configuration matrix of the σ-LFSR is generated using Algorithm 1. We shall refer to the modified scheme, shown in Figure 3.2, as KDFC-SNOW.

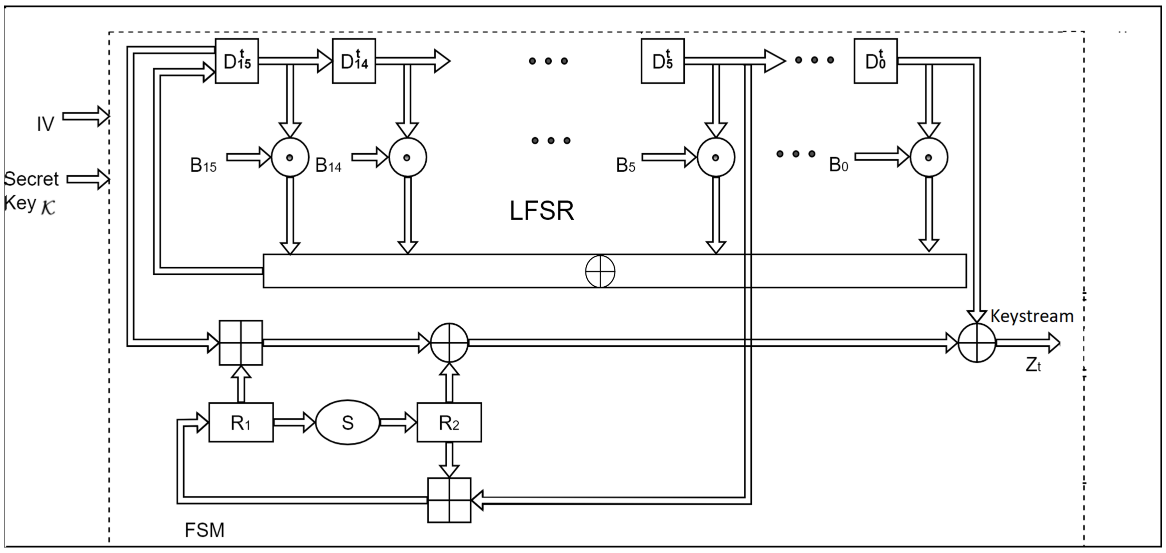


Fig. 3.2: The block diagram KDFC-SNOW

During initialization, the feedback function of the σ-LFSR is identical to that of SNOW-2. As in SNOW 2.0, the σ-LFSR is initialized using a 128-bit IV and a 128/256-bit secret key K. KDFC-SNOW is run with this configuration for 32 clock cycles without producing any symbols at the output. The vectors generated in the last 12 of these clock cycles are used in Algorithm-1 to generate a new feedback configuration. As we have already mentioned, some of the iterations of Algorithm 1 are pre-calculated, and the remaining ones are done as a part of the initialization process. In this case, it is assumed that 468 of these iterations are pre-calculated, and the last 12 iterations are carried out in the initialization process. This calculated configuration replaces the original one, and the resulting setup generates the keystream.

3.3.2 Initialization of KDFC-SNOW

- The delay blocks $D_0, \cdots, D_{15}$ are initialized using the 128/256 bit secret key K and a 128 bit IV in exactly the same manner as SNOW 2.0. The registers R_1 and R_2 are set to zero.

- The initial feedback configuration of the σ-LFSR is identical to SNOW 2.0. This is done by setting B_{11} and B_0 as matrices that represent multiplication by α^{-1} and α respectively. Further, B_2 is set to identity. The other gain matrices are set to zero.

- KDFC-SNOW is run in this configuration for 32 clock cycles without making the output externally available. The last 12 values of F^t are used as the random numbers in Algorithm 1.

- A new configuration matrix is calculated using Algorithms-1, and the corresponding feedback configuration replaces the original one. The scheme is now run with this configuration. The first 32 vectors are discarded, and the key stream starts from the 33^{rd} vector.

3.3.3 Governing Equations of KDFC-SNOW

Let $D_i^t \in \mathbb{F}_{2^{32}}$ denote the value stored in the i^{th} delay block at the t-th time instant after the key stream generation has started. The outputs of the delay blocks of the σ-LFSR satisfy the following equation:

$$D_{15}^{t+1} = B_0 D_0^t + B_1 D_1^t + \cdots + B_{15} D_{15}^t \tag{3.20}$$

Therefore,

$$D_k^{t+1} = \begin{cases} D_{k+t+1}^0 & 0 \leq k+t+1 \leq 15 \\ \sum_{i=0}^{15} B_i D_i^{t+k} & k+t+1 > 15 \end{cases} \tag{3.21}$$

Let F_t be the output of the FSM at time t. The following equation gives the value of the keystream at the t-th time instant

$$F_t = z^t + D_0^t = (D_{15}^t \boxplus R_1^t) + R_2^t \tag{3.22}$$

The registers are updated as follows:

$$R_1^{t+1} = D_5^t \boxplus R_2^t \tag{3.23}$$

$$R_2^{t+1} = S(R_1^t) \tag{3.24}$$

$$z^t = R_1^t \boxplus D_{15}^t + R_2^t + D_0^t = (R_2^{t-1} \boxplus D_4^t) \boxplus D_{15}^t + R_2^t + D_0^t \tag{3.25}$$

where R_1^t and R_2^t represent the values of registers R_1 and R_2 at time instant t. The operation "$\boxplus$" is defined as follows:

$$x \boxplus y = (x + y) \mod 2^{32} \tag{3.26}$$

The challenge for an adversary in this scheme is to find the gain matrices $\{B_0, B_1, \cdots, B_{15}\}$ in addition to the initial state $\{D_0^0, \cdots, D_{15}^0\}$.

Note that Equations 3.22 to 3.25 are got from the FSM. Since the FSM part of the keystream generator is identical for SNOW 2.0 and KDFC-SNOW, these equations are identical for both schemes.

3.3.4 Security enhancement due to KDFC-SNOW

In this section, we briefly describe the resistance of KDFC-SNOW against various kinds of attacks like algebraic attacks, distinguishing attacks, fast correlation attacks, guess and determining Attacks, and cache timing attacks.

Algebraic Attack:

We first briefly describe the Algebraic attack on SNOW 2 described in [30] and demonstrate why this attack becomes difficult with KDFC-SNOW. This attack first attempts to break a modified scheme version where the $\boxplus$ operator is approximated by $\oplus$. The state of LFSR and the value of the registers at the end of the 32 initialization cycles are considered unknown variables. This accounts for a total of $512 + 32 = 544$ unknown variables. The algebraic degree of each S-box(S) equation (156 linearly independent quadratic equations in each clock cycle) is 2. Rearranging the terms in Equation 3.25, we get the following

$$R_2^t = (R_2^{t-1} \boxplus D_4^t) \boxplus D_{15}^t + D_0^t + z^t. \tag{3.27}$$

Note that $R_1^0 = R_2^0 + z^0 + D_0^0 + D_{15}^0$.Therefore, by approximating $\boxplus$ as $\oplus$, Equation 3.27 expands to the following:

$$R_2^t = R_2^0 + \sum_{i=0}^{t} z^i + \sum_{i=0}^{t} (D_4^i + D_{15}^i + D_0^i) \tag{3.28}$$

Further, Equation 3.24 can be expanded as follows:

$$R_2^{t+1} = S(R_1^t) = S(R_2^t + z^t + D_{15}^t + D_0^t) \tag{3.29}$$

In Equation 3.29, the outputs of the delay blocks can be related to the initial state of the LFSR using the following equation.

$$D_k^{t+1} = \begin{cases} D_{k+t+1}^0 & 0 \leq k+t+1 \leq 15 \\ \alpha^{-1} D_{11}^{k+t} + D_2^{k+t} + \alpha D_0^{k+t} & k+t+1 > 15 \end{cases} \tag{3.30}$$

Because of the nature of the S-Box, Equation 3.29 gives rise to 156 quadratic equations per time instant ([30]). When these equations are linearized, the number of variables increases to $\sum_{i=0}^{2} \binom{544}{i} \approx 2^{17}$. Therefore, with $2^{17}/156 \approx 951$ samples, we get a system of equations, which can be solved in (2^{51}) time, to obtain the initial state of the LFSR and the registers. This attack is then modified to consider the $\boxplus$ operator. This attack has a time complexity of approximately (2^{294}).

The feedback equation is no longer known when a *sigma*-LFSR replaces the LFSR in SNOW 2.0. If the entries of the feedback gain matrices are considered as unknowns, then there are a total of $16 * m^2 + mb + m = 16928$ unknown variables (This includes the $16 * m^2$ entries of the feedback matrices and $mb + m$ entries corresponding to the state of the LFSR and the register R_2 at the beginning of the keystream). The output of the delay blocks at any given instant are functions of these variables. Here, Equation 3.30 is replaced by Equation 4.3. Now, if the outputs of the delay blocks in Equation 3.29 are linked to the initial state of the LFSR (i.e. the state when the key stream begins) using Equation 4.3 instead of Equation 3.30, then the resulting equations contain the feedback matrices and their products. For example, in the expressions for R_2^2 and R_2^3, D_{15}^1 and D_{15}^2 are given as follows

$$\begin{aligned}
D_{15}^1 &= B_0D_0^0 + B_1D_1^0 + \ldots + B_{15}D_{15}^0 \\
D_{15}^2 &= B_0D_0^1 + B_1D_1^1 + \ldots + B_{15}D_{15}^1 \\
&= B_0D_1^0 + B_1D_2^0 + \ldots \\
&+ B_{15}(B_0D_0^0 + B_1D_1^0 + \ldots + B_{15}D_{15}^0) \\
&= B_{15}B_0D_0^0 + (B_{15}B_1 + B_0)D_1^0 + \ldots \\
&+ (B_{15}B_{15} + B_{14})D_{15}^0
\end{aligned}$$

While D_{15}^1 is a polynomial of degree two in the unknown variables, D_{15}^2 is a polynomial of degree 3. Similarly, with each successive iteration, the degree of the expression for D_{15}^t keeps increasing till all the m^2b entries of the feedback matrices are multiplied by each other. A similar thing happens with the expressions for D_0^t. This results in a set of polynomial equations having a maximum degree equal to $m^2b + 1 = 16385$. Therefore, although the equations generated by Equation 3.29 are quadratic in terms of the initial state of the σ-LFSR, they are no longer quadratic in the set of all unknowns. We instead have a system of equations in 16982 variables with a maximum degree of over 16000. Linearizing such a system will give us a system of linear equations in $N = \sum_{i=0}^{M} \binom{16928}{i}$ unknowns where M is higher than 16000. Such an attack is, therefore, not feasible.

One could instead consider the rows of the matrix Y generated by Algorithm 1 as unknowns. Assuming that the first row is e_1^n, the total number of unknowns will now be $31 * 512 = 15872$. As we have already seen, the entries of the feedback matrices (B_is) are polynomials in these variables. From Theorem 19, the maximum degree of these polynomials is at least $mb - b$. Therefore, the maximum degree of the equations generated by Equation 3.29 will be at least $mb - b + 1 = 497$. Therefore, linearizing this system of equations gives rise to a system of linear equations in $N = \sum_{i=0}^{497} \binom{16416}{i} \approx \mathcal{O}(2^{3207})$ unknowns. Therefore, an algebraic attack on this scheme that uses linearization seems unfeasible.

Another approach to the algebraic cryptoanalysis of SNOW 2 is found in [31]. Therein, linear equations are generated by assuming a set of values for the entries of the registers of the FSM. Two methods of cryptoanalysis are presented in this chapter. In the first method, the attacker guesses the values of ten consecutive register entries R_1. The guessed value of R_1^t uniquely determines the value of $= R_2^{t+1}$ (by Equation 3.24). Given values of R_1^t and R_2^t, Equation 3.22 gives rise to two linear equations and thirty quadratic equations. These equations can be generated for nine-time instances. Further, given values of R_1^{t+1} and R_2^t, the value of D_5^t can be uniquely determined. One can thus determine 8 consecutive values of D_5^t. These, along with the linear and quadratic equations, result in an over-determined system of equations. These, when solved, give us an estimate of the current state. Since the relation between the current state and the initial state is linear, one can get an estimate of the initial state from the estimate of the current state. The correctness of the guesses can be verified by using the estimated initial state and the system equations of SNOW 2.0 to regenerate the key stream and check if it matches the actual one.

In the second method, the attacker assumes the following nine consecutive entries of the register R_1

$$R_1^t = 0, R_1^{t+1} = 2^{32} - 1, R_1^{t+2} = 0, \cdots, R_1^{t+8} = 0 \tag{3.31}$$

Now, there are 7 consecutive time instances,k, where the values of R_1^{k+1} and R_2^k are simultaneously known. Hence, at each time instance, Equation 3.23 gives rise to 32 linear equations over $GF(2)$. This accounts for a total of 224 equations. Further, there are 7 consecutive values of k wherein the value of R_1^k is 0, and the value of R_2^k is known. Therefore, Equation 3.22 gives us 32 linear equations over $GF(2)$ at each time instance. This accounts for another 224 linear equation. If the assumption made in this attack holds true, at the $t+1$-th time instance, the value in the register R_1 is zero while the value in the register R_2 is given as follows.

$$R_2^{t+1} = D_5^{t+1} + 1111\cdots 1 \tag{3.32}$$

When these values are substituted in Equation 3.22, we get another 32 linear equations. Recall that, at the $t+2$-th time instance, the value in the register R_1 is $1111\cdots 1$. As a consequence of Theorem 2 in [31], this results in Equation 3.22 generating 32 linear equations satisfied with probability half. This probability becomes 1 when $D_0^2 + z^2 + S(0)$ is zero. We thus have a total of 512 linear equations. When these equations are linearly independent, solving them gives us the state of the LFSR. The correctness of the assumptions is verified by checking if the sequence generated from this state matches the actual keystream.

In both these attacks, to verify the correctness of the assumptions, one has to generate the sequence with the calculated state of the LFSR and check if it matches the actual keystream. To do this, the feedback equation of the LFSR is needed. This verification cannot be done since this is not available in KDFC-SNOW. As a result, KDFC-SNOW is immune to these attacks. For a similar attack to work on KDFC-SNOW, the assumptions should enable the attacker to calculate 512 output words of the LFSR (as against the 32 output words calculated in these attacks). This would mean more assumptions. Consequently, the probability of these assumptions being true will be significantly lower. This will result in a much higher time complexity for the attack.

