A NEW MODULO ADDITION WITH ENHANCED SCALABLE SECURITY AGAINST ALGEBRAIC ATTACK

By

Min Hsuan Cheng

Bachelors of Applied Science

Electrical Engineering

University of Toronto

Toronto, Canada, 2012

A Project

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Engineering

in the Program of

Electrical and Computer Engineering

Toronto, Ontario, Canada, 2016

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A PROJECT

I hereby declare that I am the sole author of this Project. This is a true copy of the Project, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this Project or dissertation to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this Project by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my Project may be made electronically available to the public.

ABSTRACT

Title of Project:

A new Modulo Addition with enhanced scalable security against Algebraic Attack

Project Submitted By:

Min Hsuan Cheng, Master of Engineering, 2016

Optimization Problems Research and Application Laboratory (OPR-AL)

Electrical and Computer Engineering Department, Ryerson University

In recent years, Algebraic Attack has emerged to be an important cryptanalysis method in evaluating encryption algorithms. The attack exploits algebraic equations between the inputs and outputs of a cipher to solve for the targeted information. The complexity of the attack depends on the algebraic degree of the equations, the number of equations, and the probabilistic conditions employed. Addition Modulo 2^n had been suggested over logic XOR as a mixing element to better defend against Algebraic Attack. However, it has been discovered that the complexity of the traditional Modulo Addition can be greatly reduced with the right equations and probabilistic conditions. The presented work introduces a new Modulo Addition structure that includes an Input Expansion, Modulo Addition, and Output Compaction. The security of the new structure is scalable and user-defined as the new structure increases the algebraic degree and thwarts the probabilistic conditions.

ACKNOWLEDGEMENT

I would like to acknowledge and thank my supervisor, Dr. Reza Sedaghat, for his guidance and critique throughout this research work

I would also like to thank Patrick Siddavaatam and the members in the lab for their help and support during the time of research

Finally, I would like to thank Abby Chen, my family, and my friends, who have always been there for me.

Table of Contents

List of Tables

List of Figures

Nomenclature

Plaintext	Unencrypted input to a cipher
Ciphertext	Encrypted output from a cipher
S-Box	Substitution Box
$\oplus$	XOR Gate
M	User-defined Control Parameter
n	Data width of the design
A.I.	Algebraic Immunity
D.D.	Describing Degree
LFSR	Linear Feedback Shift Registers
SPN	Substitution Permutation Network
D	Algebraic Degree
T	Number of Monomials of Degree D
R	Number of Equations
Γ	Complexity of solving a system of equations
$\boxplus$	Addition Modulo 2^n
KI	Input Control Bit String
KO	Output Control Bit String
Pr()	Probability of ()
$\{0,1\}^n$	N-bit Binary String
$\{0,1\}^n \rightarrow \{0,1\}^m$	Transformation of N-bit Binary String to M-bit Binary String
ANF	Algebraic Normal Form
K	Secret Key
IV	Initialization Vector

1 Introduction

1.1 Overview

In the modern world, security plays a significant role in providing privacy in various activities in the society, such as: communications and storage of information. A piece of Information is secure when it can keep its privacy even in the hands of attackers. The study of how to keep information secure is then termed cryptography. The focus of cryptography is a broad research of various encryption schemes and functions. These schemes and functions provide different algorithms to encrypt, or hide, the targeted information. At the same time, they also present the corresponding algorithms to decrypt, or expose, the encrypted information. In general, cryptography can be broken down into three areas of studies: Symmetric Key Cryptography, Asymmetric Key Cryptography, and Hash Functions. In addition, a Protocol is a complex scheme that involves one or more of these areas in cryptography.

Symmetric Key Cryptography, which is also referred to as Secret Key Cryptography, is a type of encryption algorithm that generally involves the use of a common key, shared between authorized parties. In other words, the key is kept secret to other non-involved parties and is only known to the involved parties. Also, the key is pre-determined by both parties without exposing the key to the public. The encryption algorithm, or cipher, in symmetric key cryptography can be generalized to two types: Block cipher and Stream cipher.

Asymmetric Key Cryptography, which is also referred to as Public Key Cryptography, is a type of encryption algorithm that generally involves the generation and exchange of a key or a pair of keys between the authorized parties. Some of the process or parameters in generating the keys are known to the public; however, the scheme still provides security. Two of the most prominent algorithms in this area are RSA and Diffie-Hellman.

Hash Functions, unlike the other two areas of cryptography, is an algorithm that provides uniqueness to a piece of information, or a fingerprint. It is not used as an encryption algorithm since it does not provide a corresponding decryption method.

However, it is often used in conjunction with Symmetric Key or Asymmetric Key cryptographic algorithms to form a security protocol.

Cryptanalysis, in contrast, is a study of how to break or attack cryptographic algorithms. While newly proposed cryptographic algorithms can seem to be very complex and sound, cryptanalysis presents itself as an evaluation metric for the algorithms, in both theory and application. In fact, the evaluation of a cryptographic algorithm is always continuous, until new and better cryptanalysis methods prove the insecurity of the algorithm. Similar to cryptography, the study of cryptanalysis can be generally broken down to the same areas. However, it is possible that certain cryptanalysis algorithm can be extended to different areas.

1.2 Motivation

In this Project, a new cryptographic module is proposed. The new design creates a structure of expansion and compaction to encompass one of the elementary cryptographic modules, Modulo Addition. The structure is also scalable, as it can be customized to fit different algorithms. At the same time, the content of the structure can be substituted to more suitable functions if necessary. The resultant design is able to provide enhanced and scalable security against one of the cryptanalysis method – Algebraic Attack, and can be used as a building block in Symmetric Key Cryptography.

1.3 Related Work

Cryptographic functions that can improve security against Algebraic Attack have been studied over the years. Traditional Modulo Addition has been suggested over XOR to be used against Algebraic Attack in the field of cryptography. The effectiveness of using Modulo Addition to thwart Algebraic Attack has also been discussed in [6]. At the same time, Boolean functions that have high Algebraic Immunity have been studied in [1], [2], and [3]. S-Boxes that can guard against Algebraic Attack have also been studied in [4] and [5]. In contrast, the structure of expansion and compaction has not been studied as much. The form of expansion can be found in S-Boxes that provide more output bits than input bits. This type of S-Boxes has been studied in [7]. If the proposed design is

viewed like an S-Box as a whole, it resembles the type of S-Box that is key dependent. This type of S-Box has been seen in [22], [29], [30], [31], and [32], but same structure has not been seen.

1.4 Report Organization

The report is organized as the following: Chapter 2 provides preliminary concepts required to build up the proposed design. Chapter 3 discusses in detail the proposed design. Then, Chapter 4 presents the analysis of the new design while Chapter 5 demonstrates the applications of the proposed design in stream ciphers. Chapter 6 shows the results and analysis of the applications in Chapter 5 and discusses the limitations of the design. In Chapter 7, the usage of the proposed design in block cipher is discussed. Finally, Chapter 8 provides the conclusion and future possibilities.

2 Preliminaries

In Chapter 2, the fundamentals of stream ciphers and block ciphers are given. Then, Algebraic Attack is explained in detail. The cryptographic component Addition Modulo 2^n is also explained with respect to Algebraic Attack.

2.1 Stream Ciphers

Stream cipher is a symmetric key cipher that encrypts information one piece at a time, typically one bit. The algorithm utilizes the secret key to generate a stream of key bits, or key stream, and adds the key stream to the input information, typically referred to as plaintext, to produce the encrypted output, typically called the ciphertext. To decrypt the ciphertext back to plaintext, the receiving party will use the same symmetric key with the same encryption algorithm, generate the same set of key stream, and add the key stream to the ciphertext. This set of operations work because the "addition" involved is performed as addition modulo 2, or XOR logic. Table 1 provides the truth table of the 2-input XOR logic, and Figure 1 demonstrates the stream cipher scheme.

X	**Y**	**Z = X ⊕ Y**
0	0	0
0	1	1
1	0	1
1	1	0

Table 1 - Truth Table of XOR logic

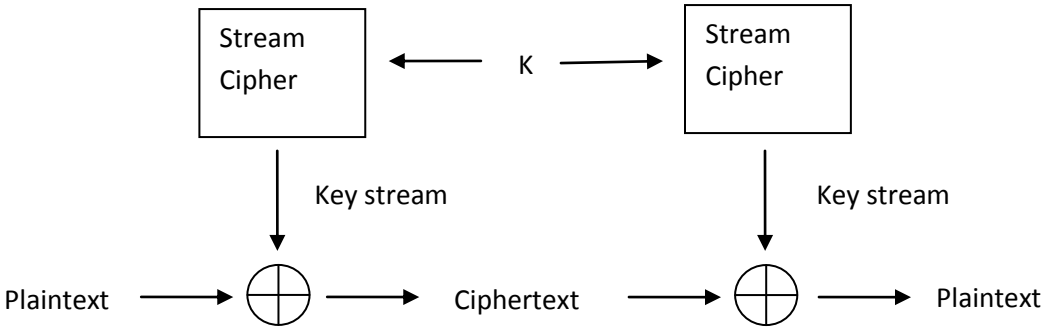

Figure 1 - Stream Cipher Scheme

The security of a stream cipher is directly related to the security of the generated key stream. In general, the stream cipher can be seen as a pseudo random number generator with security. In other words, the next bit of the key stream cannot be predicted given the previous bits of the key stream when the secret key is unknown. The only way to create the same key stream is to obtain the secret key. Therefore, a secured stream cipher can be broken only by searching for the secret key.

A typical stream cipher can be realized in the form of Linear Feedback Shift Registers (LFSR) but it is not secure because its period is predictable and breakable by algorithms such as Berlekamp-Massey algorithm [8]. As a result, the development of stream ciphers involves creative alterations to the LFSR. For example, a designer can use multiple LFSRs and combine the outputs to generate a key stream, or the clocking of the LFSR can be irregular. A famous stream cipher that utilizes the two concepts is A5 [9]. The secret key is used to initialize, typically along with another Initialization Vector, the bits or states in the LFSR. This way, the generated key stream is unique and can only be decrypted by the same key

Many cryptanalysis methods exist for deciphering a stream cipher. The most basic attack is the Brute Force attack, which simply guesses the contents of the secret key. This attack is guarded by increasing the length of the secret key, which increases the

number of combinations an attacker needs to guess. Time-Memory Trade-off attack is another common cryptanalysis method on stream ciphers. An attacker needs to pre-compute and store fragments of key stream generated from certain states in the cipher, and compares the received key stream against the pre-computed ones. Once a match, or collision, happens, the secret state of the stream cipher is discovered. Fast Correlation attack is also a common attack. The cryptanalysis looks for correlation between the key stream and the LFSR bits. If the correlation is high, it is likely that the LFSR bit is equal to the key stream. Finally, the Distinguishing attack tries to distinguish the key stream from a random sequence and looks for biases in the key stream. If the key stream has a heavy bias towards 1 or 0, the key stream can be guessed [10].

2.2 Block Ciphers

Block cipher is another type of Symmetric Key Cryptography. Unlike stream ciphers, block ciphers process multiple bits of the plaintext, or a block of the plaintext, at once. The ciphertext is produced in blocks as well. To decrypt, the same secret key is used and the block cipher is run in reverse. Figure 2 illustrates the encryption and decryption operations.

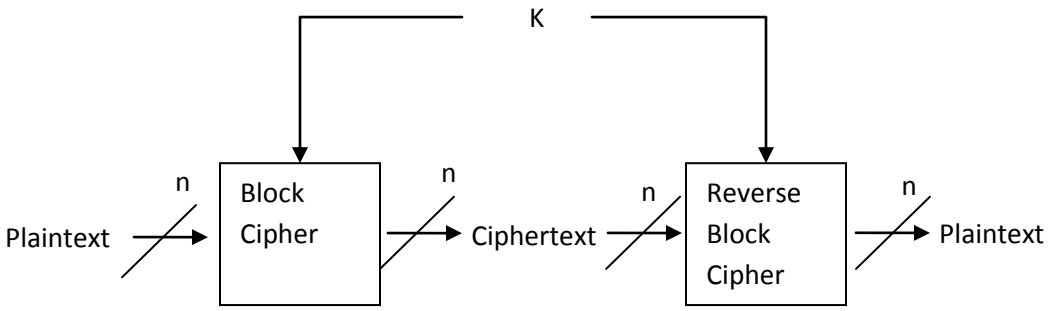

Figure 2 - Block Cipher Scheme

The security of the block cipher lies in the difference between the plaintext and the ciphertext. The block cipher is secure only if the difference can be known by guessing

the secret key. The design of the block cipher is slightly more involved than the stream cipher as the structure needs to be reversible. A block cipher can be constructed typically by utilizing one of the two structures: a Feistel Network or a Substitution Permutation Network (SPN). However, the security is provided by the series of operations that performs Substitution and Permutation. The series of operations is referred to as the round function. Substitution provides security by substituting one piece of information to another. Permutation provides security by swapping elements within a piece of information. Placing the round function in one of the two structures above will result in a block cipher. In general, a Feistel Network divides the input by half, feeds one half into the round function in the current round, and feeds the other half into the round function in the next round. The whole input will experience the round function as many times as the designer specified. On other hand, a SPN feeds the whole input into its round function in one round and feeds the result of the previous round into the round function again. Figure 3 illustrates the two structures.

It is worth noting that, the position of and the operation on the key varies according to different designs. In fact, a key schedule is generally required to derive round keys from the secret key. The round keys can be mixed, typically through XOR logic, with the input or the results from one round, before going into the next round. The mixing of the secret key makes the ciphertext unique such that the same key needs to be used for decryption.

Similar to stream cipher, many cryptanalysis methods for block cipher exist. Other than the Brute Force method, Linear cryptanalysis [11] and Differential cryptanalysis [12] are the most popular. Linear cryptanalysis uses an input mask and an output mask to try to uncover correlation between the plaintext and the ciphertext. If a pair of mask is found to have high probability, then this pair of mask can be used to guess the certain bits in the secret key. Differential cryptanalysis attempts to find plaintext pairs with a certain difference that have a high probability of generating ciphertext pairs with a certain difference. The difference can be used to uncover some of the secret key bits.