Distinguishing Attack:

In the distinguishing attack, the attacker aims to distinguish the generated keystream from a random sequence. Distinguishing attacks on SNOW 2.0 have been launched using the linear masking method [32, 33, 34]. This method adapts the linear cryptanalysis method given in [65] to stream ciphers. In this method, the algorithm of the keystream generator is assumed to consist of two parts: a linear one and a non-linear one. In the case of SNOW 2.0, the linear part is the LFSR, and the non-linear part is the FSM. The linear part satisfies a linear recurring relation of the form $f(x_n, x_{n+1}, x_{n+2}, \ldots, x_{n+k}) = 0$ for all n. We then try to find a linear relation, called the masking relation, that the non-linear part approximately satisfies. This relation is of the following form:

$$\sum_{i=0}^{\ell_1} \Gamma_i x_{n+i} = \sum_{i=0}^{\ell_2} \Lambda_i z_{n+i} \tag{3.33}$$

where $z_0, z_1, \ldots$ is the output sequence of the key stream generator. The Γ_is and Λs are linear masks that map the corresponding x_{n+i}s and z_{n+i}s to $\mathbb{F}_2$ respectively. The error in the masking relation can be seen as a random variable. If p is the probability that the non-linear part satisfies Equation 3.33, then $p - \frac{1}{2}$ is called the bias of the masking relation. The Masking relation, along with the linear recurring relation, is used to generate a relation in terms of the elements of the output sequence. The error in this relation can also be seen as a random variable. If the probability of the sequence satisfying this relation is p_f, then $p_f - \frac{1}{2}$ is the bias of this relation. This bias can be related to the bias of the masking relation using the piling up lemma in [65]. The main task in this type of attack is to find masks Γ_is and Γ_i's which maximise the bias of the masking relation. The following linear masking equation is used in [32] and [33] for the FSM of SNOW 2.

$$\begin{aligned}\Gamma_0 x_n + \Gamma_1 x_{n+1} + \Gamma_5 x_{n+5} + \Gamma_{15} x_{n+15} + \Gamma_{16} x_{n+16} &= \\ \Lambda_0 z_n + \Lambda_1 z_{n+1}\end{aligned}$$

In [32], it is assumed that all the Γ_is and Λ_is are equal. In [33] it is assumed that Γ_0, Γ_{15} and Λ_0 are equal to each other. $\Gamma_1, \Gamma_5.\Gamma_{16}$ and Λ_1 are also assumed to be equal. Since $f(x_n, x_{n+1}, \ldots, x_{n+k})$ is a linear relation, the following relation can be written purely in terms of the z_is

$$\begin{aligned}
\Gamma_0 f(x_n, x_{n+1}, \ldots, x_{n+k}) &\ + \\
\Gamma_1 f(x_{n+1}, x_{n+2}, \ldots, x_{n+k+1} &\ + \\
\Gamma_5 f(x_{n+5}, x_{n+6}, \ldots, x_{n+k+5}) &\ + \\
\Gamma_{15} f(x_{n+15}, x_{n+16}, \ldots, x_{n+k+15}) &\ + \\
\Gamma_{16} f(x_{n+16}, x_{n+17}, \ldots, x_{n+k+16}) &= 0
\end{aligned}$$

The linear relation between the elements of the output sequence in both [32] and [33] is obtained using this method. Further, if there are ℓ non-zero coefficients in f, then the random variable corresponding to the error in this relation is a sum of ℓ random variables, each corresponding to the error in the linear masking equation.

The feedback equation is unknown in the proposed σ-LFSR configuration. Therefore, the only known linear recurring relation that the output of the σ-LFSR satisfies is the one defined by its characteristic polynomial. If the characteristic polynomial is assumed to be the same as that of the LFSR in SNOW 2, then the corresponding linear recurring relation has 250 non-zero coefficients. Further, since these coefficients are elements of $\mathbb{F}_2$, the non-zero coefficients are equal to 1. Therefore, as a consequence of the piling up lemma, if the bias of the masking equation is ϵ, then the bias of the relation between the elements of the key stream is given as follows

$$\epsilon_{final} = 2^{249} \times \epsilon^{250} \tag{3.34}$$

The number of elements of the key stream needed to distinguish it from a random sequence is $\frac{1}{\epsilon^2_{final}}$. Therefore, for an identical linear masking equation, the length of the key stream for the distinguishing attack is much higher for the proposed configuration than SNOW 2. This is demonstrated in the following table.

Reference	ϵ	ϵ_{final} SNOW 2.0	ϵ_{final} KDFC SNOW	**#Keystream** SNOW 2.0	**#Keystream** KDFC-SNOW
[32]	$2^{-27.61}$	$2^{-112.25}$	$2^{-6653.5}$	2^{225}	2^{13307}
[33]	$2^{-15.496}$	$2^{-86.9}$	2^{-3625}	2^{174}	2^{7250}

Table 3.1: Comparison of Distinguishing Attack Result for SNOW 2.0 and KDFC SNOW

Fast Correlation Attack:

The Fast Correlation Attack is a commonly used technique for the cryptanalysis of LFSR-based stream ciphers. This method was first introduced for bitwise keystreams in ([59]). Here, the attacker views windows of the key stream as noisy linear encodings of the initial state of the LFSR. She then tries to recover the initial state by decoding this window. Further, linear combinations of elements in this window can be seen as encodings of subsets of the initial state. This results in smaller codes which are more efficient to decode [61]. The linear recurring relation, satisfied by the output of the LFSR, is used to generate the parity check matrix for this code. A Fast correlation attack for word-based stream ciphers was first described in [72]. An improvement on this attack is given in [34]. Both these schemes consider a linear recurring relation with coefficients in $\mathbb{F}_2$. For SNOW 2.0, this relation has order 512. This is equivalent to considering each component sequence generated by a conventional bitwise LFSR with the same characteristic polynomial as the LFSR in SNOW 2.0. The time complexity of the attack given in [34] is $2^{212.38}$. The scheme in [35] considers the LFSR in SNOW 2.0 to be over $\mathbb{F}_{2^8}$. This results in a linear recurring relation of order 64. Further, it utilizes the $k-$ tree Algorithm given in [62] to generate parity check equations. This results in a significant improvement in the time complexity of the attack. The time complexity of this attack is $2^{164.5}$, which is around 2^{49} times better than that of the attack given in [34]. However, to derive the linear recurring relation over $\mathbb{F}_{2^8}$, the knowledge of the feedback function is critical. In KDFC SNOW, the characteristic polynomial of the σ-LFSR is publicly known. The attacker can, therefore, generate a linear recurring relation over $\mathbb{F}_2$ that the output of the σ-LFSR satisfies. Therefore, the attack in [34] will also be effective against KDFC-SNOW. However, without the knowledge of the feedback function, the attacker cannot derive a linear recurring relation over $\mathbb{F}_{2^8}$. Hence, KDFC-SNOW resists the attack given in [35].

[70] uses MILP(Mixed Integer Linear Programming) to find a linear mask that gives better correlation. This results in an attack with a time complexity of $2^{162.91}$, which is $2^{1.59}$ times better than [35]. [1] further modifies this attack using a small trick in $k-$ tree algorithm. The time complexity with this modification turns out to be $2^{162.86}$. These attacks consider a feedback polynomial of degree 512 over $\mathbb{F}_2$. Therefore, KDFC-SNOW does not provide any extra security against these attacks.

Guess and Determine Attack

The guess and determine attack given in [66] can be applied on SNOW 2.0 as follows;

Consider the following equations, which are satisfied by SNOW 2.0

$$D_t^{16} = \alpha^{-1}D_t^{11} + D_t^2 + \alpha D_t^0 \tag{3.35}$$

$$R_1^t = D_t^4 + S(R_1^{t-2}) \tag{3.36}$$

$$z_t = D_t^0 + (D_t^{15} + R_1^t) + S(R_1^{t-1}) \tag{3.37}$$

These equations are used to generate the following tables The entries in the above tables

0	2	11	16
1	3	12	17
$\vdots$	$\vdots$	$\vdots$	$\vdots$
18	20	29	34

4	35	37
5	36	38
$\vdots$	$\vdots$	$\vdots$
22	53	55

0	15	36	37
1	16	37	38
$\vdots$	$\vdots$	$\vdots$	$\vdots$
18	33	54	55

Table 3.2: Index table for SNOW 2.0

correspond to the variables that are to be estimated. The entries in the first table, i.e. 0 to 34, correspond to 35 consecutive outputs of the LFSR. The entries 35 to 55 correspond to 21 consecutive entries of Register $R1$. Each row of the above tables corresponds to the values of the delay blocks and registers in Equations 3.35, 3.36 and 3.37 at a particular time instant.

Consider a multi-stage graph with 56 nodes in each stage corresponding to the 56 entries in the above tables. Each node is connected to all the nodes in the next stage, giving rise to a trellis diagram. An entry is said to be eliminated by a path if, knowing the values of the entries corresponding to the nodes in the path, the value of that entry can be calculated.

Now, recursively calculate an optimal path that eliminates all the entries. The desired basis corresponds to the nodes in this path. In the i-th iteration of this algorithm, calculate the optimal path of length i to each node in the i-th stage. To find the optimal path to the k-th node, consider all the incoming edges of node k. By appending the node k to the optimal paths of length $i-1$ ending at the source nodes of these edges, we get 55 paths of length i. Choose the edge corresponding to the path that eliminates the most variables. In case of a tie, consider the path that results in the most rows with two unknowns. This process continues until a path that eliminates all the entries is found. This algorithm results in a basis of cardinality 8 for SNOW 2.0.

In KDFC-SNOW, the feedback equation of the σ-LFSR is not known. The smallest known linear recurring relation that the output of the σ-LFSR satisfies is the relation corresponding to its characteristic polynomial. This relation is given as follows.

$$x_{n+512} = x_{n+510} + x_{n+504} + x_{n+502} + x_{n+501} + x_{n+494} + x_{n+493} + x_{n+490} + x_{n+486} + x_{n+485} + x_{n+483} + x_{n+481} + x_{n+480} + x_{n+478} + x_{n+477} + x_{n+471} + x_{n+470} + x_{n+469} + x_{n+466} + x_{n+462} + x_{n+461} + x_{n+459} + x_{n+458} + x_{n+452} + x_{n+449} + x_{n+446} + x_{n+445} + x_{n+444} + x_{n+441} + x_{n+438} + x_{n+437} + x_{n+434} + x_{n+433} + x_{n+432} + x_{n+431} + x_{n+429} + x_{n+427} + x_{n+424} + x_{n+423} + x_{n+420} + x_{n+419} + x_{n+414} + x_{n+412} + x_{n+411} + x_{n+409} + x_{n+405} + x_{n+402} + x_{n+400} + x_{n+399} + x_{n+398} + x_{n+396} + x_{n+395} + x_{n+393} + x_{n+392} + x_{n+390} + x_{n+388} + x_{n+387} + x_{n+385} + x_{n+375} + x_{n+374} + x_{n+372} + x_{n+371} + x_{n+366} + x_{n+365} + x_{n+363} + x_{n+362} + x_{n+359} + x_{n+357} + x_{n+356} + x_{n+355} + x_{n+354} + x_{n+353} + x_{n+352} + x_{n+351} + x_{n+350} + x_{n+347} + x_{n+345} + x_{n+344} + x_{n+343} + x_{n+341} + x_{n+339} + x_{n+338} + x_{n+337} + x_{n+336} + x_{n+333} + x_{n+330} + x_{n+329} + x_{n+326} + x_{n+324} + x_{n+322} + x_{n+319} + x_{n+310} + x_{n+307} + x_{n+306} + x_{n+305} + x_{n+304} + x_{n+303} + x_{n+301} + x_{n+299} + x_{n+298} + x_{n+297} + x_{n+296} + x_{n+295} + x_{n+294} + x_{n+293} + x_{n+292} + x_{n+291} + x_{n+289} + x_{n+286} + x_{n+285} + x_{n+283} + x_{n+282} + x_{n+281} + x_{n+278} + x_{n+276} + x_{n+274} + x_{n+271} + x_{n+269} + x_{n+264} + x_{n+262} +$$

$x_{n+259}+x_{n+258}+x_{n+257}+x_{n+255}+x_{n+253}+x_{n+251}+x_{n+249}+x_{n+248}+x_{n+243}+x_{n+240}+x_{n+239}+$
$x_{n+238}+x_{n+236}+x_{n+235}+x_{n+233}+x_{n+232}+x_{n+230}+x_{n+229}+x_{n+228}+x_{n+227}+x_{n+226}+x_{n+222}+$
$x_{n+217}+x_{n+216}+x_{n+215}+x_{n+214}+x_{n+213}+x_{n+210}+x_{n+208}+x_{n+206}+x_{n+203}+x_{n+201}+x_{n+199}+$
$x_{n+193}+x_{n+190}+x_{n+184}+x_{n+179}+x_{n+178}+x_{n+177}+x_{n+175}+x_{n+174}+x_{n+173}+x_{n+172}+x_{n+171}+$
$x_{n+169}+x_{n+165}+x_{n+164}+x_{n+163}+x_{n+158}+x_{n+156}+x_{n+155}+x_{n+153}+x_{n+152}+x_{n+151}+x_{n+149}+$
$x_{n+147}+x_{n+146}+x_{n+143}+x_{n+141}+x_{n+138}+x_{n+136}+x_{n+132}+x_{n+131}+x_{n+129}+x_{n+128}+x_{n+126}+$
$x_{n+125}+x_{n+124}+x_{n+123}+x_{n+121}+x_{n+120}+x_{n+119}+x_{n+118}+x_{n+117}+x_{n+116}+x_{n+115}+x_{n+113}+$
$x_{n+112}+x_{n+111}+x_{n+109}+x_{n+105}+x_{n+104}+x_{n+103}+x_{n+102}+x_{n+98}+x_{n+97}+x_{n+94}+x_{n+93}+$
$x_{n+89}+x_{n+88}+x_{n+87}+x_{n+81}+x_{n+78}+x_{n+76}+x_{n+75}+x_{n+73}+x_{n+72}+x_{n+70}+x_{n+69}+x_{n+68}+$
$x_{n+67}+x_{n+66}+x_{n+65}+x_{n+63}+x_{n+59}+x_{n+58}+x_{n+57}+x_{n+56}+x_{n+55}+x_{n+53}+x_{n+51}+x_{n+50}+$
$x_{n+49}+x_{n+47}+x_{n+46}+x_{n+45}+x_{n+44}+x_{n+41}+x_{n+39}+x_{n+37}+x_{n+36}+x_{n+33}+x_{n+30}+x_{n+26}+$
$x_{n+25}+x_{n+21}+x_{n+20}+x_{n+19}+x_{n+16}+x_{n+5}+x_0$

As in SNOW 2.0, the following equations are also satisfied,

$$\begin{aligned} R_1^t &= D_4^t + S(R_1^{t-1}) \\ z_n &= D_0^t + (D_{15}^t + R_1^t) + S(R_1^{t-1}) \end{aligned}$$

The following tables can be constructed using these three equations. We ran the Vitterbi-like

512	510	504	502	$\cdots$	5	0
513	511	505	503	$\cdots$	6	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1025	1023	1007	1005	$\cdots$	516	513

Table 3.3: Index table for $f1(x)$

4	1026	1028
5	1027	1029
$\vdots$	$\vdots$	$\vdots$
517	1539	1541

0	15	1027	1028
1	16	1028	1029
$\vdots$	$\vdots$	$\vdots$	$\vdots$
513	528	1540	1541

Table 3.4: Index table for Equation 31 and Equation 32

algorithm with the above tables on a cluster with 40 INTEL(R) XEON(R) CPUs (E5-2630 2.2GHz). The program ran for 16 iterations and generated the path {1041, 17, 15, 13, 28, 16, 11, 14, 9, 1050, 18, 39, 12, 7, 5, 0, 3}. This path has a length of 17. This corresponds to a time complexity of 2^{544}.

Cache Timing Attack

The feedback gains in the proposed KDFC scheme do not correspond to known elements of the finite field. Instead, the feedback gains are matrices which are dependent on the key. Therefore, the feedback equation cannot be calculated by using look-up tables. As a result, the attacker cannot obtain any information from the list of cache accesses as done in [36].

3.3.5 Randomness Test

In this subsection, we evaluate the randomness of the keystream generated by KDFC-SNOW.

Test Methodology

We have used the NIST randomness test suite to evaluate the randomness of a keystream generated by KDFC-SNOW. There are 16 randomness tests in the suite. Each test returns a level of significance, i.e. $P-Value$. If this value is above 0.01 for a given test, then the keystream is considered to be random for that test.

KDFC-SNOW has been implemented using SageMath 8.0. The NIST randomness tests have been conducted on the generated keystream using Python 3.6. The characterestic polynomial of the σ-LFSR has been taken as $f(x) = x^{512}+x^{510}+x^{504}+x^{502}+x^{501}+x^{494}+x^{493}+x^{490}+x^{486}+x^{485}+x^{483}+x^{481}+x^{480}+x^{478}+x^{477}+x^{471}+x^{470}+x^{469}+x^{466}+x^{462}+x^{461}+x^{459}+x^{458}+x^{452}+x^{449}+x^{446}+x^{445}+x^{444}+x^{441}+x^{438}+x^{437}+x^{434}+x^{433}+x^{432}+x^{431}+x^{429}+x^{427}+x^{424}+x^{423}+x^{420}+x^{419}+x^{414}+x^{412}+x^{411}+x^{409}+x^{405}+x^{402}+x^{400}+x^{399}+x^{398}+x^{396}+x^{395}+x^{393}+x^{392}+x^{390}+x^{388}+x^{387}+x^{385}+x^{375}+x^{374}+x^{372}+x^{371}+x^{366}+x^{365}+x^{363}+x^{362}+x^{359}+x^{357}+x^{356}+x^{355}+x^{354}+x^{353}+x^{352}+x^{351}+x^{350}+x^{347}+x^{345}+x^{344}+x^{343}+x^{341}+x^{339}+x^{338}+x^{337}+x^{336}+x^{333}+x^{330}+x^{329}+x^{326}+x^{324}+x^{322}+x^{319}+x^{310}+x^{307}+x^{306}+x^{305}+x^{304}+x^{303}+x^{301}+x^{299}+x^{298}+x^{297}+x^{296}+x^{295}+x^{294}+x^{293}+x^{292}+x^{291}+x^{289}+x^{286}+x^{285}+x^{283}+x^{282}+x^{281}+x^{278}+x^{276}+x^{274}+x^{271}+x^{269}+x^{264}+x^{262}+x^{259}+x^{258}+x^{257}+x^{255}+x^{253}+x^{251}+x^{249}+x^{248}+x^{243}+x^{240}+x^{239}+x^{238}+x^{236}+x^{235}+x^{233}+x^{232}+x^{230}+x^{229}+x^{228}+x^{227}+x^{226}+x^{222}+x^{217}+x^{216}+x^{215}+x^{214}+x^{213}+x^{210}+x^{208}+x^{206}+x^{203}+x^{201}+x^{199}+x^{193}+x^{190}+x^{184}+x^{179}+x^{178}+x^{177}+x^{175}+x^{174}+x^{173}+x^{172}+x^{171}+x^{169}+x^{165}+x^{164}+x^{163}+x^{158}+x^{156}+x^{155}+x^{153}+x^{152}+x^{151}+x^{149}+x^{147}+x^{146}+x^{143}+x^{141}+x^{138}+x^{136}+x^{132}+x^{131}+x^{129}+x^{128}+x^{126}+x^{125}+x^{124}+x^{123}+x^{121}+x^{120}+x^{119}+x^{118}+x^{117}+x^{116}+x^{115}+x^{113}+x^{112}+x^{111}+x^{109}+x^{105}+x^{104}+x^{103}+x^{102}+x^{98}+x^{97}+x^{94}+x^{93}+x^{89}+x^{88}+x^{87}+x^{81}+x^{78}+x^{76}+x^{75}+x^{73}+x^{72}+x^{70}+x^{69}+x^{68}+x^{67}+x^{66}+x^{65}+x^{63}+x^{59}+x^{58}+x^{57}+x^{56}+x^{55}+x^{53}+x^{51}+x^{50}+x^{49}+x^{47}+x^{46}+x^{45}+x^{44}+x^{41}+x^{39}+x^{37}+x^{36}+x^{33}+x^{30}+x^{26}+x^{25}+x^{21}+x^{20}+x^{19}+x^{16}+x^{5}+1$.

(This polynomial is the characteristic polynomial of the LFSR in SNOW 2.0 when it is implemented as a σ-LFSR, i.e. when multiplication by α and α^{-1} are represented by matrices).

The keystream has been generated using the following key (K) and initialization vector (IV).

$$K = [681, 884, 35, 345, 203, 50, 912, 358]$$
$$IV = [645, 473, 798, 506]$$

Test Results

The results obtained from 16 NIST tests are shown in table 3.5.

S.No.	Test	P-Value	Random
01.	Frequency Test (Monobit)	0.35966689490586123	✓
02.	Frequency Test within a Block	0.24374184001729746	✓
03.	Run Test	0.9038184342313019	✓
04.	Longest Run of Ones in a Block	0.5246846287441829	✓
05.	Binary Matrix Rank Test	0.1371167998339736	✓
06.	Discrete Fourier Transform (Spectral) Test	0.1371167998339736	✓
07.	Non-Overlapping Template Matching Test	0.3189818228443801	✓
08.	Overlapping Template Matching Test	0.211350493609367	✓
09.	Maurer's Universal Statistical test	0.4521082097311434	✓
10.	Linear Complexity Test	0.1647939201114819	✓
11.	Serial Test	0.7821664366290292	✓
12.	Approximate Entropy Test	0.880218270580662	✓
13.	Cummulative Sums (Forward) Test	0.34630799549695923	✓
14.	Cummulative Sums (Reverse) Test	0.6633686090204551	✓
15.	Random Excursions Test	0.9697352800599932	✓
16.	Random Excursions Variant Test	0.97387	✓

Table 3.5: NIST Randomness Test

These results are comparable to that of SNOW 2.0([73]). Note that the feedback configuration of SNOW 2.0 is one of the possible feedback configurations in the σ-KDFC scheme.

3.3.6 Challenges in implementation

The feedback function of a σ-LFSR can be implemented in hardware by ANDing the bits at the output of the delay blocks with the corresponding columns of the feedback matrices and then EXORing their respective outputs. A fairly fast combinational circuit can implement this.

The main problem of KDFC lies in its software implementation. Since the feedback function is not fixed, look-up tables cannot be used to implement the σ-LFSR. Further, the choice of the feedback configurations is not restricted to the set of efficiently implementable σ-LFSR configurations as given in ([23]). This makes the implementation of KDFC SNOW extremely challenging.

In our implementation, the state of the σ-LFSR is stored as a set of 32 integers. The i-th integer corresponds to the i-th output of the delay blocks. Calculating the feedback function of the σ-LFSR involves calculating the bitwise XOR of a subset of the columns of the feedback matrices (B_is). To make the implementation more efficient, for all $1 \leq i \leq 32$, the i-th columns of the feedback matrices are stored in adjacent memory locations. Thus, each integer in the state of the σ-LFSR corresponds to a set of columns of the feedback matrices stored in a contiguous memory block. The state vector is now sampled one integer at a time, and the columns of the B_is corresponding to the non-zero bits in these integers are XORed. We then do a bit-wise right shift on each of these integers and introduce the result of the XOR operation bitwise as the most significant bits. In this way, the σ-LFSR can be implemented using bitwise XORs and shifts. The FSM is implemented as in SNOW 2.0 [16]. This implementation takes 25 cycles to generate a single word on an Intel Probook 4440s machine with a 2.8 Ghz i5 processor.

Each iteration of Algorithm-1 involves solving a system of linear equations. This process is time-consuming and contributes to increasing the initialization time. The initialization process was implemented using a C code with open mp (with 3 threads). In this implementation, linear equations were solved using a parallel implementation of the LU decomposition algorithm.

3.4 Summary

In this chapter, we have described using σ-LFSRs with key-dependent feedback configurations in stream ciphers that use word-based LFSRs. In this method, an iterative configuration generation algorithm(CGA) uses key-dependant random numbers to generate a random feedback configuration for the σ-LFSR. We have theoretically analysed the algebraic degree of the resulting feedback configuration. As a test case, we have demonstrated how this scheme can be used along with the Finite State Machine of SNOW 2.0. We have analysed the security of the resulting key-stream generator against various attacks and demonstrated an improvement in security compared to SNOW 2.0. Further, the key streams generated by the proposed method are comparable to SNOW 2.0 from a randomness point of view.

Chapter 4

Application of z-primitive σ-LFSRs to resist Fast Correlation Attacks

In this chapter, we propose a method for hiding both the feedback equation of a word-based LFSR and the characteristic polynomial of the state transition matrix. This is to deny the attacker the knowledge of any linear recurring relation that the LFSR satisfies. This is done to counter Fast Correlation Attacks using linear recurring relations to generate parity check equations. Here, we employ a z-primitive σ-LFSR whose characteristic polynomial is randomly sampled from the distribution of primitive polynomials over $GF(2)$ of the appropriate degree. We propose an algorithm for locating z-primitive σ-LFSR configurations of a given degree. Further, an invertible matrix is generated from the key. This is then employed to create a public parameter to retrieve the feedback configuration using the key. If the key size is n- bits, the process of retrieving the feedback equation from the public parameter has an average time complexity $\mathbb{O}(2^{n-1})$. The proposed method has been tested on SNOW 2.0 and SNOW 3G for resistance to fast correlation attacks.

4.1 Proposed Scheme

In this section, we describe a method of converting the existing feedback configuration of the LFSR into a random z-primitive σ-LFSR configuration. In contrast to the scheme given in Chapter 3, the characteristic polynomial in this scheme is also randomly sampled from the set of primitive polynomials of a given degree. Information about this configuration is embedded in a public parameter. On the decryption side, the configuration is recovered from the public parameter using the secret key. In the proposed scheme, in addition to sharing the secret key and calculating the initial state of the LFSR, the following two steps must be performed before the start of communication.

- Generating a random z-primitive σ-LFSR configuration.

- Generating the public parameter using the state transition matrix of the z-primitive σ-LFSR and the secret key and declaring it to the entire network.

4.1.1 Generation of $z-$primitive σ LFSR Configuration

This subsection aims to generate a random configuration matrix of z-primitive $\sigma-$LFSR with b, m-input m-output delay blocks. The first step in this process is to randomly sample a primitive polynomial f of degree mb. We then calculate a primitive polynomial g with degree m such that if α is the root of f then α^z is the root of g where $z = \frac{2^{mb}-1}{2^m-1}$. After that, we calculate the distance vector of a primitive σ-LFSR with a single m-input m-output delay block having characteristic polynomial g. This distance vector generates an element of z_{mb}^{f}. This is then used to generate a σ-LFSR configuration with characteristic polynomial f. Given a randomly sampled polynomial f of degree mb, this process involves the following algorithms;

- An algorithm to find a primitive polynomial g of degree m.
- An algorithm to generate an element of z_m^g.
- An algorithm to generate the σ-LFSR configuration with characteristic polynomial f from an element of z_m^g.

Algorithm 4 Algorithm to find a primitive polynomial of degree m

Input:

1. Companion matrix, M_f of a randomly sampled primitive polynomial f of degree n over $GF(2)$,

```
1:  procedure $Find\_primitive\_deg\_m(M_f)$
2:      $z \leftarrow \frac{2^n-1}{2^m-1}$
3:      $B \leftarrow M_f^z$
4:      Sample a random vector $v \in F_2^n$.
5:      $i \leftarrow 0$
6:      while do  $i \neq (m-1)$
7:          $v_1 \leftarrow v \times B^i$
8:          $A[i,:] \leftarrow v_1$
9:          $i \leftarrow i+1$
10:     end while
11:     Solve the following equation for $a = (a_0, a_1, \cdots, a_{m-1}) \in F_2^m$.
                        $a.A = v \times B^m$
12: end procedure
13: Output:
        1. A primitive polynomial $g = a_0 + a_1 * x + a_2 * x^2 + \cdots + a_{m-1} * x^{m-1} + x^m$.
```

In the above algorithm, the matrix M_f represents the root of f. The above algorithm calculates the minimal polynomial of the matrix M_f^z, which, according to Lemma 1, is a primitive polynomial of degree m.

We now proceed to generate a random element of z_m^g. Observe that for a σ-LFSR with a single delay block with characteristic polynomial g, the feedback gain matrix can be any matrix with characteristic polynomial g. This is used to prove the following lemma.

Lemma: 20 *Given a primitive polynomial g of degree m, the set of matrix states of sequences generated by σ-LFSRs with a single delay block having characteristic polynomial g is the set of all full rank matrices. Further, corresponding to each such σ-LFSR configuration, a unique matrix state with the first row e_1^m exists.*

Proof: Consider a σ-LFSR configuration with characteristic polynomial g having a single delay block and feedback gain matrix B. The characteristic polynomial of B is g. Given any vector v, the matrix $M_1 = [v, Bv, \ldots, B^{m-1}v]$ is a matrix state of this σ-LFSR configuration. As g is a primitive polynomial, the matrix M_1 is invertible. To prove the first statement of the lemma, we now prove that any invertible matrix having first column v is a matrix state of a sequence generated by some σ-LFSR configuration with characteristic polynomial g having a single delay block.