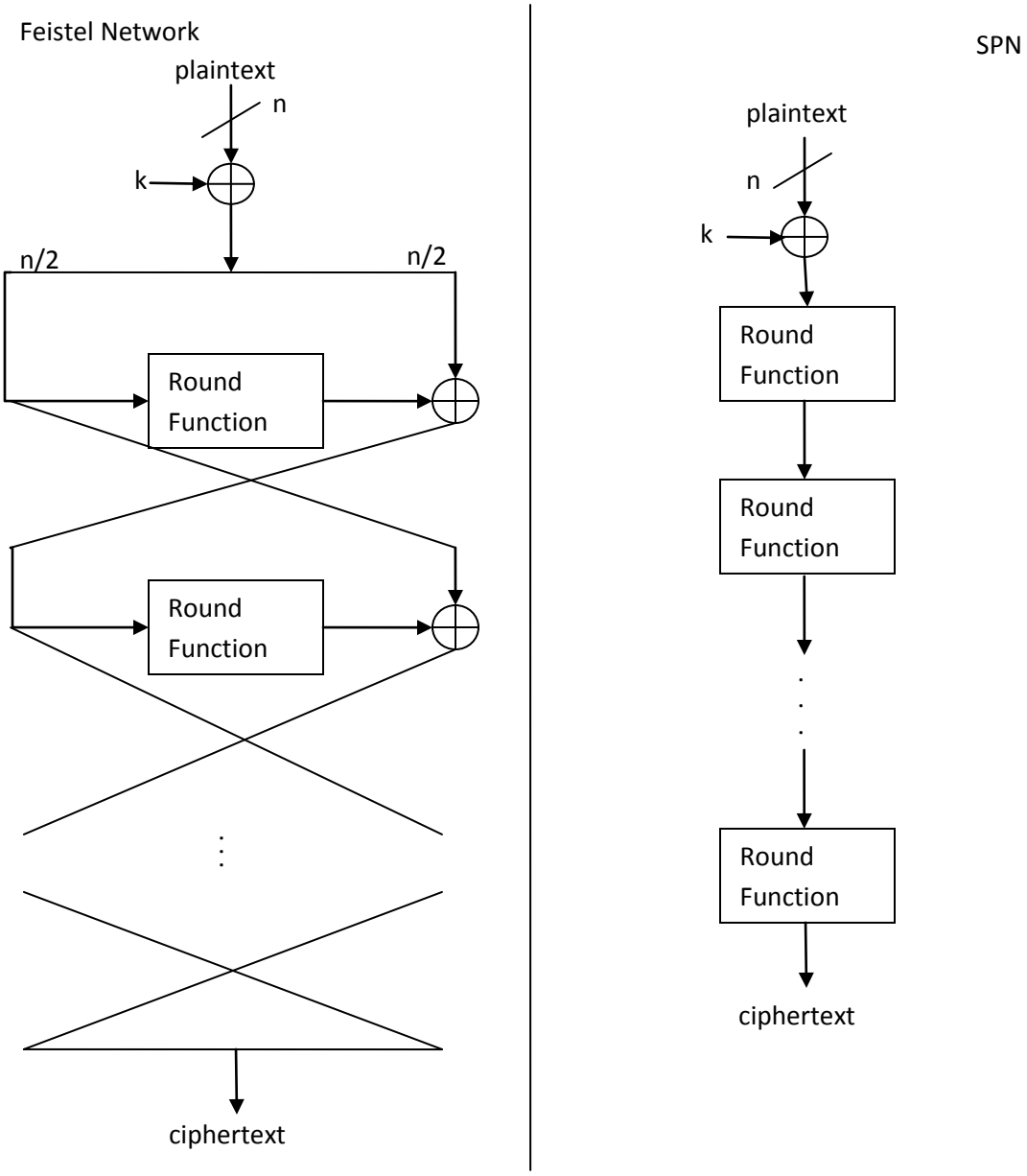

Figure 3 - Generic Feistel Network and SPN

2.3 Algebraic Attack

Algebraic Attack was first introduced in [13] and [14] to break public key scheme and later applied on stream ciphers and block ciphers [15], [16], and [17]. The attack focuses on formulating multivariate polynomial equations between the inputs and outputs with low algebraic degree. Multivariate polynomial equations are polynomial equations with more than one variable. Monomials are terms in the equations made up of variables of

various degrees. The algebraic degree is the largest degree a monomial has in a multivariate polynomial equation. The biggest difference between the Algebraic Attack and other traditional attacks, such as linear and differential cryptanalysis, is that the formulae between the inputs and the outputs are true with probability close to or equal to one. In other words, traditional cryptanalysis methods are probabilistic. As a result, the attacker would only need to solve the equations, given enough samples, to completely decipher an encryption algorithm. In recent years, techniques of solving multivariate polynomial equations have become an active area of research [41] [17] [42] [43]. However, it is possible for an attacker to estimate the complexity of solving the system of equations and this will be shown later in this section. Two issues arise when an attacker attempts to solve the system of equations. First, in a system of multivariate equations, it is common to have more variables than equations. As a counter, an attacker would try to uncover equations that use the same set of variables but are independent from the existing set of equations. The result of this provides the attacker an overdefined system of equations and the attacker is able to solve the system. The second issue is related to the complexity of solving the equations. The system of equations is more difficult to solve when the algebraic degree of the equations becomes higher and when the equations become denser. A dense equation has many monomials whereas a sparse equation uses less terms. An experienced attacker may need an additional step before attempting to solve the system of equations. The step involves multiplying carefully chosen variables to the existing equations to create lower-degree additional independent equations. In general, Algebraic Attack can be summarized in three steps:

1. Formulate equations between inputs and outputs
2. Create additional independent equations with lower algebraic degree
3. Solve the system of equations

The advancement of Algebraic Attack has attracted a lot of attention in the cryptographic community. As in traditional cryptanalysis methods, a metric has been suggested to define the resistance of a cryptographic module against Algebraic Attack.

DEFINITION 1: Algebraic Immunity is defined as the minimum algebraic degree in the system of equations [18].

Even though the Algebraic Immunity is not completely sufficient to describe the resistance against Algebraic Attack, as discussed in [19], it nevertheless provides a good indication. Various have been done to develop Boolean functions that provide high Algebraic Immunity as discussed in Chapter 1, so that these functions can be utilized in creating secure cryptographic components.

The complexity of solving Algebraic Attack has been explored in [15] and [17]. It can be generalized into a linearization problem with Gaussian reduction. For a generic stream cipher that consists of one or more known Linear Feedback Shift Registers (LFSR) and a known nonlinear output combiner function, R multivariate equations can be formed between the output bit and the states in the LFSR. Let N be the number of variables in the states and D be the algebraic degree of the equations. Then, the number of monomials of degree ≤ D can be calculated as:

$$T = \sum_{i=0}^{D} \binom{N}{D}$$

The complexity of solving this system is directly related to T and the number of samples required equals to the number of equations, R. Therefore, to form an overdefined system of equations, R > T samples are required.

In addition, the calculation of complexity is slightly different in the case of a block cipher. The round function of a generic block cipher contains a nonlinear Substitution Box (S-Box) and one or more layers of linear permutations. As the permutations are typically linear, the equations of interest lie in the S-Box. In particular, an S-Box

transforms its input variables $X = x_1, \dots, x_n$ to output variables $Y = y_1, \dots, y_m$, using a vectorial Boolean function. The number of monomials with an algebraic degree ≤ D in the set of equations that describe the function can be calculated as:

$$T = \sum_{i=0}^{D} \binom{m+n}{i}$$

The number of equations is determined by forming a matrix M of size $2^n * T$, in which the rows are all the input combinations and the columns are the monomials. The rank of the matrix M is the smaller of the row and column dimensions and is typically 2^n. Finally, the number of equations R is:

$$T - rank\ of\ Matrix$$

The complexity of solving this system of equations is defined as:

$$\Gamma = {}^{T}/_{n}^{\lceil T/R \rceil}$$

As mentioned before, the advantage of Algebraic Attack is that the equations formulated have a probability equal or close to one. The equations with probabilities close to one are referred to as conditional equations [20]. These equations, in practice, are formulated by the experienced attacker while analyzing the specific cipher. The conditional equations are beneficial to the attacker because the number of equations has increased and the complexity of solving has decreased. Evidently, the cost of these equations is the multiplication of the complexity and the probability.

2.4 Modulo Addition 2^n

In cryptography, Addition is typically carried out with modulo 2^n, and is used for "Mixing", which mixes information from one source into the other by the means of summation. It is widely used in both block ciphers, such as CAST [21], TWOFISH [22], and MARS [23]; and stream ciphers, such as SOBER-t32 [24], SNOW2.0 [25], and ZUC [26]. Though an elementary operation, Modulo Addition offers better security than a XOR operation because it is partly nonlinear. XOR is a linear operation because it is a Boolean

function with inputs that are assumed to have algebraic degrees of 1. In contrast, Modulo Addition is partly nonlinear because of the potential of having a carry. Also, each sum bit is considered. This can be seen from the general equations described in [6] and below. The symbol + is used to denote addition in GF(2), which means XOR, and the symbol $\boxplus$ is used to denote Modulo Addition 2^n. It should be noted that in cryptography, a single Modulo Addition typically has no carry-in.

$$Z = X \boxplus Y:$$

$$\begin{cases} Z_0 = X_0 + Y_0 \\ Z_1 = X_1 + Y_1 + C_0 \\ \vdots \\ Z_i = X_i + Y_i + C_{i-1} \\ \vdots \\ Z_{n-1} = X_{n-1} + Y_{n-1} + C_{n-2} \end{cases} \quad (1)$$

$$\begin{cases} C_0 = X_0 Y_0 \\ C_1 = X_1 Y_1 + (X_1 + Y_1)(X_0 Y_0) \\ C_2 = X_2 Y_2 + (X_2 + Y_2)(X_1 Y_1) + (X_2 + Y_2)(X_1 + Y_1)(X_0 Y_0) \\ \vdots \\ C_i = X_i Y_i + (X_i + Y_i)(X_{i-1} Y_{i-1}) + \sum_{p=0}^{i-2} X_p Y_p \prod_{q=p+1}^{i} (X_q + Y_q) \end{cases} \quad (2)$$

$$2 \leq i \leq n-2$$

One can view the Modulo Addition as an S-Box, with input variables X and Y and output variable Z, and evaluate its Algebraic Immunity, using Definition 1. The minimum degree of this set of equation is 1; however, if one substitutes the carry variables with X and Y variables, the algebraic degree is much more than 1. The author in [19] has made

this observation and come up with another definition to better describe the immunity of this type of cryptographic component.

DEFINITION 2: The Describing Degree is the minimum degree D such that the S-Box is entirely defined by equations of degree at most D.

Then, the author developed a new set of equations that defines the Modulo Addition with a Describing Degree of 2. In other words, the Modulo Addition can be defined by set of quadratic equations as in (3).

$$\begin{gathered} Z_0 = X_0 + Y_0 \\ Z_1 = X_1 + Y_1 + X_0 Y_0 \\ Z_2 = X_2 + Y_2 + X_1 Y_1 + (X_1 + Y_1)(X_1 + Y_1 + Z_1) \\ \vdots \\ Z_i = X_i + Y_i + X_{i-1} Y_{i-1} + (X_{i-1} + Y_{i-1})(X_{i-1} + Y_{i-1} + Z_{i-1}) \\ \vdots \\ Z_{n-1} = X_{n-1} + Y_{n-1} + X_{n-1} Y_{n-1} + (X_{n-1} + Y_{n-1})(X_{n-1} + Y_{n-1} + Z_{n-1}) \end{gathered} \tag{3}$$

Furthermore, the author is able to create additional independent equations and the number of equations has totaled to $6n - 3$. The complexity of solving this "S-Box" can then be calculated using the method described in the previous section. An example calculation for Addition Modulo 2^{32} is shown here:

$$\mathrm{T} = \sum_{\mathrm{i}=0}^{2} \binom{3 * 32}{\mathrm{i}} = 553$$

$$\mathrm{R} = 6 * 32 - 3 = 189$$

$$\Gamma = {}^{T}\!/_{2n}{}^{\lceil T/R \rceil} = {}^{553}\!/_{64}{}^{\lceil 553/189 \rceil} = {}^{553}\!/_{64}{}^{3} = 645 \approx 2^9$$

It is worth noting that odd-number Modulo Addition is not considered in this report.

2.5 Conditional Properties of Modulo Addition

Conditional equations for generic S-Box are studied in [20] and the idea can be applied in a similar manner to Modulo Addition. The goal of using conditional equations in Algebraic Attack as mentioned before is to either lower the algebraic degree of the equations or create more independent equations. Both can be achieved by exploiting the carry bits in the Modulo Addition. As a result, the cost is directly related to the probability of carries in a traditional Modulo Addition. The probability of carries can be generalized in (4) and is also demonstrated in [6]. As the number of bits increases in the addition, the probability of a carry occurring approaches $\frac{1}{2}$. Note that the inputs are assumed to be uniformly distributed.

$$C = (C_n, \dots, C_2, C_1)$$

$$\begin{cases} \Pr(C_1 = 1) = \frac{1}{2} * \frac{1}{2} = \frac{1}{4} \; ; \; X_0 = Y_0 = 1 \\ \Pr(C_1 = 0) = 1 - \Pr(C_1) = \frac{3}{4} \end{cases}$$

$$\begin{cases} \Pr(C_2 = 1) = \frac{1}{2} * \frac{1}{2} * \frac{1}{4} + \frac{1}{2} * \frac{1}{2} * \frac{3}{4} + \frac{1}{2} * \frac{1}{4} * \frac{1}{2} + \frac{1}{2} * \frac{1}{4} * \frac{1}{2} = \frac{3}{8} \; ; \begin{cases} X_1 = Y_1 = C_1 = 1 \\ X_1 = Y_1 = 1 \, , C_1 = 0 \\ X_1 = C_1 = 1 \, , Y_1 = 0 \\ Y_1 = C_1 = 1 \, , X_1 = 0 \end{cases} \\ \Pr(\mathrm{C}_2 = 0) = 1 - \Pr(\mathrm{C}_2 = 1) = \frac{5}{8} \end{cases}$$

$$\begin{cases} \Pr(C_i = 1) = {}^{2^i - 1}/_{2^{i+1}} \\ \Pr(C_i = 0) = 1 - \Pr(C_i = 1) \end{cases} \quad (4)$$

2.5.1 Condition 1: No carries

The conditional properties of Modulo Addition are first explored in [27], and then expanded in [19]. The simplest condition is for the addition to have no carries at all, and this will completely linearize the equations in (3), as each output variable becomes the addition of the two input variables in GF(2). In other words, the linearized equations hold with probability equal to 1 and the complexity of solving the equations is reduced.