Let M_2 be an arbitrary full-rank matrix with first column v. Let $P = M_2M_1^{-1}$ and $B' = PBP^{-1}$. Comparing the first columns on both sides of the equation $PM_1 = M_2$, it is apparent that v is an eigenvector of P corresponding to the eigenvalue 1. Now,

$$\begin{aligned} M_2 = PM_1 &= P[v, Bv, \ldots, B^{m-1}v] \\ &= [Pv, PBv, \ldots, PB^{m-1}v] \\ &= [v, PBP^{-1}Pv, \ldots, PB^{m-1}P^{-1}Pv] \\ &= [v, PBP^{-1}v, \ldots, PB^{m-1}P^{-1}v] \\ &= [v, B'v, \ldots, B'^{m-1}v] \end{aligned}$$

Thus, M_2 is a matrix state of a σ-LFSR with a feedback matrix B' having characteristic polynomial g. Thus, any arbitrary matrix having the first column v is a matrix state of a σ-LFSR with characteristic polynomial g. Further, as v can be arbitrarily chosen, any full rank matrix is a matrix state of a sequence generated by a σ-LFSR configuration with characteristic polynomial g having a single delay block. Alternatively, the set of matrix states of sequences generated by σ-LFSRs with a single delay block having characteristic polynomial g is the set of all full-rank matrices.

Now, the minimal polynomial of each component sequence is $g(x)$. Therefore, every set of m-consecutive binary values occurs exactly once as a sub-sequence of these sequences in each period [74]. Thus, for each sequence generated by a σ-LFSR with characteristic polynomial g, there is a unique matrix state with the first row as e_1^m. As all sequences generated by a primitive σ-LFSR configuration are just shifted versions of each other, this matrix uniquely characterizes the σ-LFSR configuration. □

As a consequence of the above lemma, for a primitive polynomial $g(x)$ with degree m and companion matrix M_g, given a random full rank matrix $M = [e_1^m; v_2; \ldots; v_{m-1}] \in \mathbb{F}_2^{m\times m}$, the set of integers $(d_1, d_2, \ldots, d_{m-1})$ such that $v_i = e_i^m M_g^i$ for $1 \leq i \leq m-1$ is an element of z_m^g. However, given an intertible matrix in $\mathbb{F}_2^{m\times m}$, calculating the set of integers $(d_1, d_2, \ldots, d_{m-1})$ is not trivial. This is illustrated in the following lemma and corollary.

Lemma: 21 *Let $a, b \in F_2^m$ and g be a primitive polynomial of degree m over $GF(2)$. Let $M_g \in F_2^{m\times m}$ be the companion matrix of g. The calculation of the value i in the equation $a \times M_g^i = b$ takes $\mathcal{O}(m \times e_K + e_K \times \sqrt{P_K} + m^3)$ time, where $2^m - 1 = \prod_{i=1}^{k} P_i^{e_i}$ and P_k is the largest factor.*

Proof: M_g is a representation of a root of g and can therefore be seen as a primitive element of F_{2^m}. Therefore, $\mathcal{B} = \{M_g, M_g^2, \cdots, M_g^{m-1}, M_g^m\}$ forms a basis for F_2^m. Hence, any M_g^i can be given by a linear combination of the elements in $\mathcal{B}$. Given an equation $a \times M_g^i = b$ for $a, b \in \mathbb{F}_2^m$, the value of M_g^i can be computed by solving the following linear equation for v

$$v \times \begin{bmatrix} a \\ a \times M_g \\ a \times M_g^2 \\ \vdots \\ a \times M_g^{m-1} \end{bmatrix}_{m\times m} = b \tag{4.1}$$

If $v = (a_0, a_1, \ldots, a_{m-1}$, then $M_g^i = (a_0 \times I + a_1 \times M_g + \cdots + a_{m-1} \times M_g^{m-1})$. Solving Equation 4.1 takes $\mathcal{O}(m^3)$ operations. The value of i can then be found from M_g^i by calculating the discrete logarithm using the Pohlig Hellman algorithm[75]. The time complexity of Pohlig Hellman algorithm is $\mathcal{O}(m \times e_k + e_k \times \sqrt{P_k})$, where $2^m - 1 = \prod_{i=1}^{k} P_i^{e_i}$ and P_k is the largest factor. Therefore, the total time needed to calculate the value of i is $\mathcal{O}(m \times e_K + e_K \times \sqrt{P_K} + m^3)$. □

Calculating an element of z_m^g from the matrix M involves solving $(m-1)$ instances of the problem discussed in the above lemma. We, therefore, have the following corollary.

Corollary: 22 *Given a matrix $M = (e_1^m, v_1, \cdots, v_{m-1}) \in F_2^{m\times m}$ and a companion matrix $M_g \in F_2^{m\times m}$ of a primitive polynomial g, Finding the distance vectors $d_1, d_2, \cdots, d_{m-1}$ using Pohlig Hellman Algorithm from the equation $e_1^m \times M_g^{d_i} = v_i$ for $i \in [1, m-1]$ takes $\mathcal{O}((m-1) \times (m \times e_K + e_K \times \sqrt{P_K} + m^3)))$ time.*

From Corollary 2 and Lemma 5, it is apparent that calculating an element of z_m^g from a random invertible matrix is computationally expensive. We therefore present a randomized algorithm (Algorithm 5) where an invertible matrix M_1 and the distance vector are simultaneously generated. The first row M_1 is taken as e_1^m. The algorithm runs for $m-1$ iterations. In the i-th iteration, an integer d_i is randomly chosen from the set of integers ranging from 1 to $2^m - 1$. The vector $e_1^m M_g$ is then evaluated. If this vector is linearly independent of the previously added rows of M_1, then d_i is appended to the list of entries of the distance vector. Otherwise, the process is repeated with a new choice of d_i. Thus, each iteration adds a new entry to the distance vector, and a new row is appended to M_1. The linear independence of the newly added vector is checked by simultaneously generating a matrix M_2 whose entries are all 0 below the anti-diagonal. Further, for all $1 \leq j \leq m$, the span of the first j rows of M_2 is the same as the span of the first j rows of M_1. The linear independence of the rows of M_2 ensures the linear independence of the rows of M_1. Theorem 23 proves the algorithm's correctness. Moreover, the proof gives an insight into the workings of the algorithm.

Algorithm 5 Randomized Algorithm to generate a z-set for g

Input:

1. A companion matrix $M_g \in \mathbb{F}_2^{m\times m}$ of primitive polynomial g.

```
1: procedure $Z - Set(M_g)$
2:     $M_1 \leftarrow \begin{bmatrix} e_1^m \\ z \\ z \\ \vdots \\ z \end{bmatrix}$  $M_2 \leftarrow \begin{bmatrix} e_1^m \\ z \\ z \\ \vdots \\ z \end{bmatrix} \in \mathbb{F}_2^{m\times m}$      ▷ z is zero vector of dimension m
3:     $i \leftarrow 2$
4:     while do  $i \neq m$
5:         $d \leftarrow Random - Integer(2, 2^m - 1)$
6:         $v \leftarrow e_1^m \times M_g^d$
7:         $M_1[i, :] \leftarrow v$
8:         $M_2[i, :] \leftarrow v$
9:         $j \leftarrow 1$
10:
11:        while do  $j \neq i$
12:            $M_2[i, :] \leftarrow M_2[i, :] + M_2[i, m - j + 1] * M_2[j, :]$
13:            $j \leftarrow j + 1$
14:        end while
15:        if  $M_2[i, m - i + 1] == 1$
16:           $List.add(d)$
17:           $i \leftarrow i + 1$
18:        endif
19:    end while
20: end procedure
21: Output:
    1. Full rank Matrix $M_1 \in F_2^{m\times m}$.
    2. The distance vector, List.
```

In Algorithm 5, we generate the z-set for a given primitive polynomial g. It uses a random number generator to generate the distance values that build an invertible matrix. Now, we calculate the average running time of Algorithm-5 to generate an invertible matrix. As the first vector for both the matrix is e_1^m, the probability of the vector,v_1, being linearly independent of e_1^m, is $1 - 2^{1-m}$. Hence, the average number of vectors that need to be sampled to get such a v_1 is $\frac{1}{(1-2^{1-m})}$. Similarly, the average number of vectors that need to be sampled to get a v_i which is independent of $e_1^m, v_1, \ldots, v_{i-1}$ is $\frac{1}{(1-2^{i-m})}$. This number is always less than 2. Hence, on average, generating a full rank matrix requires less than 63 iterations. The above algorithm's main computational challenge is finding the value M_g^d. This can be done using the Binary Exponentiation Algorithm(BEA)[76] with matrix M_g and integer d as inputs. Using BEA, for any $d \in [1, 2^m - 1]$, M_g^d can be calculated in $\mathcal{O}(m^3)$) time. Here, two matrices are multiplied in $\mathcal{O}(m^2)$ time using Algorithm 6.

Algorithm 6 Matrix-Matrix Multiplication over Binary Field

```
1: procedure MATRIX-MULTIPLICATION(A,B)
2:
3:     for $i = 0$ to $N$ do
4:         $Sum = 0$
5:
6:         for $j = 0$ to $N$ do
7:             $Sum += ((\textbf{COUNTSETBITS}((A[i]\&\&B[j])))\&\&1) \times 2^{N-1-j}$
8:
9:         end for
10:        $M[i] = Sum$
11:
12:    end for
13:
14: end procedure
```

The $COUNTSETBITS(x)$ function in the above algorithm counts the number of ones in $(A[i,:]\&\&B[:,j])$. The output of the bitwise AND of the COUNTSETBITS function with the integer 1 tells us the number is even or odd. It gives integer 1 when the count is odd and gives 0 when the count is even. It runs in $\mathcal{O}(1)$ time. The working C code of the function is given below whether

```
int Btable[256];
void initialize()
{
        Btable[0]=0;
        int i;
        for(i=0;i<256;i++)
        {
                Btable[i]=(i&1)+Btable[i/2];

        }
}

u32 COUNTSETBITS(u32 n)
{
        return (Btable[n & 0xff]+
        Btable[(n>>8)& 0xff]+
        Btable[(n>>16)&0xff]+
        Btable[(n>>24)])&1;
}
```

Theorem: 23 *Given a primitive polynomial g of degree m over $GF(2)$, Algorithm 2 generates an element of the z^g_m.*

Proof: As a result of Lemma 20, the above theorem stands proved if it is proved that the matrix M_1 is invertible. This is proved by inductively proving that each new row added to M_1 is linearly independent of all its previous rows. We now inductively prove the following statements;

- For all $1 \leq i \leq m$, the first i rows of M_2 are linear combinations of the first i rows of M_1. Further, $M_2(i, m-i+1) = 1$ and $M_2(i, k) = 0$ for all $k > m-i+1$.
- For all $1 \leq i \leq m$, the first i rows of M_1 are linearly independent.

As the first rows of M_1 and M_2 are e^m_1, both these statements are trivially true when $i = 1$. Assume they are true for all $i < \ell < m$.

Observe that, when $i = \ell$, due to lines 7 and 8 of Algorithm 5, $M_2[\ell, :]$ is initially equal to $M_1[\ell, :]$. $M_2[\ell, :]$ is then modified in the subsequent while loop (Line 11 to Line 15 of Algorithm 5) by adding it with some of the previous rows of M_2. By assumption, the first $\ell - 1$ rows of M_2 are linear combinations of the first $\ell-1$ rows of M_1. Hence, at the end of this while loop, $M_2(\ell, :)$ is a linear combination of the first ℓ rows of M_1. Further, by assumption $M_2(j, m-j+1) = 1$ and $M_2(j, k) = 0$ for all $j < \ell$ and $k > m-j+1$. Therefore, when $i = \ell$ and $j = p < \ell$, in Line 12 of Algorithm 5, the p-th row of M_2 (whose $m-p+1$-th entry is 1) is added to $M(\ell, :)$ if and only if $M(i, m-p+1) = 1$. Therefore, after this addition $M(i, m-p+1)$ becomes zero. Further, all the entries of $M(\ell, :)$ that have been made zero in the previous iterations of the while loop remain zero as $M(p, k) = 0$ for all $k > m-p+1$. Therefore, at the end of the inner while loop $M(\ell, k) = 0$ for all $k > m-\ell+1$. Now, the value of i is incremented only when $M_2(\ell, m-\ell+1) = 1$. Therefore, $M_2(\ell, m-\ell+1) = 1$ and $M_2(\ell, k) = 0$ for all $k > m-\ell+1$.

The structure of the first ℓ rows of M_2 ensures these rows are linearly independent. Further, since these rows are a linear combination of the first ℓ rows of M_1, the first ℓ rows of M_1 are linearly independent. Thus, both statements are true when $i = \ell$ and the theorem stands proved.

□

The element of z^g_m generated in Algorithm 5 is then used to find an element of z^f_{mb} using the bijection mentioned in Theorem 14. This is then used to generate the desired σ-LFSR configuration in the following algorithm.

Algorithm 7 Generation of m-companion matrix of Z primitive $\sigma-$ LFSR

```
1: Input :
       1. Distance vector $D = (d_1, d_2, \cdots, d_{m-1})$(Generated from Algorithm 5).
       2. Randomly sampled Primitive polynomial $f$ of degree $n(mb)$.
2: procedure CONFIG-GEN($INV_m, f, g$) .
3:     Compute $z \leftarrow \frac{2^{mb}-1}{2^m-1}$
4:     Construct the subspace $V_f$ of $F_2^{mb}$ by following:
           $$V_f = (e_1^n, e_1^n * M_f^{z*d_1}, e_1^n * M_f^{z*d_2}, \cdots, e_1^n * M_f^{z*d_{m-1}}) = (e_1^n, w_1, \cdots, w_{m-1})$$
                                    ▷ where $M_f$ is the companion matrix of the polynomial $f$
5:     $Q \leftarrow \begin{bmatrix} e_1^n \\ w_1 \\ \vdots \\ w_{m-1} \\ e_1^n \times M_f \\ w_1 \times M_f \\ \vdots \\ w_{m-1} \times M_f \\ \vdots \\ e_1^n \times (M_f)^{b-1} \\ w_1 \times (M_f)^{b-1} \\ \vdots \\ w_m \times (M_f)^{b-1} \end{bmatrix} \in \mathbb{F}_2^{n \times n}$
6:     $A_{mb} \leftarrow Q \times M_f \times Q^{-1}$
7: end procedure
8: Output : $M-$companion matrix $A_{mb}$ over $M_m(F_2)$.
```

Theorem: 24 *Algorithm7 generates the configuration matrix of a $z-$primitive $\sigma-$LFSR whose characteristic polynomial is the primitive polynomial f considered in Algorithm 4.*

Proof: The set $D = \{d_1, \cdots, d_{m-1}\}$ is an element of z_m^g. Therefore, by Theorem 14 the set $D' = \{zd_1, \ldots, zd_{m-1}\}$ is an element of z_{mb}^f. Therefore, by the arguments given in Section 2, the vectors $(e_1^n, w_1, \ldots, w_{m-1}, e_1^n M_f, w_1 M_f, \cdots, w_{m-1} M_f^{b-1}, \ldots, e_1^n M_f^{b-1}, w_1 M_f^{b-1}, \cdots, w_{m-1} M_f^{b-1})$ are linearly independent. Hence, by Lemma 10, the matrix $Q M_f Q^{-1}$ is an m-companion matrix with characteristic polynomial f.