One can also treat these linearized equations as new conditional equations added on top of the existing equations in (3). As the number of equations increases, the solving complexity decreases. This approach can be applied to other conditional properties that will be described below. At the same time, the attacker needs to consider the cost of this condition happening and that is the probability of all carries being equal to 0. For a 32-bit Modulo Addition, this probability can be calculated using (4) and is equal to $0.00000000055 \approx 2^{-31}$.

2.5.2 Condition 2: Output Characteristics

A less obvious condition utilizes the output characteristic of Modulo Addition, that is, when the output is $2^n - 1$, or in binary terms, all 1's. This characteristic is convenient because, through deduction, no carry exists in this scenario when the carry-in is 0. Here, the equations are again linearized with probability 1. The associated cost of this occurring is when the two inputs are of opposite polarity, meaning they are either 0 and 1 or 1 and 0. In digital logic, this input pairing is referred to as Propagate [28]. The table below lists out the terms describing different input pairings.

Input 1	Input 2	Term
0	0	Kill
0	1	Propagate
1	0	Propagate
1	1	Generate

Table 2 - Terms for Input Pairings

The probability of all input pairs being Propagate for an n-bit addition is:

$$(\frac{1}{2} * \frac{1}{2} + \frac{1}{2} * \frac{1}{2})^n = 2^{-n}$$

A natural expansion of this output characteristic is to have K consecutive 1's in the least significant bits in the output. This will linearize at least K equations and potentially

linearize the equations describing the remaining $n - K$ bits. Evidently, this potential also has an associated cost, or probability.

2.5.3 Condition 3: Input Characteristics

Moreover, there are three input characteristics one can utilize to help linearize the equations. First, the output equals to one of the inputs when the other input is simply 0. Second, the output equals to one of the input $-\ 1$ when the other input is $2^n - 1$. The last characteristic requires one of the inputs to be the Two's Complement of the other input. Two's Complement form is regularly used to in digital logic to perform subtraction and it transforms a number by complementing the number in the binary form and then adding the binary form by 1. For example:

$$X = (4)_{10} = (100)_2,\ Two's\ Complement\ of\ X = (X') + 1 = (011)_2 + \ 1 = (100)_2.$$

The output of this input pairing in Modulo Addition will always be 0. The distribution of carry bits in this scenario is as follows: There will be no carries from the least significant bits of the output to the first (1, 1) pair in the inputs. The rest of bits in the output all have carries. Therefore, if one of the inputs only has its most significant bit as a 1 and the other input is the former's Two's Complement, the sum from the Modulo Addition will generate no carries. All of the input characteristics discussed above can help linearize the equations and create more independent equations. Also, they all have the same cost, which is a probability of 2^{-n}.

The proposed design will aim to increase the difficulty to utilize these conditional properties of Modulo Addition and the complexity of solving the equations.

3 A New Addition Modulo 2^n

The proposed design is a new type of cryptographic component that provides user-defined scalable security against Algebraic Attack. Figure 4 depicts the general block diagram of the design. The 3 components include: Input Expansion, Modulo Addition, and Output Compaction. Figure 5 specifies the symbol that will be used to describe the new design.

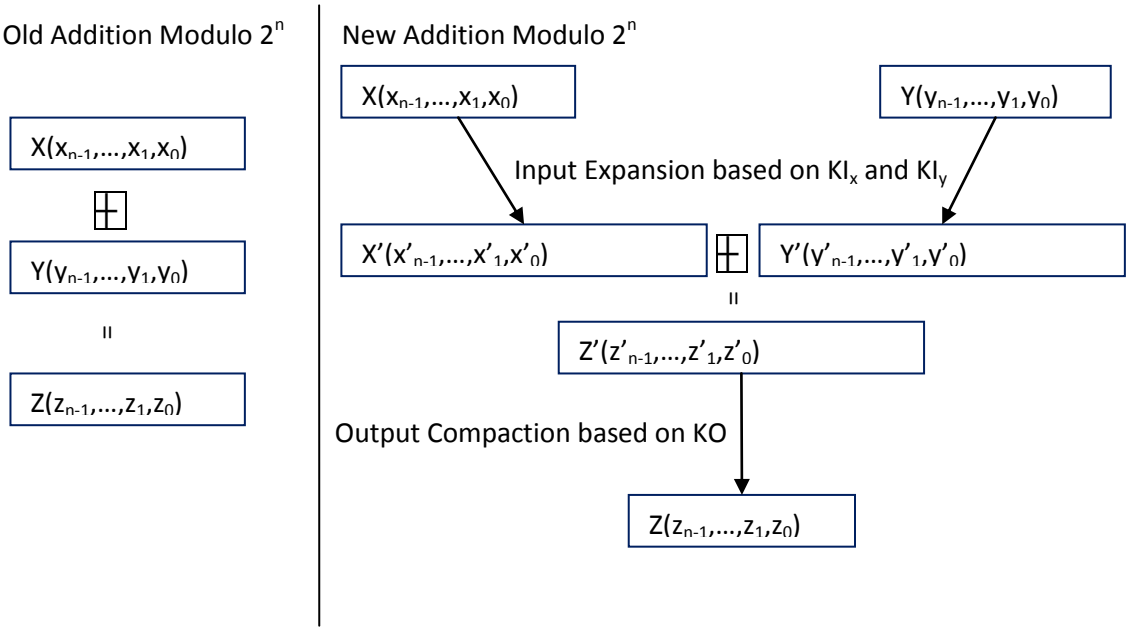

Figure 4 - Block diagram of old and new Modulo Addition

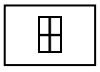

Figure 5 - Symbol of the new design

3.1 Input Expansion

The Input Expansion, $F_{IN}()$, is a function that expands a single input bit into a 2^m-bit string based on an n*m-bit control string KI. In this case, n specifies the number of bits in the Modulo Addition while m is a user defined parameter. Typically, the n*m-bit control string is generated from the internals of a cipher. As the parameter m increases,

the expanded input becomes longer. The actual expansion function is flexible; meaning, the relationship between the input and the expanded input can also be user defined. For example, the relationship can be some arithmetic operation that grows 1 bit into 2^m bits, or it can be a collection of Boolean functions, like an S-Box. Listed below is general description of input and output variables before and after applying the Input Expansion function.

$$Let\ X = (x_{n-1}, \dots, x_1, x_0)\ and\ Y = (y_{n-1}, \dots, y_1, y_0)$$

$$Let\ KI_X = (KI_{Xn-1}, \dots, KI_{X1}, KI_{X0})\ and\ KI_Y = (KI_{Yn-1}, \dots, KI_{Y1}, KI_{Y0})$$

$$Let\ KI_{Xi} = \left(KI_{Xi,m-1}, \dots, KI_{Xi,1}, KI_{Xi,0}\right)\ and\ KI_{Yi} = \left(KI_{Yi,m-1}, \dots, KI_{Yi,1}, KI_{Yi,0}\right),$$

$$KI_{Xi,j}, KI_{Yi,j}\ \in \{0,1\}^m\ ;\ 0\ \leq i\ \leq n-1\ ;0\ \leq \mathrm{j}\ \leq \mathrm{m}-1$$

$$Let\ X' = (x'_{n-1}, \dots, x'_1, x'_0) = \left(F_{IN}(x_{n-1}, KI_{Xn-1}), \dots, F_{IN}(x_1, KI_{X1}), F_{IN}(x_0, KI_{X0})\right)$$

$$= \left(x'_{(n-1)(w-1)}, \dots, x'_{(n-1)1}, x'_{(n-1)0}, \dots, x'_{1(w-1)}, \dots, x'_{11}, x'_{10}, x'_{0(w-1)}, \dots, x'_{01}, x'_{00}\right)$$

$$Let\ Y' = \left(y'_{n-1}, \dots, y'_1, y'_0\right) = (F_{IN}(y_{n-1}, KI_{Yn-1}), \dots, F_{IN}(y_1, KI_{Y1}), F_{IN}(y, KI_{Y0}))$$

$$= \left(y'_{(n-1)(w-1)}, \dots, y'_{(n-1)1}, y'_{(n-1)0}, \dots, y'_{1(w-1)}, \dots, y'_{11}, y'_{10}, y'_{0(w-1)}, \dots, y'_{01}, y'_{00}\right)$$

$$w = 2^m$$

The equation below provides an example of the function F_{IN}. This equation is chosen to give an arithmetic relationship between the inputs and output so that it is easily scalable. As mentioned before, this function can be substituted with other user-defined functions. Table 2 demonstrates the truth table when the user defines m to be 3.

$$x'_i = F_{IN}(x_i, KI_{xi}) = \begin{cases} 2^w - 1 - 2^{KI_{xi}}; for\ x_i = 0 \\ 2^{KI_{xi}}; for\ x_i = 1 \end{cases};\ KI_{xi} is\ in\ decimal\ form \quad (5)$$

x_i	KI_{Xi}			x'_{i7}	x'_{i6}	x'_{i5}	x'_{i4}	x'_{i3}	x'_{i2}	x'_{i1}	x'_{i0}
	KI_{Xi2}	KI_{Xi1}	KI_{Xi0}								

x_i	KI_{Xi}			x'_{i7}	x'_{i6}	x'_{i5}	x'_{i4}	x'_{i3}	x'_{i2}	x'_{i1}	x'_{i0}
	KI_{Xi2}	KI_{Xi1}	KI_{Xi0}								
0	0	0	0	1	1	1	1	1	1	1	0
0	0	0	1	1	1	1	1	1	1	0	1
0	0	1	0	1	1	1	1	1	0	1	1
0	0	1	1	1	1	1	1	0	1	1	1
0	1	0	0	1	1	1	0	1	1	1	1
0	1	0	1	1	1	0	1	1	1	1	1
0	1	1	0	1	0	1	1	1	1	1	1
0	1	1	1	0	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Table 3 - Truth table of (5) when M = 3

When using (5) as the Input Expansion function, it is suggested to set m ≥ 2 because when $m = 1$, the expanded input will repeat itself.

3.2 Addition Modulo 2^{n*w}

The second component of the design takes the expanded inputs and performs Modulo Addition. Nevertheless, the number of additions has increased as the inputs are expanded from $\{0,1\}^n \rightarrow \{0,1\}^{n*w}$. The addition still follows the formulae in (1), (2), and (3). The sum can be generalized by the equation similar to (3) as below.

$$Z' = X' \boxplus Y'$$

$$Let\ Z' = (z'_{n-1}, \dots, z'_1, z'_0)$$

$$= \left(z'_{(n-1)(w-1)}, \dots, z'_{(n-1)1}, z'_{(n-1)0}, \dots, z'_{1(w-1)}, \dots, z'_{11}, z'_{10}, z'_{0(w-1)}, \dots, z'_{01}, z'_{00}\right)$$

$$w = 2^m$$

$$\begin{gathered} z'_{00} = x'_{00} + y'_{00} \\ z'_{01} = x'_{01} + y'_{01} + x'_{00}y'_{00} \\ z'_{02} = x'_{02} + y'_{02} + x'_{01}y'_{01} + \left(x'_{01} + y'_{01}\right)\left(x'_{01} + y'_{01} + z'_{01}\right) \\ \vdots \\ z'_{ij} = x'_{ij} + y'_{ij} + x'_{ij-1}y'_{ij-1} + \left(x'_{ij-1} + y'_{ij-1}\right)\left(x'_{ij-1} + y'_{ij-1} + z'_{ij-1}\right) \\ 0 \le i \le n-1\ ;\ 0 \le j \le w-1 \end{gathered} \quad (6)$$

It is worth noting that the carry-out generated by each output block, such as z'_0, becomes the carry-in of the next block, such as z'_1.