Now, by Theorem 12, the first m-rows of the matrix Q constitute a matrix state of the sequence generated by a σ-LFSR with configuration matrix QM_fQ^{-1}. Therefore, the distance vector of this σ-LFSR is $\{zd_1, \cdots, zd_{m-1}\}$. As each entry of this vector is a multiple of z, the σ-LFSR is z-primitive.

□

Complexity Analysis:- Each $M_f^{z \times d_i}$ in Step 4 of Algorithm 7 can be calculated in $\mathcal{O}(n^3)$ time. Since this operation is done $(n-1)$-times, the computational complexity of Step 4 is $\mathcal{O}(n^4)$. Further, $Q \times M_f \times Q^{-1}$ can be calculated in $\mathcal{O}(n^3)$ time. Thus, the overall time complexity of Algorithm 4 is $\mathcal{O}(n^4)$.

Example: 6 *For the case where $m = 3$ and $b = 3$, we aim to generate a z-primitive σ-LFSR configuration with characteristic polynomial $f(x) = x^9 + x^6 + x^4 + x^3 + x^2 + x + 1$. The following is the companion matrix of $f(x)$*

$$M_f = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

1. $z = \frac{2^9-1}{2^3-1} = 73$

2. $g(x) = (x + \alpha^{73}) * (x + \alpha^{73*2}) * (x + \alpha^{73*4}) \pmod{f(x)} = x^3 + x^2 + 1$ *, where α is root of $f(x)$.*

3. *Randomly sample a matrix from linear group $GL(3, GF(2))$. Let the sampled matrix be*

$$Mat = \left(\begin{array}{cc|c} 0 & 0 & 1 \\ \hline 1 & 0 & 0 \\ 0 & 1 & 0 \end{array}\right)$$

4. *The distance vector corresponding to the above matrix is* $\{6, 5\}$

5. *The subspace V_f in Step 4 of Algorithm 4 is as follows:*

$$V_f = \{e_1^9, e_1^9 * M_f^6, e_1^9 * M_f^5\}$$

6. *The matrix Q in Step 5 of Algorithm 4 is as follows*

$$Q = \begin{pmatrix} 0&0&0&0&0&0&0&0&1\\ 0&1&1&0&0&0&1&0&1\\ 1&0&0&0&0&1&0&1&1\\ 0&0&0&0&0&0&0&1&0\\ 1&1&0&0&0&1&0&1&1\\ 0&0&0&0&1&0&1&1&1\\ 0&0&0&0&0&0&1&0&0\\ 1&0&0&0&1&0&1&1&0\\ 0&0&0&1&0&1&1&1&0 \end{pmatrix}$$

7. *The configuration matrix of the z-primitive σ-LFSR is calculated as follows,*

$$Q \times M_f \times Q^{-1} = \begin{pmatrix} 0&0&0&1&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1\\ 1&0&1&1&0&1&0&1&0\\ 1&0&0&1&0&0&0&0&1\\ 0&1&0&0&1&0&1&1&0 \end{pmatrix} = \left(\begin{array}{ccc|ccc|ccc} 0&0&0&1&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0\\ 0&0&0&0&0&1&0&0&0\\ \hline 0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1\\ \hline 1&0&1&1&0&1&0&1&0\\ 1&0&0&1&0&0&0&0&1\\ 0&1&0&0&1&0&1&1&0 \end{array}\right)$$

8. *The gain matrices for the configuration matrix ($Q \times M_f \times Q^{-1}$) are follows* $B_0 = \begin{bmatrix} 1&0&1\\ 1&0&0\\ 0&1&0 \end{bmatrix}$

$B_1 = \begin{bmatrix} 1&0&1\\ 1&0&0\\ 0&1&0 \end{bmatrix}$ $\quad B_2 = \begin{bmatrix} 0&1&0\\ 0&0&1\\ 1&1&0 \end{bmatrix}$

4.1.2 Generation of the Public Parameter

Recall that in the generation of the configuration matrix, the primitive polynomial in Algorithm-4 and the elements of the distance vector in Algorithm 5 are randomly chosen independent of the key. Hence, the key does not completely contain the information needed to recover the σ-LFSR configuration. Therefore, in addition to the key, we generate a publicly known parameter matrix C to enable the receiver to recover the σ-LFSR configuration.

Note that the proposed scheme has two secret keys $K_1, K_2 \in F_2^{128}$ and two initialization vectors $IV_1, IV_2 \in F_2^{128}$. We now state the algorithm that generates the public matrix C followed by a brief explanation.

Algorithm 8 Generation of Public Matrix on the Sender Side

```
1: Input:
2: Two random keys $K_1$, $K_2 \in F_2^{128}$.
3: Two public initialization vectors $IV_1$ and $IV_2 \in F_2^{128}$.
4: procedure GEN_CMatrix($f, K_1, K_2, IV_1, IV_2$)
5:     Take the Configuration matrix $M_1$ from Algorithm 7 with $m = 32$ and $b = 16$.
6:     Generate the vector $v_1 \in F_2^{512}$ using the initial state generation algorithm of SNOW 2.0
   with $(K_1, IV_1)$. Let $v_1 \leftarrow v_{1,1}||v_{1,2}||v_{1,3}||v_{1,4}$ where $v_{1,i} \in F_2^{128}$.
7:     $U[1, :] \leftarrow v_1$.
8:
9:     for $i = 2$ to 512 do
10:        $v_{i,k} \leftarrow AES - 128(v_{i-1,k}, K_2, IV_2)$                    ▷ $k \in [4]$
11:        $v_i \leftarrow v_{i,1}||v_{i,2}||v_{i,3}||v_{i,4}$
12:        $U[i, :] \leftarrow v_i$
13:
14:    end for
15:    $a \leftarrow 2^{512} - 1$.
16:
17:    for $i = 0$ to 511 do
18:        $Mask[i] \leftarrow (a >> i)$
19:
20:    end for
21:
22:    for $i = 0$ to 511 do
23:        $U[i, :] \leftarrow (U[i] \&\& Mask[i])$
24:        $U[i, :] \leftarrow U[i, :] \oplus (1 << (511 - i))$
25:
26:    end for
27:    $L \leftarrow U^T$
28:    $S \leftarrow L \times U$
29:    Compute $C = M_1[480 : 512, :] \times S$.
30: end procedure
31: Output: $C \in F_2^{32 \times 512}$.
```

In the above algorithm, the secret key K_1, K_2 is used to generate a secret matrix S multiplied with the last 32 rows of the M-companion matrix, M_1, to produce the matrix C. This C is then made public. To generate a matrix M, a vector v_1 is generated using the (key, IV) pair K_1, IV_1. This is done using the procedure to generate the initial state of SNOW 2.0 and SNOW 3G. v_1 is then divided into four equal parts of 128 bits. Each part is encrypted using the AES-128 algorithm using the key IV pair K_2, IV_2. The encrypted words are then concatenated to generate the next vector v_2. This procedure is then followed recursively to generate 512 vectors. These vectors are considered as rows of a matrix. The elements of the matrix below the diagonal are then discarded, and all the diagonal elements are made 1. This results in an upper triangular matrix U. The matrix S is then generated by multiplying the transpose of U with U. That is $S = U \times U^T$.

Algorithm 9 Recovery of Public matrix on the Receiver Side

```
1: Input:
2: Take two random keys $K_1$, $K_2 \in F_2^{128}$.
3: Take two public initialization vectors $IV_1$ and $IV_2 \in F_2^{128}$.
4: procedure GEN_CMatrix($f, K_1, K_2, IV_1, IV_2$)
5:     Generate the vector $v_1 \in F_2^{512}$ using the initial state generation algorithm of SNOW 2.0
       with $(K_1, IV_1)$. Let $v_1 \leftarrow v_{1,1}||v_{1,2}||v_{1,3}||v_{1,4}$ where $v_{1,i} \in F_2^{128}$.
6:     $U[1,:] \leftarrow v_1$.
7:
8:     for $i = 2$ to 512 do
9:         $v_{i,k} \leftarrow AES - 128(v_{i-1,k}, K_2, IV_2)$                $\triangleright\ k \in [4]$
10:        $v_i \leftarrow v_{i,1}||v_{i,2}||v_{i,3}||v_{i,4}$
11:        $U[i,:] \leftarrow v_i$
12:
13:    end for
14:    $a \leftarrow 2^{512} - 1$.
15:
16:    for $i = 0$ to 511 do
17:        $Mask[i] \leftarrow (a >> i)$
18:
19:    end for
20:
21:    for $i = 0$ to 511 do
22:        $U[i,:] \leftarrow (U[i] \&\& Mask[i])$
23:        $U[i,:] \leftarrow U[i,:] \oplus (1 << (511 - i))$
24:
25:    end for
26:    $L \leftarrow U.transpose()$
27:    Compute $S = U \times L$.
28:    Compute the inverse of $S$.
29: end procedure
30: Output: $C \times S^{-1} \in F_2^{32\times512}$.
```

The receiver computes the matrix S from (K_1, K_2). The feedback gain matrices B_i are recovered by multiplying the public matrix C(Generated in Algorithm 8) with the inverse of S. Thus, the receiver regenerates the LFSR configuration from the keys (K_1, K_2) and the public matrix C

Whenever a user intends to transmit confidential data using an LFSR-based word-oriented stream cipher, Algorithm-5 uses (K_1, IV_1, K_2, IV_2) pair to mask the configuration matrix of the

LFSR. The process of generating the challenge matrix occurs during the pre-computing phase prior to the initialization cycle of the Cipher. The $\sigma-$LFSR configuration corresponding to the M-companion matrix generated in Algorithm 7 is then used for Keystream Generation.

4.1.3 Initialization Phase:

The initialization phase of this scheme is the same as the initialization phase of SNOW 2.0 or SNOW 3G. It runs for 32 clock cycles using the same feedback polynomial over $GF(2^{32})$, and the adversary cannot access the keystreams during this period. At the last clock, it replaces the coefficients of the feedback polynomial over $GF(2^{32})$ by the gain matrices $B_i \in F_2^{32\times 32}$ M_1.

4.1.4 Keystream Generation Phase:

In the Key generation phase, the LFSR part of SNOW 2.0 and SNOW 3G is regulated by the following equation:

$$D_{15}^{t+1} = B_0 D_0^t + B_1 D_1^t + \cdots + B_{15} D_{15}^t \tag{4.2}$$

where $B_i \in F_2^{32\times 32}, i \in [16]$ is the gain matrices of M_1. And, each delay block($D_j, j \in [16]$) is updated as:

$$D_k^{t+1} = \begin{cases} D_{k+t+1}^0 & 0 \leq k+t+1 \leq 15 \\ \sum_{i=0}^{15} B_i D_i^{t+k} & k+t+1 > 15 \end{cases} \tag{4.3}$$

The FSM update is the same as in SNOW 2.0 and SNOW 3G.

4.1.5 Security of the proposed scheme

Here, in addition to the keystream, the attacker has access to the public matrix C generated by Algorithm 8. He/She could potentially use this matrix to retrieve the gain matrices of the LFSR. On average, a brute force attack will require 2^{255} guesses to get to the correct key. We now show that other methods of deriving the feedback configuration are computationally more expensive.

Lemma: 25 *Given any symmetric invertible matrix $S' \in \mathbb{F}_2^{mb\times mb}$, there exists a matrix $M' \in \mathbb{F}_2^{m\times mb}$ such that the public parameter C is a product of M' and S.*

Proof: Let M_1 be the configuration matrix of the σ-LFSR and let $C = M_1[mb-m+1 : mb, :] \times S$ be the public parameter matrix derived in Algorithm 8. Given any symmetric invertible matrix $S' \in \mathbb{F}_2^{mb\times mb}$,

$$\begin{aligned} C &= M_1[mb-m+1 : mb, :] \times S \times S'^{-1} \times S' \\ &= M' \times S' \end{aligned}$$

where $M' = M_1[mb - m + 1 : mb, :] \times S \times S'^{-1} \in \mathbb{F}_2^{m \times mb}$. $\square$

To find the σ-LFSR gain matrices from the public parameter, one can sample an invertible symmetric matrix S' and find the corresponding values of M' such that $C = M' \times S'$. If the feedback configuration corresponding to M' is z-primitive, assuming this to be the feedback configuration, one can launch any existing attack to generate the initial state of the LFSR. The number of invertible symmetric matrices in $\mathbb{F}_2^{mb \times mb}$ is $2^{\frac{mb(mb-1)}{2}}$, Hence, the average number of attempts needed to get the correct invertible matrix is $\mathcal{O}(2^{\frac{mb(mb-1)}{2}-1})$ which is prohibitively high.

Alternatively, one could sample a z-primitive LFSR configuration and consider the configuration matrix a potential M_1. One can then check if the public parameter can be written as a product of this matrix's last m rows and a symmetric matrix S. This involves checking if a set of $m \times mb$ linear equations in $\frac{mb(mb-1)}{2}$ variables has a solution (For SNOW 2.0 and SNOW 3G, the number of equations is 8192 and the number of variables is 130816). If this set of equations has a solution, one can launch any known attack to recover the initial state of the LFSR. For a given value of m and b, the total number of z-primitive LFSR configurations is $\frac{|GL(m,\mathbb{F}_2)|}{2^m-1} \times \frac{\phi(2^{mb}-1)}{mb}$. For SNOW 2.0 and SNOW 3G, this number turns out to be 2^{1493}. Therefore, the average number of z-primitive LFSRs that must be sampled to get to the right configuration is approximately 2^{1492}.