3.3 Output Compaction

The final component is a function that compresses Z' to Z, i.e. $\{0,1\}^{n*w} \rightarrow \{0,1\}^n$, based on a $n * m$-bit output control string KO. In particular, the Output Compaction uses a 2^m:1 MUX and the select lines are provided by KO. The below equations define the variables involved in this function.

$$Let\ KO = (KO_{n-1}, \dots, KO_1, KO_0)$$

$$KO_i = \left(KO_{i(m-1)}, \dots, KO_{i1}, KO_{i0}\right)\ or\ KO_i \in \{0,1\}^m\ ;\ 0 \le i \le n-1$$

$$Z = (Z_{n-1}, \dots, Z_1, Z_0) = (F_{OUT}(z'_{n-1}, KO_{n-1}), \dots, F_{OUT}(z'_1, KO_1), F_{OUT}(z'_0, KO_0))$$

Equation (7) gives an algebraic expression for the sum of products form of a MUX function.

$$Z_i = F_{OUT}(z'_i, KO_i) = \sum_{p=0}^{w-1} z'_{ip} \prod_{q=0}^{m-1} (-1)^{\frac{p}{2^q}+1} KO_{pq}\ ;(-1)complements\ KO_{pq} \quad (7)$$

An example with m = 3 is provided below:

$$\mathrm{Z_i} = \sum_{p=0}^{w-1} z'_{ip} \prod_{q=0}^{m-1} (-1)^{\left\lfloor \frac{p}{2^q} \right\rfloor + 1} KO_{pq} \; ; m = 3$$

$$\mathrm{Z_i} = \sum_{p=0}^{8-1} z'_{ip} \prod_{q=0}^{3-1} (-1)^{\left\lfloor \frac{p}{2^q} \right\rfloor + 1} KO_{iq}$$

$$= z'_{i0}(-1)^1 KO_{i0}(-1)^1 KO_{i1}(-1)^1 KO_{i2} + z'_{i1}(-1)^2 KO_{i0}(-1)^1 KO_{i1}(-1)^1 KO_{i2}$$

$$+ z'_{i2}(-1)^3 KO_{i0}(-1)^2 KO_{i1}(-1)^1 KO_{i2} + z'_{i3}(-1)^4 KO_{i0}(-1)^2 KO_{i1}(-1)^1 KO_{i2}$$

$$+ z'_{i4}(-1)^5 KO_{i0}(-1)^3 KO_{i1}(-1)^2 KO_{i2} + z'_{i5}(-1)^6 KO_{i0}(-1)^3 KO_{i1}(-1)^2 KO_{i2}$$

$$+ \mathrm{z'_{i6}}(-1)^7 KO_{i0}(-1)^4 KO_{i1}(-1)^2 KO_{i2} + z'_{i7}(-1)^8 KO_{i0}(-1)^4 KO_{i1}(-1)^2 KO_{i2}$$

$$= \mathrm{z'_{i0}}\overline{\mathrm{KO_{i0}KO_{i1}KO_{i2}}} + \mathrm{z'_{i1}KO_{i0}}\overline{\mathrm{KO_{i1}KO_{i2}}} + \mathrm{z'_{i2}}\overline{\mathrm{KO_{i0}}}\mathrm{KO_{i1}}\overline{\mathrm{KO_{i2}}} + \mathrm{z'_{i3}KO_{i0}KO_{i1}}\overline{\mathrm{KO_{i2}}}$$

$$+ \mathrm{z'_{i4}}\overline{\mathrm{KO_{i0}KO_{i1}}}\mathrm{KO_{i2}} + \mathrm{z'_{i5}KO_{i0}}\overline{\mathrm{KO_{i1}}}\mathrm{KO_{i2}} + \mathrm{z'_{i6}KO_{i0}KO_{i1}}\overline{\mathrm{KO_{i2}}} + \mathrm{z'_{i7}KO_{i0}KO_{i1}KO_{i2}}$$

4 Analysis of the Proposed Design

In this chapter, the proposed design is analyzed with respect to Algebraic Attack.

4.1 Probability of Carries and Conditional Properties of Modulo Addition in the New Design

In (4), the probability of carries in a traditional Modulo Addition is defined. The probability of carries in the new design can be analyzed in a similar way. The Input Expansion function described in (5) is not only chosen to be scalable, but also 0-1 balanced. One can view the Input Expansion function as a collection of Boolean functions, or a vectorial Boolean function. Each expanded input variables is the result of a $\{0,1\}^{m+1} \rightarrow \{0,1\}^1$ Boolean function and is 0-1 balanced as the function outputs the same amount of 0 and 1. In other words, the probability of each Boolean function producing a 0 or a 1 is uniformly distributed. This coincides with the assumption made when calculating the probability of carries in (4). The probability of carries in the new design can then be defined as below.

$$C' = (c'_{n-1}, \dots, c'_1, c'_0)$$

$$= \left(c'_{(n-1)w}, \dots, c'_{(n-1)2}, c'_{(n-1)1}, \dots, c'_{1w}, \dots, c'_{12}, c'_{11}, c'_{0w}, \dots, c'_{02}, c'_{01}\right);\ w = 2^m$$

$$\begin{cases} \Pr(c'_{01} = 1) = \frac{1}{2} * \frac{1}{2} = \frac{1}{4}\ ;\ x'_{00} = y'_{00} = 1 \\ \Pr(c'_{01} = 0) = \frac{3}{4} \end{cases}$$

$$\begin{cases} \Pr(c'_{02} = 1) = \frac{1}{2} * \frac{1}{2} * \frac{1}{4} + \frac{1}{2} * \frac{1}{2} * \frac{3}{4} + \frac{1}{2} * \frac{1}{4} * \frac{1}{2} + \frac{1}{2} * \frac{1}{4} * \frac{1}{2} = \frac{3}{8}\ ; \begin{cases} x'_{01} = y'_{01} = c'_{01} = 1 \\ x'_{01} = y'_{01} = 1\ , c'_{01} = 0 \\ x_{01} = c'_{01} = 1\ , y'_{01} = 0 \\ y'_{01} = c'_{01} = 1\ , x'_{01} = 0 \end{cases} \\ \Pr(c'_{02} = 0) = 1 - \Pr(c'_{02} = 1) = \frac{5}{8} \end{cases}$$

$$\begin{cases} \Pr(c'_{ij} = 1) = {2^{i*w+j} - 1}/{2^{i*w+j+1}} \\ \Pr(c'_{ij} = 0) = 1 - \Pr\,(c'_{ij} = 1) \end{cases} \quad (8)$$

$$0 \le i \le n-1\,;\; 1 \le j \le 2^m\,;\; \mathrm{w} = 2^m$$

In both (4) and (8), the probability of carry approaches ½ as the number of bits in the modulo addition increases.

4.1.1 Condition 1: No Carries

The attacker may try to use the conditional properties of Modulo Addition to linearize this part of the equations. However, the attacker would discover that the Output Compaction function is lossy, i.e., the attacker cannot derive all of Z'_i from Z_i even if the output control bits are given. In other words, the MUX function is not reversible. Nonetheless, if somehow the attacker is able to obtain Z'_i, the conditional properties for Modulo Addition should still be analyzed. For the equations to have no carries at all, the probability has now roughly become $2^{-(n*w-1)}; w = 2^m$.

4.1.2 Condition 2: Output Characteristics

For the condition where every bit of the sum is a 1, it is required that the expanded input pair needs to be all Propagate. This implies that, the two m-bits input control string for each input pair, $KI_{Xi}\ and\ KI_{Yi}$, need to be the same. This is because the Input Expansion function will generate no carries only under this condition. The probability of this condition is $(2^{-n})(2^{-mn}) = 2^{-n(1+m)}$, or an increase of 2^{mn}. The derivation is shown below.

$$For\ m = 2\ and\ KI_{Xi} = KI_{Yi},$$

$$Let\ T = (KI_{Xi}, KI_{Yi}) = \left(KI_{Xi,1}, KI_{Xi,0}, KI_{Yi,1}, KI_{Yi,0}\right) = (T_3, T_2, T_1, T_0)$$

$$For\ (T_3, T_2) = (T_1, T_0), there\ are\ 4\ out\ of\ 16\ such\ cases, i.e.$$

$$(0,0)(0,0), (0,1)(0,1), (1,0)(1,0), (1,1)(1,1)$$

$$Similarly\ for\ m = 3, and\ KI_{Xi} = KI_{Yi},$$

$$Let\ T = (KI_{Xi}, KI_{Yi}) = \left(KI_{Xi,2}, KI_{Xi,1}, KI_{Xi,0}, KI_{Yi,2}, KI_{Yi,1}, KI_{Yi,0}\right)$$

$$= (T_5, T_4, T_3, T_2, T_1, T_0)$$

$$for\ (T_5, T_4, T_3) = (T_2, T_1, T_0), there\ are\ 8\ out\ of\ 64\ such\ cases$$

$$The\ probability\ of KI_{Xi} = KI_{Yi}\ is: {}^{2^m}/_{2^{2m}} = 2^{-m}$$

$$For\ n - bits\ addition, it\ is: 2^{-mn}$$

4.1.3 Condition 3: Input Characteristics

The conditions of one of the inputs being all 0 or all 1 in binary form do not happen in this Input Expansion function. However, it is possible for the expanded inputs to be Two's Complement of each other. The probability is derived below.

$$For\ m = 2\ , y'_i = Two's\ Complement\ (x'_i),$$

$$\left(x'_{i3}, x'_{i2}, x'_{i1}, x'_{i0}, y'_{i3}, y'_{i2}, y'_{i1}, y'_{i0}\right)$$

$$= (1,1,1,0,0,0,1,0), (0,0,1,0,1,1,1,0), (1,0,0,0,1,0,0,0)$$

$$Let\ T = (X_i, KI_{Xi}, Y_i, KI_{Yi}) = \left(X_i, KI_{Xi,1}, KI_{Xi,0}, Y_i, KI_{Yi,1}, KI_{Yi,0}\right)$$

$$= (0,0,0,1,0,1), (1,0,1,0,0,0), (1,1,1,1,1,1)$$

$$The\ Probability\ is\ {}^{3}/_{64}$$

$$Similarly\ for\ m = 3\ and\ y'_i = Two's\ Complement\ (x'_i),$$

$$\left(x'_{i7}, x'_{i6}, x'_{i5}, x'_{i4}, x'_{i3}, x'_{i2}, x'_{i1}, x'_{i0}, y'_{i7}, y'_{i6}, y'_{i5}, y'_{i4}, y'_{i3}, y'_{i2}, y'_{i1}, y'_{i0}\right)$$

$$= (1,1,1,1,1,1,1,0,0,0,0,0,0,0,1,0), (0,0,0,0,0,0,1,0,1,1,1,1,1,1,1,0), (1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0)$$

$$Let\ T = (X_i, KI_{Xi}, Y_i, KI_{Yi}) = \left(X_i, KI_{Xi,2}, KI_{Xi,1}, KI_{Xi,0}, Y_i, KI_{Yi,2}, KI_{Yi,1}, KI_{Yi,0}\right)$$

$$= (0,0,0,0,1,0,0,1), (1,0,0,1,0,0,0,0), (1,1,1,1,1,1,1,1)$$

$$The\ Probability\ is\ {}^{3}/_{256}$$

$$In\ general, the\ probabiliy\ of\ this\ condition\ is\ {}^{3}/_{2^{2m+2}}{}^{n} = 3^n 2^{-2n(m+2)} \ll 2^{-n}$$

4.2 Complexity of Solving the New Design

The proposed design, similar to Modulo Addition, can be viewed as an S-box, and its algebraic degree can be evaluated. This "S-Box" performs the transformation of $\{0,1\}^{2n+3nm} \rightarrow \{0,1\}^n$. It can also be seen as key dependent, as its values and operations are dependent on the input and output control bits. In order to evaluate the complexity of solving the new design, the algebraic degree of the components in the new design should be studied. For the Input Expansion function, each expanded input variable is a result of a Boolean function. The algebraic degree of a Boolean function can be determined from its Algebraic Normal Form (ANF). This form expresses a Boolean function using addition in GF(2), or XOR logic. An algorithm is provided in the figure below to transform the Truth Table of a Boolean function to its Algebraic Normal Form.

The ANF Algorithm

1. Consider a Boolean function f with n variables: $f(x_{n-1},\ldots,x_1,x_0)$
2. Let g be a Boolean function with the same n variables: $g(x_{n-1},\ldots,x_1,x_0)$
3. Set $g(x_{n-1},\ldots,x_1,x_0) = f(0,\ldots,0,0)$
4. Let k = 1 to 2^n-1, and its binary expression is defined as: $k = b_1 + b_2 2 + b_3 2^2 + \ldots + b_n 2^n$-1, do:
    a. If $g(b_n,\ldots,b_1,b_0) \neq f(b_n,\ldots,b_1,b_0)$
    b. Set $g(x_{n-1},\ldots,x_1,x_0) = g(x_{n-1},\ldots,x_1,x_0) \oplus \prod_{i=1}^{n} x_i{}^{b_i}$

Figure 6 - The ANF Algorithm

Table 4 provides an example of using the ANF algorithm to obtain the Algebraic Normal Form of the targeted Boolean function. The ANF of the Boolean function is: $1 \oplus KI_{x1}KI_{x0} \oplus x_i$.

x_i	KI_{x1}	KI_{x0}	$f(x_i, KI_{x1},$	k	$g(x_i, KI_{x1},$	Comments

			KI_{x0})		KI_{x0})	
0	0	0	1		1	g = f(0,0,0)
0	0	1	1	1	1	g(k) = f(k)
0	1	0	1	2	1	g(k) = f(k)
0	1	1	0	3	$1 \oplus KI_{x1}KI_{x0}$	g(k) ≠ f(k)
1	0	0	0	4	$1 \oplus KI_{x1}KI_{x0} \oplus x_i$	g(k) ≠ f(k)
1	0	1	0	5		g(k) = f(k)
1	1	0	0	6		g(k) = f(k)
1	1	1	1	7		g(k) = f(k)

Table 4 - Example of ANF Algorithm

4.2.1 Algebraic Degree of Input Expansion

The algebraic degree for each expanded input variables is the monomial with the largest degree in its Algebraic Normal Form. When the user defines the input control bit parameter m, the algebraic degree is defined to be m as well. This can be seen in the derivations shown in Table 5 and 6. On the other hand, it is quite intuitive that each expanded variable depends on the combination of all m bits in the input control string.

X_i	ANF	Algebraic Degree
x'_{i0}	$KI_{x1} \oplus KI_{x0} \oplus KI_{x1}KI_{x0} \oplus x_i$	2
x'_{i1}	$1 \oplus KI_{x0} \oplus KI_{x1}KI_{x0} \oplus x_i$	2
x'_{i2}	$1 \oplus KI_{x1} \oplus KI_{x1}KI_{x0} \oplus x_i$	2
x'_{i3}	$1 \oplus KI_{x1}KI_{x0} \oplus x_i$	2

Table 5 - ANF of Input Expansion m = 2

X_i	ANF	Algebraic Degree
x'_{i0}	$KI_{x0} \oplus KI_{x1} \oplus KI_{x1}KI_{x0} \oplus KI_{x2}$	3

	$\oplus KI_{x2}KI_{x0} \oplus KI_{x2}KI_{x1} \oplus KI_{x2}KI_{x1}KI_{x0} \oplus x_i$	
x'_{i1}	$1 \oplus KI_{x0} \oplus KI_{x1}KI_{x0} \oplus KI_{x2}KI_{x0} \oplus KI_{x2}KI_{x1}KI_{x0} \oplus x_i$	3
x'_{i2}	$1 \oplus KI_{x1} \oplus KI_{x1}KI_{x0} \oplus KI_{x2}KI_{x1} \oplus KI_{x2}KI_{x1}KI_{x0} \oplus x_i$	3
x'_{i3}	$1 \oplus KI_{x1}KI_{x0} \oplus KI_{x2}KI_{x1}KI_{x0} \oplus x_i$	3
x'_{i4}	$1 \oplus KI_{x2} \oplus KI_{x2}KI_{x0} \oplus KI_{x2}KI_{x1} \oplus KI_{x2}KI_{x1}KI_{x0} \oplus x_i$	3
x'_{i5}	$1 \oplus KI_{x2}KI_{x0} \oplus KI_{x2}KI_{x1}KI_{x0} \oplus x_i$	3
x'_{i6}	$1 \oplus KI_{x2}KI_{x1} \oplus KI_{x2}KI_{x1}KI_{x0} \oplus x_i$	3
x'_{i7}	$1 \oplus KI_{x2}KI_{x1}KI_{x0} \oplus x_i$	3

Table 6 - ANF of Input Expansion m = 3

Table 7 tabulates a quick comparison of the input algebraic degrees.

	Input to traditional Modulo Addition	**Input to Modulo Addition in the new design**
Algebraic Degree	1	m

Table 7 - Comparison of Input Algebraic Degrees

4.2.2 Algebraic Degree of Modulo Addition

After the algebraic degree is obtained for the Input Expansion component, the degree of the Modulo Addition component is required. Using the quadratic equations in (3), the algebraic degree is at least m. This can be realized from the "linear" equation for the least significant bit addition, which is a linear combination, or XOR, of two degree m equations. At this point, it can already be seen that the combination of Input Expansion

and Modulo Addition increases the Algebraic Immunity to m, compared to 1 before. Nonetheless, the describing degree is more of the concern as discussed in the earlier section. Therefore, the maximum algebraic degree in the set of equations needs to be uncovered. Equation (3) limits the describing degree to two times the degree of the linear equation because its largest degree comes from the multiplication of two degree-1 variables. In other words, the quadratic equations have a degree of $1 + 1 = 2$. In this case, the "linear" equation has a degree of m; therefore, the quadratic equations have a degree of $m + m = 2m$. The describing degree has effectively been increased. An example is listed below.

$$z'_{00} = x'_{00} + y'_{00}$$
$$z'_{01} = x'_{01} + y'_{01} + x'_{00}y'_{00}$$
$$z'_{02} = x'_{02} + y'_{02} + x'_{01}y'_{01} + (x'_{01} + y'_{01})(x'_{01} + y'_{01} + z'_{01})$$
$$\vdots$$
$$z'_{0i} = x'_{0i} + y'_{0i} + x'_{0i-1}y'_{0i-1} + (x'_{0i-1} + y'_{0i-1})(x'_{0i-1} + y'_{0i-1} + z'_{0i-1})$$

In terms of degree:

$$Z_0 = m + m \rightarrow m$$
$$Z_1 = m + m + m * m \rightarrow 2m$$
$$Z_2 = m + m + 2m + (m + m)(m + m + 1) \rightarrow 2m$$
$$\vdots$$
$$Z_i = m + m + 2m + (m + m)(m + m + 1) \rightarrow 2m$$

It is important to note that, the use of quadratic equations is valid only when the attacker is able to access the internal sums, or Z'. The attacker can also define Z' as extra variables in the S-Box so that the equations can be used; however, this method may not be beneficial, as the number of variables will increase.

The Modulo Addition component can also be described by equations (1) and (2); in particular, the algebraic degree is dominated by (2). With careful inspection, it can be observed that the algebraic degrees of equations defining the traditional Modulo Addition are 1 for Z_0 and $i + 1$ for Z_i. This makes sense because the more significant the

carry bit is, the more dependent it is on the previous input bits. The derivation is shown below.

$$Z_0 = X_0 + Y_0 \rightarrow \text{Degree } 1$$

$$Z_1 = X_1 + Y_1 + C_0 = X_1 + Y_1 + X_0Y_0 \rightarrow Degree\ 2$$

$$Z_2 = X_2 + Y_2 + C_1 = X_2 + Y_2 + X_1Y_1 + (X_1 + Y_1)(X_0Y_0) \rightarrow Degree\ 3$$

$$Z_i = X_i + Y_i + C_{i-1} \rightarrow \text{Degree i} + 1$$

Similarly, the algebraic degrees of equations defining the Modulo Addition component in the new design are m for z'_{00} and $(i * w + j + 1)m$ for z'_{ij} ; $0 \leq i \leq n-1$; $0 \leq j \leq 2^m - 1; w = 2^m$. An Example is also shown below.

$$z'_{00} = x'_{00} + y'_{00} \rightarrow \text{Degree m}$$

$$z'_{01} = x'_{01} + y'_{01} + c'_{00} = x'_{01} + y'_{01} + x'_{00}y'_{00} \rightarrow \text{Degree 2m}$$

$$z'_{02} = x'_{02} + y'_{02} + c'_{01} = x'_{02} + y'_{02} + x'_{01}y'_{01} + \left(x'_{01} + y'_{01}\right)\left(x'_{00}y'_{00}\right)$$

$$\rightarrow \text{Degree 3m}$$

$$z'_{10} = x'_{10} + y'_{10} + c'_{0(w-1)} =$$

$$x'_{10} + y'_{10} + x'_{0(w-1)}y'_{0(w-1)} + \left(x'_{0(w-1)} + y'_{0(w-1)}\right)\left(x'_{0(w-2)}y'_{0(w-2)}\right)$$

$$+ \sum_{p=0}^{w-3} X_{0p}Y_{0p} \prod_{q=p+1}^{w-1} \left(X_{0q} + Y_{0q}\right) \rightarrow \text{Degree } (1 * \text{w} + 1)\text{m}$$

A comparison of degrees using different methods between the traditional Modulo Addition and the new design is given in Table 8.

	Sum of traditional Modulo Addition	**Sum of Modulo Addition in the new design**
Algebraic Degree using (3)	$1 \rightarrow 2$	$\text{m} \rightarrow 2\text{m}$

Algebraic Degree using (1) and (2)	$1 \rightarrow \mathrm{i}+1$ for $\mathrm{Z_i}$ $0 \leq i \leq n-1$	$\mathrm{m} \rightarrow (\mathrm{i} * \mathrm{w} + \mathrm{j} + 1)\mathrm{m}$ for $\mathrm{z'_{ij}}$ $0 \leq i \leq n-1$ $0 \leq j \leq 2^m - 1$ $\mathrm{w} = 2^{\mathrm{m}}$

Table 8 - Comparison of Modulo Addition Degrees

4.2.3 Algebraic Degree of Output Compaction

Finally, the algebraic degree of the Output Compaction component needs to be determined. Again, the Output Compaction function is a user defined 2^m:1 MUX function and the ANF of a MUX function can be obtained from the ANF algorithm. The algebraic degree of the function is $\mathrm{m}+1$ when the inputs to the MUX all have degree 1. This is intuitive as the MUX output is generated from using all select lines to select the desired input. In fact, equation (7) can be used as the ANF of the MUX function and the algebraic degree can be immediately discovered to be $\mathrm{m}+1$. In addition, the algebraic degree can still be derived by using the ANF algorithm described above and the result is the same. Examples of ANF of different MUX functions are provided in the table below.

	ANF	**Algebraic Degree**
M = 2	$Z_i = z'_{i0} \oplus z'_{i0}KO_{i0} \oplus z'_{i0}KO_{i1} \oplus z'_{i0}KO_{i1}KO_{i0} \oplus z'_{i1}KO_{i0} \oplus z'_{i1}KO_{i1}KO_{i0} \oplus z'_{i2}KO_{i1} \oplus z'_{i2}KO_{i1}KO_{i0} \oplus z'_{i3}KO_{i1}KO_{i0}$	3
M = 3	$Z_i = z'_{i0} \oplus z'_{i0}KO_{i0} \oplus z'_{i0}KO_{i1} \oplus z'_{i0}KO_{i1}KO_{i0} \oplus z'_{i0}KO_{i2} \oplus z'_{i0}KO_{i2}KO_{i0} \oplus z'_{i0}KO_{i2}KO_{i1} \oplus z'_{i0}KO_{i2}KO_{i1}KO_{i0} \oplus z'_{i1}KO_{i0} \oplus z'_{i1}KO_{i1}KO_{i0} \oplus z'_{i1}KO_{i2}KO_{i0} \oplus$	4

	$z'_{i1}KO_{i2}KO_{i1}KO_{i0} \oplus z'_{i2}KO_{i1} \oplus z'_{i2}KO_{i1}KO_{i0} \oplus z'_{i2}KO_{i2}KO_{i1} \oplus z'_{i2}KO_{i2}KO_{i1}KO_{i0} \oplus z'_{i3}KO_{i1}KO_{i0} \oplus z'_{i3}KO_{i2}KO_{i1}KO_{i0} \oplus z'_{i4}KO_{i2} \oplus z'_{i4}KO_{i2}KO_{i0} \oplus z'_{i4}KO_{i2}KO_{i1} \oplus z'_{i4}KO_{i2}KO_{i1}KO_{i0} \oplus z'_{i5}KO_{i2}KO_{i0} \oplus z'_{i5}KO_{i2}KO_{i1}KO_{i0} \oplus z'_{i6}KO_{i2}KO_{i1} \oplus z'_{i6}KO_{i2}KO_{i1}KO_{i0} \oplus z'_{i7}KO_{i2}KO_{i1}KO_{i0}$	

Table 9 - Example ANF of MUX functions

4.2.4 New Design Complexity

In Table 10, the final algebraic degree of the new design is combined and compared with the traditional Modulo Addition. Using Definition 1 in Chapter 2, the Algebraic Immunity has increased from 1 to 2m, displaying an increase of 2m. Similarly, the Describing Degree has increased $\mathrm{i}+1$ to $m(1+2^m(i+1))$, showing an increase of roughly 2^m. It can also be seen that the degree is scalable according to user definition.

	Traditional Modulo Addition	**New Modulo Addition**
Algebraic Degree of Input	1	m
Algebraic Degree of Addition using (1) and (2)	$1 \rightarrow \mathrm{i}+1$ for Z_i $0 \leq i \leq n-1$	$\mathrm{m} \rightarrow (\mathrm{i} * \mathrm{w} + \mathrm{j} + 1)\mathrm{m}$ for z'_{ij} $0 \leq i \leq n-1$

		$0 \leq j \leq 2^m - 1$ $\mathrm{w} = 2^{\mathrm{m}}$
Algebraic Degree of Output	-	$\mathrm{m} + 1$
Total	$1 \rightarrow \mathrm{i} + 1$ $0 \leq i \leq n - 1$	$2\mathrm{m} \rightarrow \mathrm{m} + (\mathrm{i} * 2^{\mathrm{m}} +$ $2^{\mathrm{m}} - 1 + 1)\mathrm{m} =$ $\mathrm{m}(1 + 2^{\mathrm{m}}(\mathrm{i} + 1))$ $0 \leq i \leq n - 1$

Table 10 - Comparison of Algebraic Degree

In Table 11, the complexity of the traditional Modulo Addition and the new design is compared. Equation (3) is used to calculate the complexity of the traditional Modulo Addition, and the calculation has been done in earlier chapter as well. The complexity of the new design using equations (1) and (2) becomes very high. An attacker would definitely try to uncover other possible methods to reduce the algebraic degree of the new design. Such case is explored in the next section.

	Traditional Modulo Addition Using (3)	**New Modulo Addition Using (1) and (2)**
Number of Input Variables	$2\mathrm{n}$	$3\mathrm{mn} + 2\mathrm{n}$
Number of Output Variables	n	n
Number of Extra Variables	0	0
Number of Equations	$\mathrm{R} = 6\mathrm{n} - 3$	$\mathrm{R} = \mathrm{n}$
Number of Monomials	$\mathrm{T} = \sum_{\mathrm{i}=0}^{2} \binom{3 * \mathrm{n}}{\mathrm{i}}$	$\mathrm{T} = \sum_{\mathrm{i}=0}^{\mathrm{m}(1+2^{\mathrm{m}}(\mathrm{i}+1))} \binom{3\mathrm{n}(1+\mathrm{m})}{\mathrm{i}}$
Complexity	$\Gamma = {}^{T}/_{2n}^{\lceil {}^{T}/_{R} \rceil}$	$\Gamma = {}^{T}/_{(3mn+2n)}^{\lceil {}^{T}/_{R} \rceil}$

Table 11 - Comparison of Complexity

4.2.5 Corner Case Analysis

Instead of solving directly using the equations and algebraic degrees derived before, an attacker can rely on probabilistic properties to solve the new design. In other words, conditional properties of the new design can be used. For the Input Expansion, it can be observed that if the m-bit control strings are known, the expanded inputs largely depend on the input or the complement of the input. Furthermore, if all m-bit control strings are 0, the expanded inputs are going to equal to the complement of the input and the least significant bit of the expanded input is simply the input. This condition greatly lowers the degree of the subsequent Modulo Addition component. In fact, the highest degree of adding the first 2^m bits of expanded input is 2 while the lowest degree of this addition resides in the least significant bit and is only 1. However, the degree of adding the subsequent 2^m bits of expanded input becomes higher and can be generalized to $i + 2$ for the $2^m - 1$ bits in the expanded input bits of the i^{th} input bit. The degree of adding the least significant bit in the expanded input bits of the i^{th} input bit is $i + 1$. The below table demonstrates the derivation when m is defined to be 2.