4.1.6 Resistance to Attacks:

Several Known plaintext attacks like Algebraic Attacks[30, 31], Distinguishing Attacks([32, 33]),Fast Correlation Attacks[34, 35, 70, 64, 1], Guess and Determine Attacks[77, 66], Cache Timing Attacks([36, 68]) are reported for SNOW 2.0 and SNOW 3G. All these attacks either use the feedback equation of the σ-LFSR or the linear recurring relation corresponding to the characteristic polynomial of the LFSR. A method of hiding the configuration matrix is explained in [78]. However, the characteristic polynomial of the σ-LFSR is publicly known. The scheme given in [78] is therefore vulnerable to schemes that use the characteristic polynomial of the σ-LFSR. This polynomial gives rise to a linear recurring equation of degree 512 with coefficients in $\mathbb{F}_2$. These schemes include the fast correlation attack given in [1] and the fault attack on SNOW3G given in [79]. The characteristic polynomial in this scheme is not known. Therefore, to get to the characteristic polynomial, the attacker has to keep sampling from the set of primitive polynomials of degree 512 till he/she gets to the correct characteristic polynomial. As the number of primitive polynomials of degree 512 over $\mathbb{F}_2$ is 2^{502}, the attacker on average will sample 2^{501} polynomials. Alternatively, by sampling the key space, the attacker could try to generate the symmetric matrix S in Algorithm 8. This, on average, will take 2^{255} attempts to get to the correct key.

References	Applied Ciphers	FCA Complexity	Our Scheme
Lee et al.,2008[34]	SNOW 2.0	$2^{204.38}$	2^{256}
Zhang et al.,2015[35]	SNOW 2.0	$2^{164.15}$	2^{256}
Todo et al.,2018[63]	SNOW 2.0	$2^{162.91}$	2^{256}
Yang et al.,2019[64]	SNOW 3G	2^{177}	2^{256}
Gong et al.,2020[1]	SNOW 2.0, SNOW 3G	$2^{162.86}, 2^{222.33}$	$2^{256}, 2^{256}$

Table 4.1: Comparison results of our scheme with various schemes

4.2 Experiment Result

The proposed scheme has been implemented in C (using a GCC compiler) on a machine with an Intel Core i5-1135G7 processor having 8GB RAM and a 512 GB HD drive. The parallel implementation of $e_1^n \times M_f^{z \times d_i}$ for $n = 512$, $M_f \in F_2^{512 \times 512}$ and $1 \leq |d_i| \leq 512$ where $i \in \{1, 31\}$, took a total of 0.04 seconds. The calculation $P \times Q \times P^{-1}$(step-6 in Algorithm 7) took another .08 seconds. Algorithm 7 was completed in 0.13 seconds. The total initialization time for our scheme is, on average, 0.2 seconds (Averaged over 200 test cases). Besides, to accelerate the keystream generation process, 32- bit vector-matrix multiplications in the $\sigma-$LFSR are done using Algorithm 6 with constant time complexity, $\mathbb{O}(c)$, where $c = 32$. This lead to an improvement in performance over existing implementations. The Key generation times(KGT) for SNOW 3G and our proposed scheme are given in the following table. ook-up table-based implementation of LFSR in SNOW 3G can be cryptanalysed by cache timing attacks[36]. Hence, we have considered the implementation of the LFSR part of SNOW 3G(especially field multiplication over $GF(2^{32})$) programmed without any look-up tables.

Number of Keystreams	KGT for SNOW3G	KGT for Proposed SNOW3G)
2^{10}	.009490 Seconds	.003138 Seconds
2^{15}	.2032 Seconds	.0586 Seconds
2^{20}	6.397 Seconds	1.8152 Seconds

Table 4.2: Comparison of Key Generation Time of our proposed scheme with SNOW 3G

4.2.1 Hardware Implementation

Tools and Framework	AMD Xilinx Vitis HLS
Target Device	Zynq 7000 ZC702 Evaluation Board xc7z020-clg484-1

Table 4.3: Specification for Hardware Implementation

For our hardware implementation, we use an HLS (High-Level Synthesis) framework to synthesise both baseline SNOW 3G and Proposed SNOW 3G on the device highlighted in

Table 4.3. We generate $z-$primitive σ LFSR Configuration generated from the software and use it in the implementation of our proposed SNOW 3G scheme on FPGA.

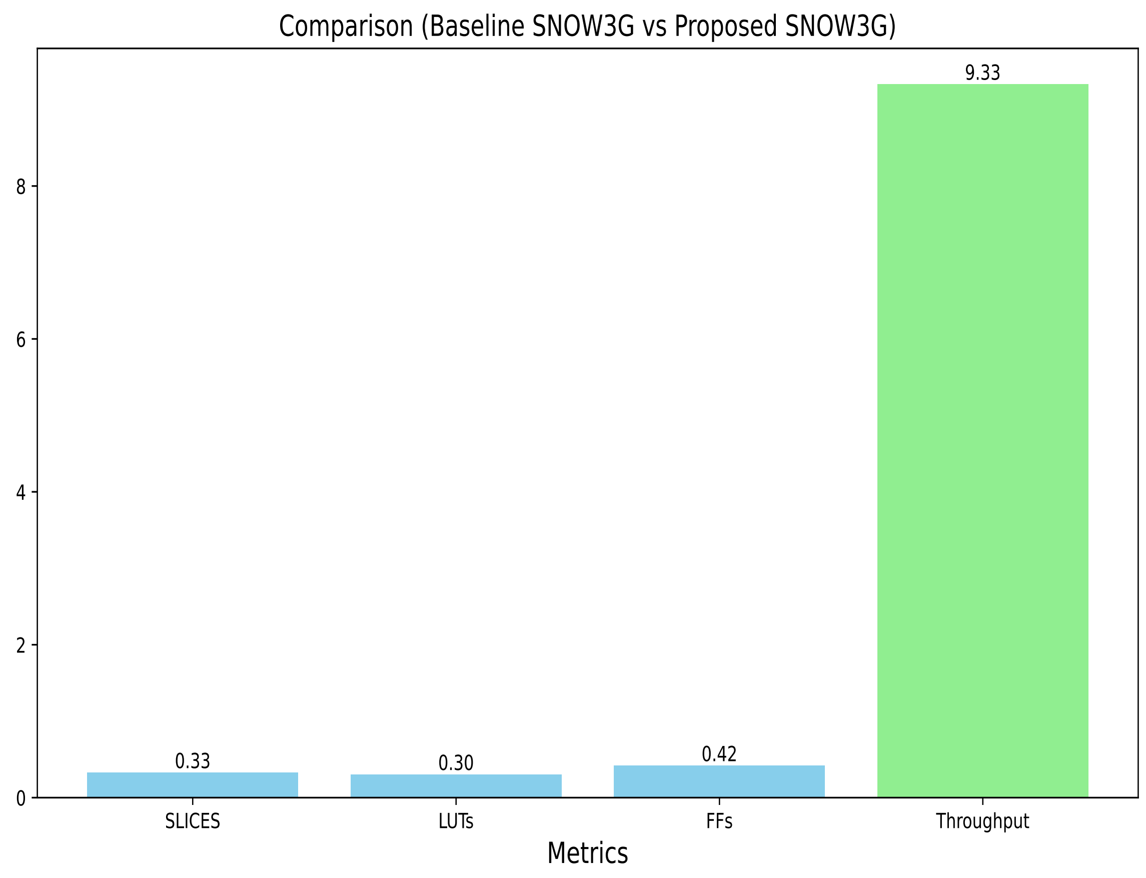


Fig. 4.1: Comparison of Hardware Implementation

Figure **??** illustrates the comparison between our proposed SNOW3G and the baseline SNOW3G implemented on FPGA. Our analysis reveals significant resource savings in our scheme, with reductions of 0.33, 0.3, 0.42 times in terms of Slices, LUTs, and FFs respectively, compared to the baseline SNOW3G. Moreover, our scheme demonstrates a remarkable 9.3x improvement in throughput, attributable to our optimized algorithms.

4.3 Summary

In this chapter, we have introduced a z-primitive sigma-LFSR generation algorithm to generate an m-companion matrix of m input-output and b number of delay blocks for word-based LFSR. Besides, to hide the feedback polynomial over $GF(2)$, we have multiplied a key-dependent invertible matrix with the m-companion matrix. Finding the part of the multiplied matrix shared as a public parameter is analogous to searching a symmetric matrix from the space of $2^{256\times 511}$. Our scheme can resist Fast Correlation Attacks (FCA). We have shown that applying our scheme to SNOW 2.0, SNOW 3G, and Sosemanuk can withstand FCA. Besides, our scheme is robust against any known plaintext attacks based on the Feedback equation of the LFSR.

Chapter 5

Removing the Public Parameter

In this chapter, we discuss methods of doing away with the public parameter in the scheme described in chapter-4. In this scheme, the receiver cannot regenerate the feedback configuration of the σ-LFSR from just the key. This is because some parameters in the initialization process are randomly sampled. This includes the characteristic polynomial of the σ-LFSR and the entries of the distance vector in Algorithm 5. The public parameter hides the feedback configuration using a matrix generated from the secret key. The receiver uses this parameter and the secret key to recover the feedback configuration. However, sharing the public parameter incurs a communication cost, which can be undesirable. The public parameter can be eliminated if the randomly sampled elements in the initialization process are made key-dependent. The rest of the chapter discusses methods of doing the same. During the configuration generation process, the scheme's LFSR and FSM are assumed to be identical to SNOW 3G. After the initialization process, the LFSR is replaced with the proposed σ-LFSR configuration.

5.1 Generating a random primitive polynomial

In this section, we describe an algorithm that generates a random primitive polynomial that acts as the characteristic polynomial of the σ-LFSR in the scheme proposed in the previous chapter.

As already stated, the LFSR and FSM are initially identical to SNOW 3G. However, the LFSR is initialized with a 256-bit key and a 256-bit IV (as against a 128-bit key in SNOW 3G). The output of the FSM is added to the output of the first delay block, and the scheme is run for 32 clock cycles. In this phase, the output of the scheme is discarded. The state of the system at the end of the process is considered the initial state for the characteristic polynomial generation algorithm. The algorithm for this initialization process is given below.

```
Algorithm 10 Initialization Algorithm
Input: Key $K = (k_0, \cdots, k_7) \in \mathbb{F}_2^{256}$ where each $k_i \in \mathbb{F}_2^{32}$ and Initialization vector $IV = (IV_0, IV_1, \cdots, IV_7) \in \mathbb{F}_2^{256}$ where each $IV_i \in \mathbb{F}_2^{32}$.
1: procedure INIT-SNOW3G($(K, IV)$)
2:     $(D_0, \cdots, D_{15}) \leftarrow (k_0 \oplus 1, k_1 \oplus 1, k_2 \oplus 1, k_3 \oplus 1, k_4 \oplus 1, k_5 \oplus 1, k_6 \oplus 1, k_7 \oplus 1, k_0 \oplus IV_0, k_1 \oplus IV_1, k_2 \oplus IV_2, k_3 \oplus IV_3, k_4 \oplus IV_4, k_5 \oplus IV_5, k_6 \oplus IV_6, k_7 \oplus IV_7)$
3:     $(R_1^0, R_2^0, R_3^0) \leftarrow (0, 0, 0)$
4:     for $t \leftarrow 1$ to 32 do
5:         $F \leftarrow FSMupdate()$
6:         $temp = \alpha^{-1} D_{11} \oplus D_2 \oplus \alpha D_0$
7:         $(D_{15} \cdots, D_0) \leftarrow (temp, D_{15}, \cdots, D_1 \oplus F)$
8:     end for
9: end procedure
```

In Algorithm 10, the D_is denote the outputs of the delay blocks, and the R_is denote the values of the registers.

Starting with the state generated in Algorithm 10, the scheme is run identically to SNOW 3G. The functionality of SNOW 3G for a single clock pulse is given as follows:

```
Algorithm 11 SNOW-3G-KSG
Input:Last Updated Value of$(D_0, \cdots, D_{15}, R_1, R_2, R_3)$
Output:Keystream $z$ at time t.
1: procedure KEYSTREAM-GENERATION($(D_0, \cdots, D_{15}, R_1, R_2, R_3)$)
2:     $F \leftarrow FSMupdate()$
3:     $z \leftarrow F \oplus D_0$
4:     $temp \leftarrow \alpha^{-1} D_{11} \oplus D_2 \oplus \alpha D_0$
5:     $(D_{15} \cdots, D_0) \leftarrow (temp, D_{15}, \cdots, D_1)$
6:     Output $z$
7: end procedure
```

The state vector is considered as a 512-bit integer. At each instant, the co-primeness of this integer with $2^{512} - 1$ is checked. As soon as an integer x co-prime to $2^{512} - 1$ is obtained, the minimal polynomial of M^x is calculated where M is the companion matrix of the characteristic polynomial of the LFSR in SNOW 3G. This polynomial is primitive and is taken as the characteristic polynomial of the σ-LFSR. This procedure is given in Algorithm (12).

Algorithm 12 Randomised Primitive Polynomial of degree-512 Generation Algorithm

```
1: procedure $Gen\_Key\_Based\_Prim\_Poly((M_f, K, IV))$
2:     $(i, j) \leftarrow (0, 0)$
3:     while $i < 8$ do
4:         $k_i \leftarrow rand\_int(1, 2^{32} - 1)$
5:         $i \leftarrow i + 1$
6:     end while
7:     while $j < 8$ do
8:         $IV_j \leftarrow rand\_int(1, 2^{32} - 1)$
9:         $j \leftarrow j + 1$
10:    end while
11:    $INIT - SNOW3G(K, IV)$
12:    $x \leftarrow (D_0 + (D_1 << 32) + (D_2 << 48) + \cdots + (D_{15} << 480))$
13:    $t \leftarrow 33$
14:    while do$\{GCD(x, 2^{512} - 1)! = 1\}$
15:        $F \leftarrow FSMupdate()$
16:        $temp = \alpha^{-1}D_{11} \oplus D_2 \oplus \alpha D_0$
17:        $(D_{15} \cdots, D_0) \leftarrow (temp, D_{15}, \cdots, D_1)$
18:        $x \leftarrow (D_0 + (D_1 << 32) + (D_2 << 48) + \cdots + (D_{15} << 480))$
19:        $t \leftarrow t + 1$
20:    end while
21:    $f_1 \leftarrow Compute\_Minimalpoly(M^x)$
22: end procedure
23: Output:Primitive Polynomial $f_1$.
```

5.1.1 Time Complexity Analysis:

Algorithm 12 samples a random primitive polynomial from the set of $\frac{\phi(2^{512}-1)}{512}$ primitive polynomials. The probability of an integer x being co-prime to $2^{512} - 1$ is $\frac{\frac{\phi(2^{512}-1)}{512}-1}{2^{512}} \approx \frac{1}{2^{10}}$. Hence, the average number of iterations of SNOW 3G needed to get the desired x is around 2^{10}. Calculating M_f^x and its minimal polynomial has an average time complexity of $\mathcal{O}(n^2 log(n)$.

5.2 Finding the z-set

This section describes the key-dependent version of the Algorithm 5. We use the SNOW 3G KSG algorithm as a random number generator.