X_i, Y_i, Z_i	ANF (KI = 0)	Algebraic Degree
x'_{00}	x_0	1
x'_{01}	$1 \oplus x_0$	1
x'_{02}	$1 \oplus x_0$	1
x'_{03}	$1 \oplus x_0$	1
y'_{00}	y_0	1
y'_{01}	$1 \oplus y_0$	1
y'_{02}	$1 \oplus y_0$	1
y'_{03}	$1 \oplus y_0$	1
z'_{00}	$x'_{00} \oplus y'_{00} = x_0 \oplus y_0$	*1*
z'_{01}	$1 \oplus x_0 \oplus 1 \oplus y_0 \oplus x_0 y_0 = x_0 \oplus y_0 \oplus x_0 y_0$	2

z'_{02}	$1 \oplus x_0 \oplus 1 \oplus y_0 \oplus (1 \oplus x_0)(1 \oplus y_0) \oplus (1 \oplus x_0 \oplus 1 \oplus y_0)(1 \oplus x_0 \oplus 1 \oplus y_0)(x_0 \oplus y_0) = 1 \oplus x_0 \oplus y_0 \oplus x_0y_0$	2
z'_{03}	$1 \oplus x_0 \oplus 1 \oplus y_0 \oplus (1 \oplus x_0)(1 \oplus y_0) \oplus (1 \oplus x_0 \oplus 1 \oplus y_0)(1 \oplus x_0 \oplus 1 \oplus y_0)(1 \oplus x_0)(1 \oplus y_0) \oplus (1 \oplus x_0 \oplus 1 \oplus y_0)(1 \oplus x_0 \oplus 1 \oplus y_0)(1 \oplus x_0 \oplus 1 \oplus y_0)(x_0 \oplus y_0) = 1 \oplus x_0 \oplus y_0 \oplus x_0y_0$	2
x'_{10}	x_1	1
x'_{11}	$1 \oplus x_1$	1
y'_{10}	y_1	1
y'_{11}	$1 \oplus y_1$	1
z'_{10}	$x_1 \oplus y_1 \oplus (1 \oplus x_0)(1 \oplus y_0) \oplus (1 \oplus x_0 \oplus 1 \oplus y_0)(1 \oplus x_0 \oplus 1 \oplus y_0 \oplus z'_{03}) = x_1 \oplus y_1 \oplus x_0y_0$	*2*
z'_{11}	$1 \oplus x_1 \oplus 1 \oplus y_1 \oplus x_1y_1 \oplus (x_1 \oplus y_1)(x_1 \oplus y_1 \oplus z'_{10}) = x_1 \oplus y_1 \oplus x_1y_1 \oplus x_1x_0y_0 \oplus y_1x_0y_0$	3

Table 12 - Input Expansion and Resultant ANF for m = 2 and KI = 0

The scalability of the algebraic degree has been reduced; however, the degree of the sum is still equal or greater than $i + 1$ or 2, which are the degrees of the sum in the traditional Modulo Addition calculated using different equations.

The attacker would also observe that the degree can be greatly lowered in the Output Compaction function if the m-bit control strings are all 0's. The degree is simply 1 in this case. This can be easily derived from equation (7) and table 9. As a result, if all input and output control bits are 0's, the degree of each output variable is strictly dependent on the combined degree of Input Expansion and Modulo Addition. Under this specific condition, the overall system has a degree of $i + 1$.

This corner case scenario can be avoided as the goal is to avoid having all input and output control bits to be 0 simultaneously. One can use one string of control bits for one input and the complement of the same string for the other input. There are also many other ways to generate the control strings within a cipher, such as in [21], [22], and [32]. Furthermore, the cost associated to this condition is 2^{3mn} for all control bits to be 0, because the probability of such corner case occurring is 2^{-3mn}. Table 13 provides a comparison of complexity between the traditional Modulo Addition and the corner case of the new design.

	Traditional Modulo Addition Using (3)	**New Modulo Addition Corner Case**
Number of Input Variables	$2n$	$3mn + 2n$
Number of Output Variables	n	n
Number of Extra Variables	0	0
Number of Equations	$R = 6n - 3$	$R = n$
Algebraic Degree	2	n
Condition Cost	0	2^{3mn}
Number of Monomials	$T = \sum_{i=0}^{2} \binom{3 * n}{i}$	$T = \sum_{i=0}^{n} \binom{3n(1+m)}{i}$
Complexity	$\Gamma = {}^{T}/_{2n}{}^{\lceil T/R \rceil}$	$\Gamma = 2^{3mn} * {}^{T}/_{(3mn+2n)}{}^{\lceil T/R \rceil}$

Table 13 - Comparison of Corner Case Complexity

Overall, the new design has provided a better and scalable security against Algebraic Attack by improving the algebraic degree to at least n, increasing the number of monomials, and adding a conditional cost of 2^{3mn}.

5 Applications of the New Design

In this chapter, a toy example is given to demonstrate the effect of the new design. Then, the proposed design is applied to two stream ciphers. The overviews of the two stream ciphers are given, and the application is discussed. The results are explained in the next chapter.

5.1 Toy Example

In the toy example, a simple 3-bit cipher is employed. The unencrypted input P is fed into a traditional Addition Modulo 23, along with the secret key K, to generate the encrypted output C. This cipher is refereed as Cipher A. Then, the traditional Modulo Addition is replaced by the new design and this is referred as Cipher B. The input and output control strings are termed KI_p, KI_k, and KO and are assumed to be supplied by some other sources that have uniform probability. Also, m is defined to be 2 for simplicity. The two ciphers are illustrated in the below figure.

Figure 7 - Toy Example Block Diagram

A quick analysis can be provided to show the difference in algebraic degree between the two ciphers. Note that + is used to denote XOR operation. For Cipher A:

$$C_0 = P_0 + K_0$$

$$C_1 = P_1 + K_1 + P_0K_0$$

$$C_2 = P_2 + K_2 + P_1K_1 + (P_1 + K_1)(P_0K_0)$$

The Algebraic Immunity of this set of equations is 1 whereas the Describing Degree is 3. The attacker would then attempt to solve this set of equations to obtain the secret key K. At the same time, the conditional properties mentioned in Chapter 2 can be used. Both the probabilities of output characteristic and input characteristic are 2^{-3}. Similarly, for Cipher B:

$$C_0 = \Big(\big(KI_{p01} + KI_{p00} + KI_{p01}KI_{p00} + P_0\big) + (KI_{k01} + KI_{k00} + KI_{k01}KI_{k00} + K_0)\Big)\overline{KO_{01}KO_{00}}$$

$$+\Big(\big(1 + KI_{p00} + KI_{p01}KI_{p00} + P_0\big) + (1 + KI_{k00} + KI_{k01}KI_{k00} + K_0) + \big(KI_{p01} + KI_{p00} + KI_{p01}KI_{p00} + P_0\big)(KI_{k01} + KI_{k00} + KI_{k01}KI_{k00} + K_0)\Big)\overline{KO_{01}}KO_{00}$$

$$+\Big(\big(1 + KI_{p01} + KI_{p01}KI_{p00} + P_0\big) + (1 + KI_{k01} + KI_{k01}KI_{k00} + K_0) + \big(1 + KI_{p00} + KI_{p01}KI_{p00} + P_0\big)(1 + KI_{k00} + KI_{k01}KI_{k00} + K_0) + \big(1 + KI_{p00} + KI_{p01}KI_{p00} + P_0 + 1 + KI_{k00} + KI_{k01}KI_{k00} + K_0\big)\big(KI_{p01} + KI_{p00} + KI_{p01}KI_{p00} + P_0\big)(KI_{k01} + KI_{k00} + KI_{k01}KI_{k00} + K_0)\Big)KO_{01}\overline{KO_{00}}$$

$$+\big(\big(1 + KI_{p01}KI_{p00} + P_0\big) + (1 + KI_{k01}KI_{k00} + K_0) + \big(1 + KI_{p01} + KI_{p01}KI_{p00} + P_0\big)(1 + KI_{k01} + KI_{k01}KI_{k00} + K_0) + \big(1 + KI_{p01} + KI_{p01}KI_{p00} + P_0 + 1 + KI_{k01} + KI_{k01}KI_{k00} + K_0\big)\big(1 + KI_{p00} + KI_{p01}KI_{p00} + P_0\big)(1 + KI_{k00} + KI_{k01}KI_{k00} + K_0) + \big(1 + KI_{p01} + KI_{p01}KI_{p00} + P_0 + 1 + KI_{k01} + KI_{k01}KI_{k00} + K_0\big)(1 + KI_{p00} + KI_{p01}KI_{p00} + P_0 + 1 + KI_{k00} + KI_{k01}KI_{k00} + K_0)\big(KI_{p01} + KI_{p00} + KI_{p01}KI_{p00} + P_0\big)(KI_{k01} + KI_{k00} + KI_{k01}KI_{k00} + K_0)\big)KO_{01}KO_{00}$$

$$\vdots$$

Only the equations describing the least significant bit of the output of Cipher B is shown here. However, the increase in algebraic degree can already be seen. The minimum degree has increased to 4, or 2m, in this set of equations. The maximum degree in this set is 10. This corresponds to the summary in Table 10. At the same time,

the probability of the conditional properties can be calculated using the equations described in Chapter 4. Specifically, the output characteristic has a probability of $2^{-3}2^{-3*2} = 2^{-9}$ whereas the probability of the corner case is 2^{-18}. As a result, the complexity of Algebraic Attack against Cipher A has been increased by employing the new design.

5.2 Application on Stream Ciphers with Linear Feedback

The targeted stream cipher belongs to a classical type of stream cipher that typically employs shift registers with nonlinear feedback and a linear output combiner function. This type of stream cipher is used very often as a pseudo-random number generator and encryption algorithm in resource constrained environment. The targeted stream cipher is Bivium, which is a subset of Trivium. Trivium is one of the finalists in the ECRYPT stream cipher project for hardware specific stream ciphers [33]. The specification of Bivium and the application of the new design will be described.

5.2.1 Overview of Bivium

In [34], Trivium is introduced as a hardware efficient stream cipher. It consists of three shift registers of lengths 93, 84, and 111. It operates in the initialization mode, which utilizes an 80-bit secret key and an 80-bit initialization vector, and the key stream generation mode, which is capable of generating up to 2^{64} output bits. In addition, Trivium can be truncated into a smaller stream cipher, Bivium. Bivium has two variants, Bivium A and Bivium B, and both variants use two shift registers of lengths 93 and 84. The operational modes, the length of secret keys, and the length of initialization vector are the same as Trivium. The stream cipher of interest in Bivium B and its operations are described and illustrated in the figures below. It is worth noting that the secret key and the initialization vector are loaded in the LFSRs for initialization. Also, as mentioned in Chapter 2, the unencrypted message input will be mixed with the key stream generated from the stream cipher to form the encrypted output. This general procedure is true for other stream cipher applications in this chapter as well.

Bivium B – Initialization Mode

1st LFSR → (S1, S2, …, S92, S93)

2nd LFSR → (S94, S95, …, S176, S177)

Secret Keys → (K1, K2, …, K79, K80)

IV → (IV1, IV2, …, IV79, IV80)

(S1, S2, …, S92, S93) = (K1, K2, …, K79, K80, 0, 0, …,0)

(S94, S95, …, S176, S177) = (IV1, IV2, …, IV79, IV80, 0, 0, …,0)

For i = 1 to 4*177, do:

t1 = S66 ⊕ S93 ⊕ S91S92 ⊕ S171

t2 = S162 ⊕ S177 ⊕ S175S176 ⊕ S69

(S1, S2, …, S92, S93) = (t2, S1, …, S92)

(S94, S95, …, S176, S177) = (t1, S94, …, S176)

End for

Bivium B – Keystream Generation Mode

For i = 1 to N, do:

t1 = S66 ⊕ S93

t2 = S162 ⊕ S177

zi = t1 ⊕ t2

t1 = t1 ⊕ S91S92 ⊕ S171

t2 = t2 ⊕ S175S176 ⊕ S69

(S1, S2, …, S92, S93) = (t2, S1, …, S92)

(S94, S95, …, S176, S177) = (t1, S94, …, S176)

End for

Figure 8 - Operation of Bivium B

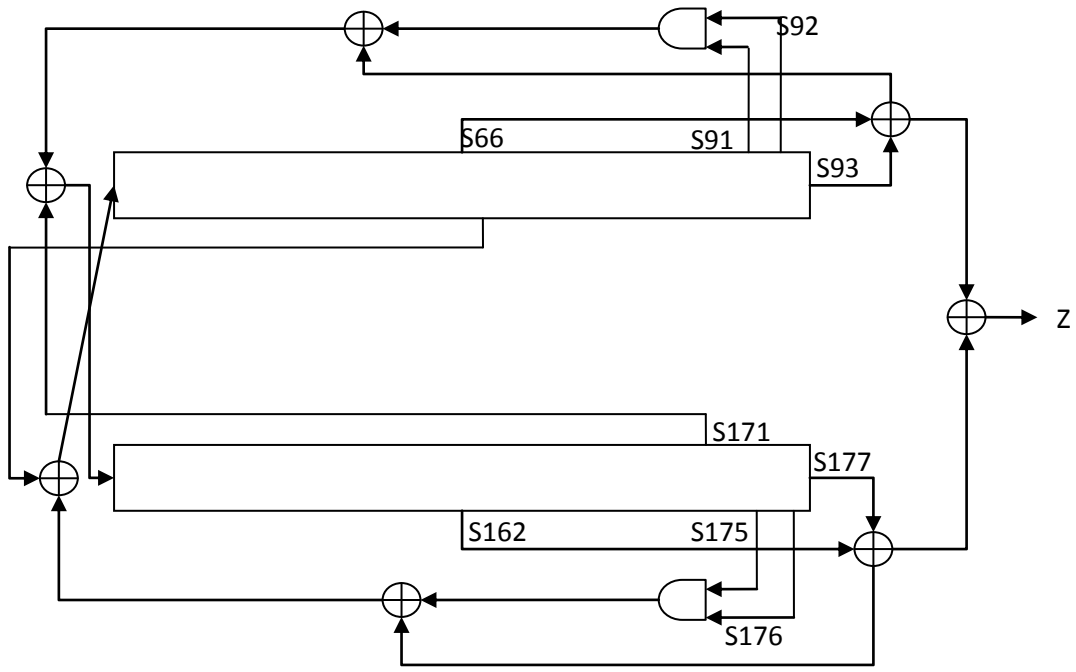

Figure 9 - Block Diagram of Bivium B

5.2.2 Application of the New Design in Bivium B

The new design will replace the linear output combiner function and the user defined parameter m is selected to be 4. The number of control bits is $3mn = 3 * 4 * 1 = 12$. As mentioned before, there are many ways to select the control bits; however, three conditions are used in the selection process. First, the control bits will be coming from the states in the shift registers so that no extra work is required to generate the control bits. Second, the states will be chosen using the Full Positive Difference Set (FPDS). The FPDS means that the difference between any two stages in the set is distinct. In other words, the difference between any two numbers in a set of numbers should be distinct. The FPDS is considered as it provides better resistance to correlation attack [35][36]. In addition, it is shown in [37] that states between S91 and S176 appear more often in the algebraic formulation between the states and the output bits. Therefore, the third condition for choosing the control bits is to limit the states to be less than S91. Finally, the control bits will be supplied by the three sets: (S16, S50, S70, S84), (S3, S8, S14, S30),

and (S11, S23, S41, S54). The operation of the new Bivium B is described in Figure 10. Figure 11 illustrates the block diagram of Bivium B with the new design.