Algorithm 13 Key Dependent Algorithm to generate a z-set for g

Input:

1. A companion matrix $M_g \in \mathbb{F}_2^{m\times m}$ of primitive polynomial g.
2. Take the Key $K_2 \in F_2^{128}$ and initialization vector $IV_2 \in F_2^{128}$.

```
1:  procedure $z - set(M_g, K_2, IV_2)$
2:      $M_1 \leftarrow \begin{bmatrix} e_1^m \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}$   $M_2 \leftarrow \begin{bmatrix} e_1^m \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix} \in \mathbb{F}_2^{m\times m}$      ▷ $\mathbf{0}$ is zero vector of dimension m
3:      $i \leftarrow 2$
4:      while do  $i \neq m$
5:          $d \leftarrow SNOW\_3G((D_0, D_1, \cdots, D_{15}, R_1, R_2, R_3))$
6:          $v \leftarrow e_1^m \times M_g^d$
7:          $M_1[i, :] \leftarrow v$
8:          $M_2[i, :] \leftarrow v$
9:          $j \leftarrow 1$
10:
11:         while do  $j \neq i$
12:             $M_2[i, :] \leftarrow M_2[i, :] + M_2[i, m - j + 1] * M_2[j, :]$
13:             $j \leftarrow j + 1$
14:         end while
15:         if  $M_2[i, m - i + 1] == 1$
16:             $List.add(d)$
17:             $i \leftarrow i + 1$
18:         endif
19:     end while
20: end procedure
21: Output:
        1. Full rank Matrix $M_1 \in F_2^{m\times m}$.
        2. The distance vector, List.
```

In Algorithm 13, we generate the z-set for a given primitive polynomial g. It uses SNOW-3G-KSG to generate the distance values that build an invertible matrix. Now, we calculate the average running time of Algorithm 13 to generate an invertible matrix. The probability of the vector,v_1, being linearly independent of e_1^m, is $1 - 2^{1-m}$. Hence, the average number of vectors that need to be sampled to get such a v_1 is $\frac{1}{(1-2^{1-m})}$. Similarly, the average number of vectors that need to be sampled to get a v_i which is independent of $e_1^m, v_1, \ldots, v_{i-1}$ is $\frac{1}{(1-2^{i-m})}$. This number is always less than 2. Hence, on average, generating a full rank matrix requires less than 63 iterations. Using the binary exponentiation algorithm, $e_1^m \times M_g^d$ requires $m^2 log(m)$. The following lemma proves that replacing the M_g by M_g^T generates a valid distance vector.

Lemma: 26 *Consider a m-th degree primitive polynomial $g(x)$. Let M_g be the companion matrix of g. Let $d_1, d_2, \cdots, d_{m-1}$ be integers such that the following matrix* $R = \begin{bmatrix} e_1^m \\ e_1^m \times M_g^{T^{d_1}} \\ \vdots \\ e_1^m \times M_g^{T^{d_{m-1}}} \end{bmatrix}$ *is full rank. Then, the matrix* $R^{'} = \begin{bmatrix} e_1^m \\ e_1^m \times {M_g}^{d_1} \\ \vdots \\ e_1^m \times {M_g}^{d_{m-1}} \end{bmatrix}$ *is also full rank.*

Proof: M_g and M_g^T are similar to each other, i.e., there exists an invertible matrix T such that $M_g^T = T^{-1} \times M_g \times T$. Hence, R can be written as follows,

$$R = \begin{bmatrix} e_1^m \\ e_1^m \times M_g^{T^{d_1}} \\ \vdots \\ e_1^m \times M_g^{T^{d_{m-1}}} \end{bmatrix} = \begin{bmatrix} e_1^m \\ e_1^m \times (T^{-1} \times M_g \times T)^{d_1} \\ \vdots \\ e_1^m \times (T^{-1} \times M_g \times T)^{d_{m-1}} \end{bmatrix} = \begin{bmatrix} e_1^m \\ e_1^m \times T^{-1} \times M_g^{d_1} \times T \\ \vdots \\ e_1^m \times (T^{-1} \times M_g^{d_{m-1}} \times T). \end{bmatrix}$$

Therefore,

$$R \times T^{-1} = \begin{bmatrix} e_1^m \times T^{-1} \\ e_1^m \times T^{-1} \times M_g^{d_1} \\ \vdots \\ e_1^m \times (T^{-1} \times M_g^{d_{m-1}}) \end{bmatrix}$$

As a consequence of the characteristic polynomial of M_g being primitive, there exists $\alpha \in \{1, 2^m - 1\}$ such that $e_1^m \times T^{-1} \times M_g^{\alpha} = e_1^m$. Multiplying M_g^{α} in both sides of the above equation, we get,

$$R \times T^{-1} \times M_g^{\alpha} = \begin{bmatrix} e_1^m \times T^{-1} \times M_g^{\alpha} \\ e_1^m \times T^{-1} \times M_g^{d_1} \times M_g^{\alpha} \\ \vdots \\ e_1^m \times (T^{-1} \times M_g^{d_{m-1}} \times M_g^{\alpha}) \end{bmatrix}$$

$$= \begin{bmatrix} e_1^m \\ e_1^m \times T^{-1} \times M_g^{\alpha} \times M_g^{d_1} \\ \vdots \\ e_1^m \times (T^{-1} \times M_g^{\alpha} \times M_g^{d_{m-1}}) \end{bmatrix}$$

$$= \begin{bmatrix} e_1^m \\ e_1^m \times M_g^{d_1} \\ \vdots \\ e_1^m \times M_g^{d_{m-1}}) \end{bmatrix} = R'$$

As R, T and M_g^{α} are invertible matrices, $R^{‘} = R \times T^{-1} \times M_g^{\alpha}$ is also invertible.

□

In Algorithms 13,5, $e_1^m \times M_g^{d_1}$ can be calculated using the Binary exponentiation algorithm and Algorithm-6. This method has a time complexity of $\mathcal{O}(m^3)$. This can be reduced to $\mathcal{O}(m^2)$ by replacing M_g with M_g^T. Recall that if α is a root of the primitive polynomial g, then the set $(1, \alpha, \alpha^2, \ldots, \alpha^{m-1})$ is a basis of $\mathbb{F}_{2^m}$ as a vector space over $\mathbb{F}_2$. Observe that, in this basis, e_1^m represents α^{m-1}) and M_g^T represents multiplication by α. Therefore, calculating $e_1^m \times M_g^{d_1}$ is equivalent to finding the expression of α^{m+d_1-1}. This calculation can also be done using the Binary exponentiation algorithm. This involves performing m multiplications over $\mathbb{F}_2^m$. Each such multiplication can be done in $\mathcal{O}(m)$ time using the Algorithm (15).

Algorithm 14 is the binary exponentiation algorithm for calculating $\alpha^z \pmod{f(\alpha)}$

Algorithm 14 Algorithm to find $\alpha^z \pmod{f(\alpha)}$

Input:

- n-bit integer z.

```
1: procedure $POW\_Z(z)$
2:     $Coeff \leftarrow \{z_0, z_1, \cdots, z_{m-1}\}$        ▷ Binary representation of z
3:     $power \leftarrow \{2\}$
4:     while $len(power) \leq len(Coeff)$ do    ▷ len(X) is the number of elements in the set X
5:         $x \leftarrow Mult(power[len(power)-1], power[len(power)-1], Primpoly)$
6:         $power.insert(x)$
7:     end while
8:     $result \leftarrow 1$
9:     $i \leftarrow 0$
10:    while $i \leq len(Coeff)$ do
11:        IF$(Coeff[i] == 1)$
12:        $result \leftarrow Mult(power[i], result, Primpoly)$
13:        EndIF
14:    end while
15: end procedure
16: Return:$result$
```

The multiplication over $\mathbf{F}_{2^m}$ subroutine, $Mult(a, b, Primpoly)$, used in Algorithm 14 is as follows

Algorithm 15 Finite Field Multiplication over $GF(2^m)$

```
1: procedure $Mult((a, b, Primpoly))$
2:     $result \leftarrow 0$
3:     while $b > 0$ do
4:         IF$(b\&\&1)$
5:         $result = result$ BITXOR $a$
6:         EndIF
7:         $a = a << 1$
8:         IF$(a\&\&(1 << n))$
9:         $a = a$ BITXOR $Primpoly$
10:        EndIF
11:        $b = b >> 1$
12:    end while
13: end procedure
14: Output: result
```

Given an integer $0 \leq z \leq 2^m - 1$, α^z can be expressed as follows

$$\alpha^z = \prod_{i=0}^{m-1} \alpha^{z_i 2^i} = \prod_{\substack{0 \leq i \leq m-1 \\ z_i = 1}} \alpha^{2^i} \tag{5.1}$$

where $(z_{m-1}z_{m-2}\ldots z_1z_0)$ is the m-bit binary representation of z. The first while loop in Algorithm (14) (Steps 4 to 7) calculates α^{2^i} mod $f(\alpha)$ for $0 \leq i \leq m-1$. These values are then used to calculate α^z in the second while loop. Both these while loops invoke the *mult* function given in Algorithm. Given polynomials $p_1(\alpha)$ and $p_2(\alpha)$ having degree less than m, this function calculates $p_1(\alpha) \times p_2(\alpha)$ mod $f(\alpha)$. p_1 and p_2 are represented as m-bit binary strings (unsigned integer) whose entries correspond to the coefficients of these polynomials. The polynomial f (given as *Primepoly* in Algorithm (15) is similarly considered as an m-bit string where the coefficient of α^m is left out.

Here, $a, b \in F_{2^m}$ and Primpoly is hexadecimal representations of $f(x)$. For example, the hexadecimal representation of $f(x) = x^4 + x + 1$ would be $0x13$. **BITXOR** denotes the bitwise xor operation. Algorithm 15 is called m-times by the Algorithm 14. Hence, the time complexity to find out α^{m-1+z} is $\mathcal{O}(mlog_2(m))$.

5.3 Summary

In this chapter, we propose two key-based randomised algorithms, Algorithm 12 and Algorithm 13. By replacing the randomised algorithms of the previous scheme with these two algorithms, we can initiate the encryption of the previous one without using the public parameter. We also have reduced the average time complexity of both algorithms by an efficient finite field multiplication over $GF(2)$ by the Algorithm 15.

Chapter 6

Future Work and Conclusion

In this chapter, we present a summary of the results that we found in our research and also discuss the scope for future research.

6.1 Summary of the Results

Here, we demonstrated the application of the randomized feedback configuration of primitive σ-LFSR in word-based stream cipher, SNOW 2.0 and SNOW 3G, to resist known plaintext attacks.

In Chapter-3, we propose the σ-KDFC scheme and apply it to SNOW 2.0 to resist attacks. It replaces the existing feedback configuration of the LFSR with a random feedback configuration of a σ-LFSR generated by the 256-bit secret key of SNOW 2.0. It resists the algebraic attacks [30, 31] by increasing the algebraic degree of the keystream equation. We theoretically analyze the algebraic degree of the unknown gain matrices, which is a function of the initial state of σ-LFSR. As this scheme hides the feedback equation of SNOW 2.0, it also resists attacks like Distinguishing Attacks, Fast Correlations Attacks, Guess and Determine Attacks and Cache Timing Attacks.

In Chapter-4, we mainly focus on attacks on Word Based Ciphers, where the LRR over $GF(2)$ is shared publicly. Fast Correlation Attack by [1] is such a kind of attack. Here, we propose a z-primitive σ-LFSR generation algorithm. Further, this algorithm depends on a randomly sampled primitive polynomial of degree 512 and z-set of a primitive polynomial of degree 32. In addition, the random z-primitive $\sigma-$LFSR configuration is masked by a key-dependent secret matrix and shared as a public parameter to the receiver. The receiver can access the configuration matrix with the help of the scheme's private key and the public parameter. We applied this scheme on SNOW 3G, and it resists all kinds of Known plaintext attacks.

In Chapter-5, we modified the previous scheme by removing the public parameter. As sharing the public parameter may incur communication costs, we generate a primitive polynomial of degree 512 and a z-set of the primitive polynomial degree 32 using the 256-bit key of SNOW 3G initialization algorithm.

6.2 Scope for Future Work

The work presented in this thesis can be extended to include the following aspects:

- The main problem in the $\sigma-$KDFC scheme is that the system's running time in the initialization phase of the cipher may take some time. The efficient implementation of the Linear solver and the inverse of the Hankel matrix computation may be taken as further studies.
- Vector-Matrix multiplication is another problem in $\sigma-$LFSR implementation. Traditional SNOW ciphers are implemented using look-up tables, which is faster than our scheme. However, look-up table-based finite field multiplication may cause Fault attacks and Cache timing attacks. Finding $\sigma-$LFSR configuration with two or three nonzero gain matrices may be taken as a future study.
- Finding the configuration matrices over $GF(2^m)$ from the search space of $\frac{|GL(m,GF(2))|}{2^m-1}$ z-primitive σ-LFSR and implementing the finite field multiplication efficiently is another interesting problem.
- Designing a public key cryptosystem using our proposed scheme would be another problem to study.

Publications

1. Nandi S, Krishnaswamy S, Zolfaghari B, Mitra P. ”Key-dependent feedback configuration matrix of primitive σ–lfsr and resistance to some known plaintext attacks”. IEEE Access. 2022 Jan 7;10:44840-54.

2. ”Recent Results on Some Word Oriented Stream Ciphers: SNOW 1.0, SNOW 2.0 and SNOW 3G”,Nandi, Subrata and Krishnaswamy, Srinivasan and Mitra, Pinaki ,2022,IntechOpen.

3. Nandi, Subrata, Satyabrata Roy, Srinivasan Krishnaswamy, and Pinaki Mitra. ”IEMS3: An Image Encryption Scheme Using Modified SNOW 3G Algorithm.” In International Conference on Network Security and Blockchain Technology, pp. 161-172. Singapore: Springer Nature Singapore, 2023.

4. Subrata Nandi, Srinivasan Krishnaswamy, Nitesh Narayana GS et al. Chakravyuha: A Scheme to Resist Fast Correlation Attack for Word Oriented LFSR based Stream Cipher, 11 April 2024, PREPRINT (Version 1) available at Research Square.(Under Review in Design Codes and Cryptography) [https://doi.org/10.21203/rs.3.rs-4228602/v1]

References

[1] X. Gong and B. Zhang, “Fast computation of linear approximation over certain composition functions and applications to snow 2.0 and snow 3g,” *Designs, Codes and Cryptography*, pp. 1–25, 2020.

[2] B. Schneier, “Description of a new variable-length key, 64-bit block cipher (blowfish),” in *International Workshop on Fast Software Encryption.* Springer, 1993, pp. 191–204.

[3] J. Daemen and V. Rijmen, “The block cipher rijndael,” in *International Conference on Smart Card Research and Advanced Applications.* Springer, 1998, pp. 277–284.

[4] C. Sanchez-Avila and R. Sanchez-Reillol, “The rijndael block cipher (aes proposal): a comparison with des,” in *Proceedings IEEE 35th Annual 2001 international carnahan conference on security technology (Cat. No. 01CH37186).* IEEE, 2001, pp. 229–234.

[5] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. Robshaw, Y. Seurin, and C. Vikkelsoe, “Present: An ultra-lightweight block cipher,” in *Cryptographic Hardware and Embedded Systems-CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings 9.* Springer, 2007, pp. 450–466.

[6] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, “The simon and speck families of lightweight block ciphers,” *cryptology eprint archive*, 2013.

[7] S. Banik, A. Bogdanov, T. Isobe, K. Shibutani, H. Hiwatari, T. Akishita, and F. Regazzoni, “Midori: A block cipher for low energy,” in *Advances in Cryptology–ASIACRYPT 2015: 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29–December 3, 2015, Proceedings, Part II 21.* Springer, 2015, pp. 411–436.

[8] M. Hell, T. Johansson, A. Maximov, and W. Meier, “A stream cipher proposal: Grain-128,” in *2006 IEEE International Symposium on Information Theory.* IEEE, 2006, pp. 1614–1618.

[9] C. De Canniere and B. Preneel, “Trivium,” in *New Stream Cipher Designs: The eSTREAM Finalists.* Springer, 2008, pp. 244–266.

[10] S. Babbage and M. Dodd, "The stream cipher mickey 2.0," *ECRYPT Stream Cipher*, pp. 191–209, 2006.

[11] V. Amin Ghafari and H. Hu, "Fruit-80: a secure ultra-lightweight stream cipher for constrained environments," *Entropy*, vol. 20, no. 3, p. 180, 2018.

[12] E. Dubrova and M. Hell, "Espresso: A stream cipher for 5g wireless communication systems," *Cryptography and Communications*, vol. 9, pp. 273–289, 2017.

[13] G. Paul and S. Maitra, *RC4 stream cipher and its variants.* CRC press, 2011.

[14] G. Rose and P. Hawkes, "The t-class of sober stream ciphers," *Unpublished manuscript. http://www. home. aone. net. au/qualcomm*, 1999.

[15] P. Ekdahl and T. Johansson, "Snow-a new stream cipher," in *Proceedings of First Open NESSIE Workshop, KU-Leuven.* Citeseer, 2000, pp. 167–168.

[16] P. Ekdahl and T. Johansson, "A new version of the stream cipher snow," in *International Workshop on Selected Areas in Cryptography.* St. John's, Newfoundland, Canada: Springer, August 2002, pp. 47–61.

[17] G. G. Rose and P. Hawkes, "Turing: A fast stream cipher," in *International Workshop on Fast Software Encryption.* Lund, Sweden: Springer, February 2003, pp. 290–306.

[18] G. Orhanou, S. El Hajji, and Y. Bentaleb, "Snow 3g stream cipher operation and complexity study," *Contemporary Engineering Sciences-Hikari Ltd*, vol. 3, no. 3, pp. 97–111, 2010.

[19] F. Xiu-tao, "Zuc algorithm: 3gpp lte international encryption standard," *Information Security and Communications Privacy*, vol. 12, 2011.

[20] P. Ekdahl, T. Johansson, A. Maximov, and J. Yang, "A new snow stream cipher called snow-v," *IACR Transactions on Symmetric Cryptology*, pp. 1–42, 2019.

[21] C. Berbain, O. Billet, A. Canteaut, N. Courtois, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier *et al.*, "Sosemanuk, a fast software-oriented stream cipher," in *New stream cipher designs.* Springer, 2008, pp. 98–118.

[22] P. Ekdaahl, T. Johansson, A. Maximov, and J. Yang, "A new snow stream cipher called snow-v," *IACR Cryptology ePrint Archive, Report 2018/1143*, 2018.

[23] G. Zeng, W. Han, and K. He, "High efficiency feedback shift register: sigma-lfsr." *IACR Cryptology ePrint Archive*, vol. 2007, p. 114, 2007.

[24] H. Niederreiter, "The multiple-recursive matrix method for pseudorandom number generation," *Finite Fields and their Applications*, vol. 1, no. 1, pp. 3–30, 1995.

[25] G. Zeng, K. He, and W. Han, "A trinomial type of σ-lfsr oriented toward software implementation," *Science in China Series F: Information Sciences*, vol. 50, no. 3, pp. 359–372, 2007.

[26] B. Tsaban and U. Vishne, "Efficient linear feedback shift registers with maximal period," *Finite Fields and Their Applications*, vol. 8, no. 2, pp. 256–267, 2002.

[27] S. R. Ghorpade, S. U. Hasan, and M. Kumari, "Primitive polynomials, singer cycles and word-oriented linear feedback shift registers," *Designs, Codes and Cryptography*, vol. 58, pp. 123–134, 2011.

[28] S. Krishnaswamy and H. K. Pillai, "On multisequences and their extensions," *arXiv preprint arXiv:1208.4501*, 2012.

[29] G.-M. Tan, G. Zeng, W.-B. Han, and X.-H. Liu, "Construction and enumeration of a class of primitive sigma-lfsr sequences," *Ruanjian Xuebao/Journal of Software*, vol. 23, no. 4, pp. 952–961, 2012.

[30] O. Billet and H. Gilbert, "Resistance of snow 2.0 against algebraic attacks," in *Cryptographers' Track at the RSA Conference.* San Francisco, Ca, USA: Springer, February 2005, pp. 19–28.

[31] N. T. Courtois and B. Debraize, "Algebraic description and simultaneous linear approximations of addition in snow 2.0." in *International Conference on Information and Communications Security.* Springer, 2008, pp. 328–344.

[32] D. Watanabe, A. Biryukov, and C. De Canniere, "A distinguishing attack of snow 2.0 with linear masking method," in *International Workshop on Selected Areas in Cryptography.* Springer, 2003, pp. 222–233.

[33] K. Nyberg and J. Wallén, "Improved linear distinguishers for snow 2.0," in *International Workshop on Fast Software Encryption.* Graz, Austria: Springer, March 2006, pp. 144–162.

[34] J.-K. Lee, D. H. Lee, and S. Park, "Cryptanalysis of sosemanuk and snow 2.0 using linear masks," in *International Conference on the Theory and Application of Cryptology and Information Security.* Springer, 2008, pp. 524–538.

[35] B. Zhang, C. Xu, and W. Meier, "Fast correlation attacks over extension fields, large-unit linear approximation and cryptanalysis of snow 2.0," in *Annual Cryptology Conference.* Springer, 2015, pp. 643–662.

[36] G. Leander, E. Zenner, and P. Hawkes, "Cache timing analysis of lfsr-based stream ciphers," in *IMA International Conference on Cryptography and Coding.* Springer, 2009, pp. 433–445.

[37] S. Kiyomoto, T. Tanaka, and K. Sakurai, "K2: a stream cipher algorithm using dynamic feedback control." in *Secrypt*, 2007, pp. 204–213.

[38] S. Ma and J. Guan, "Improved key recovery attacks on simplified version of k2 stream cipher," *The Computer Journal*, vol. 64, no. 8, pp. 1253–1263, 2021.

[39] J. D. Golić, "Cryptanalysis of alleged a5 stream cipher," in *International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 1997, pp. 239–255.

[40] S. Krishnaswamy and H. K. Pillai, "On the number of linear feedback shift registers with a special structure," *IEEE transactions on information theory*, vol. 58, no. 3, pp. 1783–1790, 2011.

[41] I. N. Herstein, *Topics in algebra.* John Wiley & Sons, 1991.

[42] R. Lidl and H. Niederreiter, *Introduction to finite fields and their applications.* Cambridge university press, 1994.

[43] D. S. Dummit and R. M. Foote, *Abstract algebra.* Wiley Hoboken, 2004, vol. 3.

[44] C. E. Shannon, "Communication theory of secrecy systems," *The Bell system technical journal*, vol. 28, no. 4, pp. 656–715, 1949.

[45] J. Zaman and R. Ghosh, "Review on fifteen statistical tests proposed by nist," *Journal of Theoretical Physics and Cryptography*, vol. 1, pp. 18–31, 2012.

[46] S. W. Golomb, *SHIFT REGISTER SEQUENCES: Secure and Limited-Access Code Generators, Efficiency Code Generators, Prescribed Property Generators, Mathematical Models.* World Scientific, 1982.

[47] J. Massey, "Shift-register synthesis and bch decoding," *IEEE transactions on Information Theory*, vol. 15, no. 1, pp. 122–127, 1969.

[48] S. Krishnaswamy and H. K. Pillai, "On the number of special feedback configurations in linear modular systems," *Systems & Control Letters*, vol. 66, pp. 28–34, 2014.

[49] T. W. Cusick and P. Stanica, *Cryptographic Boolean functions and applications.* Academic Press, 2017.

[50] C. Carlet, “Boolean functions for cryptography and error correcting codes,” in *Boolean Methods and Models.* Cambridge, UK: Cambridge Univ. Press, 2007.

[51] S. S. G. OGLU, “Cryptological viewpoint of boolean functions,” Ph.D. dissertation, MIDDLE EAST TECHNICAL UNIVERSITY, 2003.

[52] D. K. Dalai, “On some necessary conditions of boolean functions to resist algebraic attacks,” *Indian Statistical Institute*, 2006.

[53] P. Ekdahl and T. Johansson, “Snow – a new stream cipher,” in *1st NESSIE Workshop*, Heverlee, Belgium, November 2000.

[54] M. Hell, T. Johansson, and L. Brynielsson, “An overview of distinguishing attacks on stream ciphers,” *Cryptography and Communications*, vol. 1, no. 1, pp. 71–94, 2009.

[55] N. T. Courtois and W. Meier, “Algebraic attacks on stream ciphers with linear feedback,” in *Advances in Cryptology—EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003 Proceedings 22.* Springer, 2003, pp. 345–359.

[56] W. Meier, E. Pasalic, and C. Carlet, “Algebraic attacks and decomposition of boolean functions,” in *International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 2004, pp. 474–491.

[57] F. Armknecht and M. Krause, “Algebraic attacks on combiners with memory,” in *Advances in Cryptology-CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings 23.* Springer, 2003, pp. 162–175.

[58] Siegenthaler, “Decrypting a class of stream ciphers using ciphertext only,” *IEEE Transactions on computers*, vol. 100, no. 1, pp. 81–85, 1985.

[59] W. Meier and O. Staffelbach, “Fast correlation attacks on certain stream ciphers,” *Journal of cryptology*, vol. 1, no. 3, pp. 159–176, 1989.

[60] P. Chose, A. Joux, and M. Mitton, “Fast correlation attacks: An algorithmic point of view,” in *Advances in Cryptology—EUROCRYPT 2002: International Conference on the Theory and Applications of Cryptographic Techniques Amsterdam, The Netherlands, April 28–May 2, 2002 Proceedings 21.* Springer, 2002, pp. 209–221.

[61] V. V. Chepyzhov, T. Johansson, and B. Smeets, "A simple algorithm for fast correlation attacks on stream ciphers," in *International Workshop on Fast Software Encryption.* Springer, 2000, pp. 181–195.

[62] D. Wagner, "A generalized birthday problem," in *Annual International Cryptology Conference.* Springer, 2002, pp. 288–304.

[63] Y. Funabiki, Y. Todo, T. Isobe, and M. Morii, "Several milp-aided attacks against snow 2.0," in *International Conference on Cryptology and Network Security.* Springer, 2018, pp. 394–413.

[64] J. Yang, T. Johansson, and A. Maximov, "Vectorized linear approximations for attacks on snow 3g," *IACR Transactions on Symmetric Cryptology*, pp. 249–271, 2019.

[65] M. Matsui, "Linear cryptanalysis method for des cipher," in *Workshop on the Theory and Application of of Cryptographic Techniques.* Springer, 1993, pp. 386–397.

[66] H. Ahmadi and T. Eghlidos, "Heuristic guess-and-determine attacks on stream ciphers," *IET Information Security*, vol. 3, no. 2, pp. 66–73, 2009.

[67] M. S. N. Nia and A. Payandeh, "The new heuristic guess and determine attack on snow 2.0 stream cipher." *IACR Cryptology ePrint Archive*, vol. 2014, p. 619, 2014.

[68] B. B. Brumley, R. M. Hakala, K. Nyberg, and S. Sovio, "Consecutive s-box lookups: A timing attack on snow 3g," in *International conference on information and communications security.* Springer, 2010, pp. 171–185.

[69] A. Maximov and T. Johansson, "Fast computation of large distributions and its cryptographic applications," in *International Conference on the Theory and Application of Cryptology and Information Security.* Springer, 2005, pp. 313–332.

[70] Y. Todo, T. Isobe, W. Meier, K. Aoki, and B. Zhang, "Fast correlation attack revisited," in *Annual International Cryptology Conference.* Springer, 2018, pp. 129–159.

[71] J. Liesen and Z. Strakos, *Krylov subspace methods: principles and analysis.* Numerical Mathematics and Scie, 2013.

[72] F. Jönsson and T. Johansson, "Correlation attacks on stream ciphers over gf (2ˆ n)," in *IEEE International Symposium on Information Theory (ISIT), 2001*, 2001, pp. 140–140.

[73] E. Yılmaz, "Two versions of the stream cipher snow," Master's thesis, Middle East Technical University, 2004.

[74] S. W. Golomb and G. Gong, *Signal design for good correlation: for wireless communication, cryptography, and radar.* Cambridge University Press, 2005.

[75] E. Teske, "The pohlig–hellman method generalized for group structure computation," *Journal of symbolic computation*, vol. 27, no. 6, pp. 521–534, 1999.

[76] C.-L. Wu, D.-C. Lou, and T.-J. Chang, "An efficient montgomery exponentiation algorithm for public-key cryptosystems," in *2008 IEEE International Conference on Intelligence and Security Informatics*, 2008, pp. 284–285.

[77] M. S. N. Nia and T. Eghlidos, "Improved heuristic guess and determine attack on snow 3g stream cipher," in *7'th International Symposium on Telecommunications (IST'2014).* Tehran, Iran: IEEE, September 2014, pp. 972–976.

[78] S. Nandi, S. Krishnaswamy, B. Zolfaghari, and P. Mitra, "Key-dependent feedback configuration matrix of primitive σ–lfsr and resistance to some known plaintext attacks," *IEEE Access*, vol. 10, pp. 44 840–44 854, 2022.

[79] B. Debraize and I. Marquez Corbella, "Fault analysis of the stream cipher snow 3g," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC).* Lausanne, Switzerland: IEEE, September 2009, pp. 105–112.