New Bivium B – Key stream Generation Mode

For i = 1 to N, do:

t1 = S66 $\oplus$ S93

t2 = S162 $\oplus$ S177

KI_x = S16 | S50<<1 | S70<<2 | S84<<3

KI_y = S3 | S8<<1 | S14<<2 | S30<<3

K_o = S11 | S23<<1 | S41<<2 | S54<<3

T1_ext = Input Expansion (t1, KI_x)

T2_ext = Input Expansion (t2, KI_y)

Sum = Modulo Addition (T1_ext, T2_ext)

zi =Output Compaction (Sum, K_o)

t1 = t1 $\oplus$ S91S92 $\oplus$ S171

t2 = t2 $\oplus$ S175S176 $\oplus$ S69

(S1, S2, …, S92, S93) = (t2, S1, …, S92)

(S94, S95, …, S176, S177) = (t1, S94, …, S176)

End for

Figure 10 - Operation of New Bivium B Application

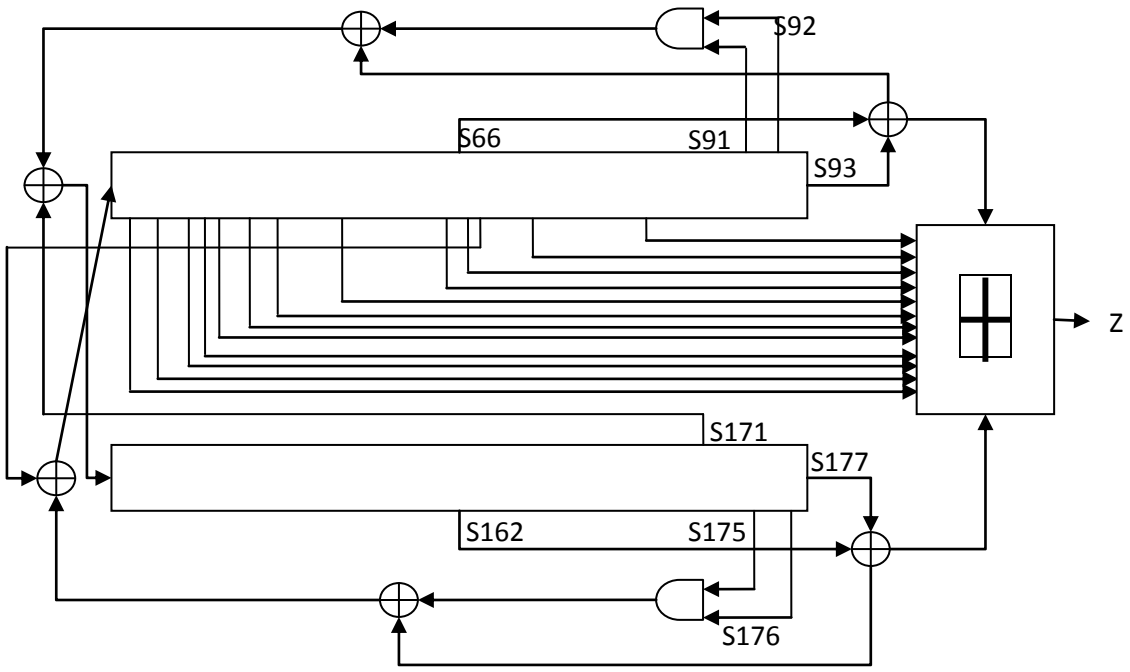

Figure 11 - Application of New Design in Bivium B

In table 14, first 10 bits of the resultant key stream of the application is shown. The intermediate steps of the new design are also tabulated. The secret key and initialization vector are chosen at random and are used in both old Bivium B and the new application. They are provided as:

$$K = 0xFDB7F6C605587A225269$$

$$IV = 0xE800600DBD553DD5FCF0$$

The 10 bits of key stream from the old Bivium B and the application is provided below:

$$z = 1111001110$$

$$z' = 0101101111$$

x_i	KI_{xi}	x'_i	y_i	KI_{yi}	y'_i	z'_i	KO_i	z'
0	1100	1110111111111111	1	0000	0000000000000001	1111000000000000	1010	0
0	1000	1111111011111111	1	1110	0100000000000000	0011111011111111	1011	1

1	0011	0000000000001000	0	1111	0111111111111111	1000000000000111	0011	0
0	1101	1101111111111111	1	0001	0000000000000010	1110000000000001	1110	1
0	0100	1111111111110111	0	1001	1111111011111111	1111110111101110	0011	1
1	1000	0000000100000000	1	0100	0000000000010000	0000000100010000	0101	0
0	0111	1111111101111111	1	0100	0000000000010000	1111111110001111	1110	1
1	1001	0000000100000000	0	1010	1111110111111111	1111111011111111	0100	1
1	1100	0001000000000000	0	1111	0111111111111111	1000111111111111	0000	1
0	1001	1111110111111111	0	1111	0111111111111111	0111110111111110	0110	1

Table 14 - Output from New Bivium B Application

The analysis of this application will be explained in detail in the next chapter.

5.3 Application on Stream Ciphers Using Combiners with Memory

In this section, the new design is applied to a type of stream cipher that uses shift registers with nonlinear feedback and an output combining function that contains memory registers. This type of stream cipher is also used widely as pseudo random number generator or encryption algorithm. In particular, the targeted stream cipher would be SNOW2.0. An overview of SNOW2.0 is provided and the application of the new design is described.

5.3.1 Overview of SNOW2.0

In [25], a linear feedback shift register (LFSR) based stream cipher named SNOW2.0 is proposed. It consists of a length 16 LFSR over GF(2^{32}); in other words, there are 16 elements in the LFSR, and each element consists of a 32-bit word. This type of stream cipher is based on the idea of a general word-oriented summation generator [8]. The feedback polynomial is defined by $\pi(x) = \alpha x^{16} + x^{14} + \alpha^{-1}x^5 + 1 \, \epsilon \, F_{2^{32}}[x]$. In simpler terms, the feedback polynomial is the XOR combination of the 0^{th} element (S0) multiplied by α, the 2^{nd} element (S2), and the 11^{th} element (S11) divided by α. A Finite State Machine (FSM) is also used in conjunction to produce the key stream. The FSM

consists of two 32-bits registers R1 and R2. The value of R2 is determined by feeding the value of R1 through a set of AES S-Boxes and the AES Mix Column operation. The value of R1 comes from performing Addition Modulo 2^{32} between R2 and S5. Finally, the key stream output comes from adding S15 and R1, XOR the result with R2, and XOR again with S0. The operation of the stream cipher begins with a key initialization step that initializes the 16 elements and the 2 registers with the aid of the 128-bit secret key and 128-bit initialization vector. Then, the cipher is clocked to produce a 32-bit key stream. Each clock will update the FSM and then the LFSR. Figure 12 shows the block diagram of the setup.

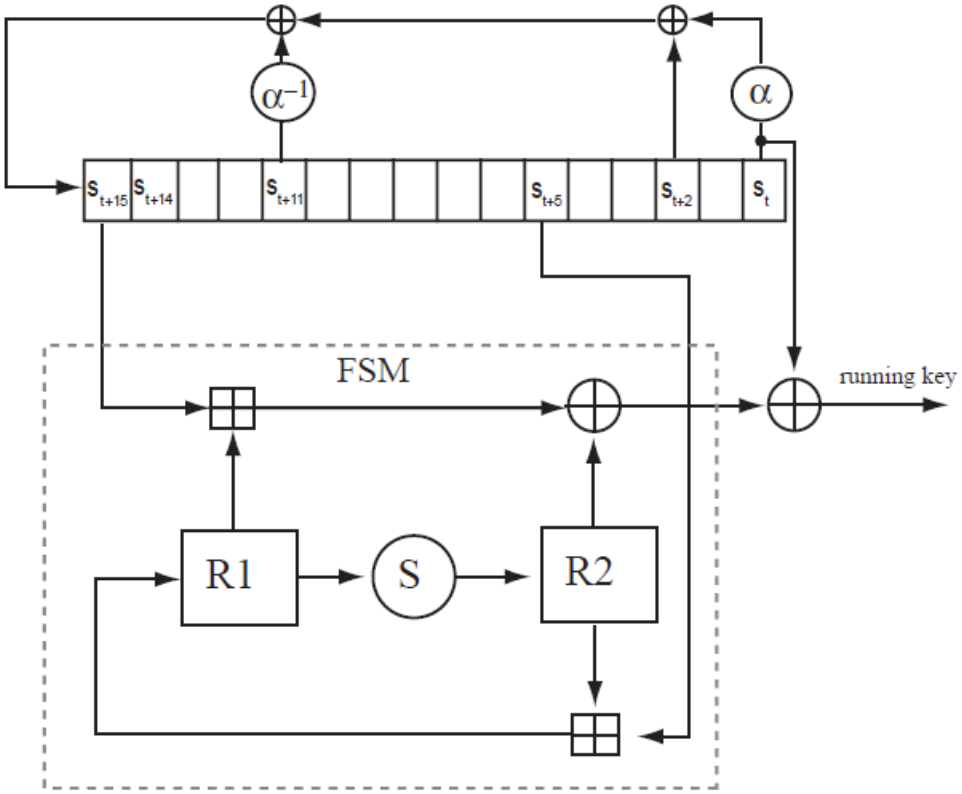

Figure 12 - Block Diagram of SNOW2.0 [25]

5.3.2 Application of the New Design in SNOW2.0

The new design is used to replace the two traditional Modulo Additions in SNOW2.0. In this application, m = 3 is used. This means that for each addition, 288 control bits are used, and a total of 576 bits are required. The supply of the control bits come from

utilizing the state S14 the following way: For each bit of the first input X, the 3 input control bits will be the least significant three bits of the 3-bit circular left shifted S14. Therefore, $KI_{X0} = (S14_2, S14_1, S14_0)$ and $KI_{X1} = (S14_{31}, S14_{30}, S14_{29})$. In contrast, for each bit of the second input Y, the 3 input control bits will be the least significant three bits of the 3-bit circular right shifted and inverted S14. Let S14' denote the bit-wise inverted S14, then $KI_{Y0} = (S14'_2, S14'_1, S14'_0)$ and $KI_{Y1} = (S14'_5, S14'_4, S14'_3)$. The output control bits is provided by the least significant three bits of the 3-bit circular right shifted S14, namely, $KO_0 = (S14_2, S14_1, S14_0)$ and $KO_1 = (S14_5, S14_4, S14_3)$. This control bit setup can at least guarantee that the input control bits for the first bit of both inputs will not be 0 simultaneously. Figure 13 shows the block diagram of the new design while Figure 14 and 15 list out the detailed operations.

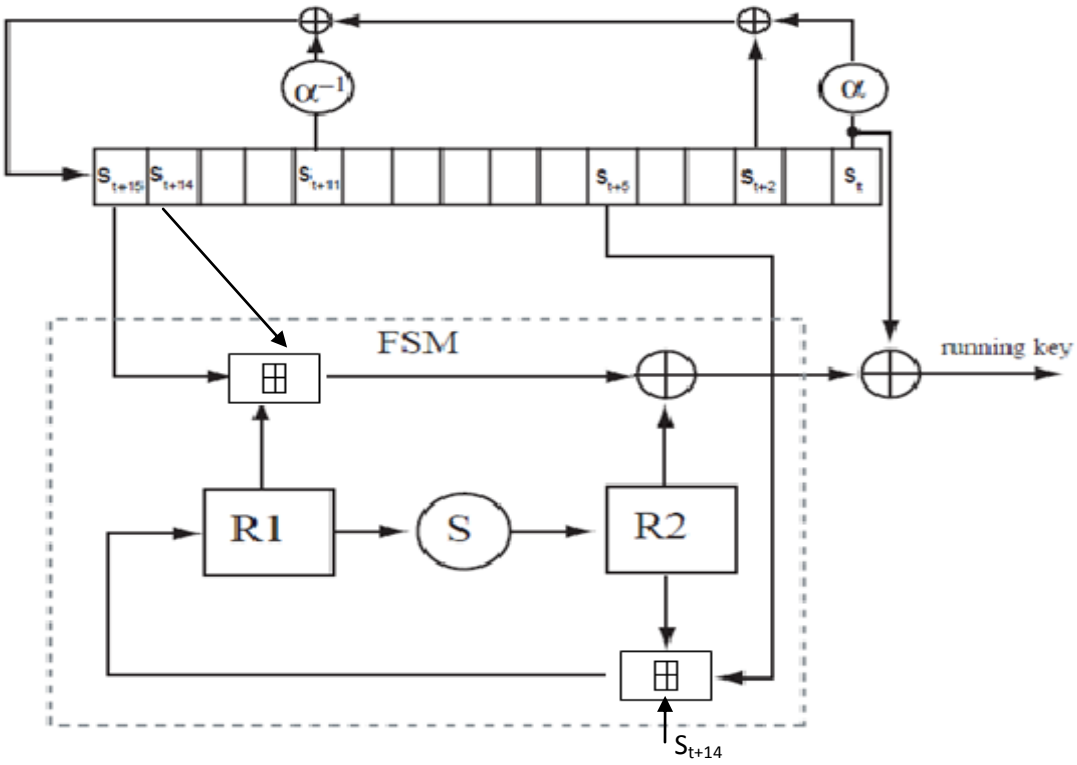

Figure 13 - Application of New Design in SNOW2.0

New SNOW2.0

New Modulo Addition:

X_ext = Input Expansion (X, S14)

Y_ext = Input Expansion (Y, S14)

Sum = Modulo Addition (X_ext, Y_ext)

Z = Output Compaction (Sum, S14)

Return Z

FSM Update:

F = New Modulo Addition (S15, R1, S14)

F = F^R2

R = New Modulo Addition (R2, S5, S14)

R2 = SBox (R1)

R1 = R

Return F

LFSR Init:

V = (S0*α)^(S11/ α)^S2^F

For i = 1 to 16

S(i) = S(i+1)

S15 = V

LFSR Key Stream:

V = (S0*α)^(S11/ α)^S2

For i = 1 to 16

S(i) = S(i+1)

S15 = V

Figure 14 - Operation of the New SNOW2.0

New SNOW2.0

Initialization Mode:

1st LFSR → (S0, S1, …, S14, S15)

Secret Keys → (K0, K1, K2, K3)

IV → (IV0, IV1, IV2, IV3)

S0 = K0^0xFFFFFFFF, S1 = K1^0xFFFFFFF, S2 = K2^0xFFFFFFFF, S3 = K3^0xFFFFFFFF, S4 = K0

S5 = K1, S6 = K2, S7 = K3, S8 = K0^0xFFFFFFFF, S9 = K1^0xFFFFFFFF^IV3

S10 = K2^0xFFFFFFFF^IV2, S11 = K3^0xFFFFFFFF, S12 = K0^IV1, S13 = K1, S14 = K2, S15 = K3^IV0

R1 = 0, R2 = 0

For i = 1 to 32:

    F = FSM Update (S15, S14, R1, R2)

    LFSR Init (F, S)

End For

Key Stream Generation Mode:

FSM Update (S15, S14, R1, R2)

LFSR Key Stream (S)

For i = 1 to N:

    F = FSM Update (S15, S14, R1, R2)

    Z = F^S0

    LFSR Key Stream (S)

End For

Figure 15 - Operation of the New SNOW2.0 Continued

An example is given in the below table to demonstrate the internal steps of applying the new design on S15 and R1 during the generation of the first output key stream. The secret key used is $0xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA$ and the initialization vector used is $0x00000004000000030000000200000001$. This is one of

the test vector listed in the original design. At this specific timeframe, $S15 = 0xCC15A50B$, $R1 = 0xAAB91A68$, and $S14 = 0x5164B6D9$.

x_i	KI_{xi}	x'_i	y_i	KI_{yi}	y'_i	z'_i	c'_i	KO_i	z'
1	001	00000010	0	110	10111111	11000001	0	110	1
1	010	00000100	0	100	11101111	11110011	0	101	11
0	100	11101111	0	100	11101111	11011110	1	011	111
1	010	00000100	1	100	00010000	00010101	0	101	0111
0	110	10111111	0	100	11101111	10101110	1	001	10111
0	010	11111011	1	110	01000000	00111100	1	101	110111
0	010	11111011	1	110	01000000	00111100	1	101	1110111
0	110	10111111	0	100	11101111	10101111	1	001	11110111
1	110	01000000	0	110	10111111	00000000	1	001	011110111
0	110	10111111	1	101	00100000	11100000	0	001	0011110111
1	110	01000000	0	010	11111011	00111011	1	001	10011110111
0	010	11111011	1	011	00001000	00000100	1	101	010011110111
0	101	11011111	1	010	00000100	11100100	0	010	1010011110111
1	000	00000001	0	010	11111011	11111100	0	111	11010011110111
0	101	11011111	0	010	11111011	11011010	1	010	011010011110111
1	100	00010000	0	010	11111011	00001100	1	011	1011010011110111
1	100	00010000	1	011	00001000	00011001	0	011	11011010011110111
0	101	11011111	0	011	11110111	11010110	1	010	111011010011110111
1	101	00100000	0	010	11111011	00011100	1	010	1111011010011110111
0	101	11011111	1	111	10000000	01100000	1	010	01111011010011110111
1	101	00100000	1	010	00000100	00100101	0	010	10111011010011110111
0	100	11101111	1	101	00100000	00001111	1	011	11011110110100111110111
0	101	11011111	0	001	11111101	11011101	1	010	111011110110100111110111
0	010	11111011	1	001	00000010	11111110	0	101	1111011110110100111110111
0	001	11111101	0	001	11111101	11111010	1	110	11111011110110100111110111
0	011	11110111	1	001	00000010	11111010	0	100	11111011110110100111110111
1	001	00000010	0	101	11011111	11100001	0	110	11111011110110100111110111
1	001	00000010	1	101	00100000	00100010	0	110	01111111011110110100111110111
0	011	11110111	0	001	11111101	11110100	1	100	10111111101111011010011110111
0	011	11110111	1	101	00100000	00011000	1	100	11011111110111011010011110111
1	011	00001000	0	011	11110111	00000000	1	100	01101111110111011010011110111
1	011	00001000	1	101	00100000	00101001	1	100	00110111111101111011010011110111

Table 15 - Example Output from the New SNOW2.0

The output of the new design becomes $0x37F7B4F7$, whereas the result of the original Modulo Addition is $0x76CEBF73$. Also, the first output key stream of the new design becomes 0x91CC022F while the original key stream is $0xC355385D$.

6 Application Results and Analysis

In this chapter, the results from the applications in Chapter 5 are analyzed. At the same time, limitations of the new design are discussed.

6.1 Analysis of New Bivium B

To begin Algebraic Attack on the original Bivium B, the adversary needs to formulate the relationship between the states and the output. Two methods can be used to achieve this. The first method simply uses all the states as variables and this requires 177 variables. During each clock, the states can be updated using these variables and the feedback values t1 and t2 can also be described with these variables as well. In addition, the output is the linear combination of t1 and t2 and can be described with the 177 variables. It can be observed that the output equation has an algebraic degree of 1 during the first 66 clock cycles. In other words, the first feedback value, which has an algebraic degree of 2 because of the AND logic, will be propagated to the output after 66 clock cycles. As the clock cycle increases, the degree of the output increases. For each clock cycle, one equation is generated; therefore, this formulation uses 177 variables with n equations for n clock cycles.

The second approach aims at lowering the algebraic degree of the overall equations. This is accomplished by introducing two new variables at every clock cycle. The two new variables are used to represent the feedback value, which includes the degree 2 AND logic. As a result, two new equations are generated to describe the two new variables and the equations always have an algebraic degree 2. The output equation will be limited to degree 1 because the new variables successfully hide the higher degree relationship. Overall, there are $2n + 177$ variables and 3n equations for n clock cycles. In [37], the authors have successfully simulated the attack by using the first method and have had to guess 56 variables in 320 clock cycles. For the second method, 56 variables are also required to be guessed and 2000 clock cycles are needed.

Replacing the output linear combiner function of Bivium B with the new design increases the algebraic degree of the equations and the number of variables required to guess. As deducted in the earlier chapter, the output algebraic degree with the new design can be generalized to $m(1 + 2^m(i + 1))$. In this case, the parameter m is 4 and the output is only 1 bit; therefore, the algebraic degree with the new design is at least 68. The degree is 68 under the assumption that the degree of the inputs to the new design is kept at 1. In other words, this can be achieved by using the second method described above. If the first method is used, the degree is 68 for the first 66 cycles and increases as the cipher progresses.

Moreover, the adversary may try to exploit the corner case scenario for the new design. The corner case relies on all control bits being 0's. The algebraic degree of the output equations will be the same as the old Bivium B using both methods because the effect of the control bits on the algebraic degree is revoked. The adversary can analyze the new system and determine the variables that appear more frequently in constructing the control bits. Then, those variables would be guessed on top of the 56 variables guessed before. Otherwise, the attacker could simply guess 12 bits with probability of 2^{-12} or wait for the corner case to occur with the same probability. The corner case requires all control strings to be 0's, and the algebraic degree falls back to the same as the original design. Table 16 gives a summary of the analysis of the new Bivium B.

	Bivium B (Method 1)	**Bivium B (Method 2)**	**New Bivium B (Method 1)**	**New Bivium B (Method 2)**
Algebraic Degree	≥ 1	1	$\geq m(1 + 2^m(i + 1))$	$m(1 + 2^m(i + 1))$
Equations	n (for n clocks)	3n	n	3n
Variables	177	$2n + 177$	177	$2n + 177$
Variables Guessed	56	56	≥ 56	≥ 56
Corner Case	-	-	2^{-12}	2^{-12}

Probability				
Corner Case Algebraic Degree			≥ 1	1

Table 16 - Result Analysis of New Bivium B

6.2 Analysis of New SNOW2.0

In [27] and [19], two methods have been used to linearize the quadratic equations formed by Modulo Addition. First, the attacker tries to fix all the carries to 0 for two Modulo Additions in 17 consecutive clock cycles. The probability of this occurring is calculated to be $3/4^{(31*2*17)} \approx 2^{-438}$, which is closed to exhaustive search 2^{-576}. In this case, the exhaustive search is defined to be finding the initial states of the cipher, which includes 16 32-bit states and 2 32-bit registers. With the new design, the probability of fixing the carry becomes $2^{-(n*w*17)}$, where $w = 2^m$. If $m = 3$ is used, the probability of fixing all carries to 0 in order to linearize one set of Modulo Addition equations becomes $2^{-(31*8*17)} = 2^{-4216}$ and much is larger than exhaustive search.

Secondly, the attacker tries to manipulate the output characteristics of Modulo Addition to linearize the algebraic equations. In particular, 9 consecutive values of the register R1, which comes from summing the values of the register R2 and state S5, are fixed. The desired output values from the summation are $R1_1 = 0, R1_2 = 2^{32} - 1, R1_3 = 0, R1_4 = 0, R1_5 = 0, R1_6 = 0, R1_7 = 0, R1_8 = 0,$ $and\ R1_9 = 0$. Due to the nature of the LFSR used in the cipher and the structure of the cipher, this condition requires fixing the values in 9 states: S5, S6, S7, S8, S9, S10, S11, S12, and S13. The probability of the 9 states being fixed is in general $2^{-(32*9)}$ or 2^{-288}. This method proves to be better than the first method; however, it is not the case when the new design is applied. As discussed in the earlier chapter, exploiting the output characteristics becomes implausible because Output Compaction function is lossy. In other words, $R1 = 0$ or $2^{32} - 1$ does not guarantee that there is no carries or that all carries exist in the Modulo Addition. In the case where the attacker still wants to fix the output to all 1's, the probability is $2^{-n}2^{-mn}$ as mentioned in the earlier chapter. The

probability then becomes 2^{-128}. In addition, to fix the output to all 0's, the probability is $3^n 2^{-2n(m+2)}$. In this case, the probability becomes $3^{32}2^{-2*32(3+2)} \approx 2^{-205}$. For a total of 9 consecutive cycles, the total probability becomes $2^{-205*8} * 2^{-128} = 2^{-1768}$.

The adversary may wish to utilize the corner case scenario by fixing all the control bits. In theory, 3nm bits need to be fixed and that will give a probability of $2^{-(3mn*9)}$ for 9 consecutive cycles. In this application; however, all the control bits come from the state S14. This means that the probability has now become $2^{-(32*9)} = 2^{-288}$, which is the same as the probability of the second method. It is important to note that the corner case does not completely linearize the new design. The lowest algebraic degree has become 1 and the highest degree is 32 as discussed in table 13. More importantly, the corner case will not happen on the LSB of the output because the input control strings for the LSBs of the input pair cannot be simultaneously 0. This is due to the control string generation mechanics. As a result, the corner case is not applicable. The table below provides a comparison of the two methods in the old SNOW2.0 and the new SNOW2.0.

	SNOW2.0	**New SNOW2.0**
Method 1: Fix Carries to 0	2^{-248}	2^{-4216}
Method 2: Fix Consecutive Outputs	2^{-288}	2^{-1768}
Corner Case	-	NA

Table 17 - Result Analysis of New SNOW2.0

6.3 Limitations of the New Design

6.3.1 Speed

Speed is a limitation of the new design in both software and hardware as more operations are added to the targeted cipher. In the software scenario, there are n times more additions occurring compared to only 1 addition happening for an n-bit Modulo Addition. Moreover, the carries need to propagate to the next block when n > 1. There

are also 2n Input Expansions and n Output Compaction. Control bits generation is required as well. In terms of software implementation, the Input Expansion function can be implemented as a look-up table if the user-defined parameter m is known ahead of time. This would consume less time, though more memory, compared to other implementations. Also, if the expanded input is less than 32 bits, it is possible to lump all the additions into one, as the expanded input can be held in one integer variable. In the applications in Chapter 5, the all functions are implemented using mainly bit-level shift operation. This method is scalable but takes up more time. The time required to perform the new design may be too small to be measured in a software scenario. Therefore, the time is measured by using the first application in Chapter 5. This is a good example because the input data width is only 1 bit. In the below table, Bivium B is used to compare the time added when different values of m are used. The application is carried out in C++ and 1000000 samples are used for averaging. It can be seen that, at least 4.7% of extra time is required to produce a single key stream bit when the new design is applied.

	Time (ns)	**% Increase**
M = 0	697.3212	-
M = 2	730.081	4.7%
M = 3	761.2814	9.1%
M = 4	781.6614	11%
M = 5	796.4611	14.4%

Table 18 - Software Performance Comparison

It is worth noting that, the user-defined parameter can be increased to more than 5. However, the expanded input will be longer than 32 bits and will not be held in one integer variable. In that case, FOR loops may be added and integer-overflow check may be required. The time required may vary depending on different implementations. It is up to the user to decide the tradeoff between time and security.

The speed limitation of the new design in hardware can be estimated. A gate is assumed to be a generic 2-input XOR gate. The gate delay is the time required for the 2-input XOR gate to obtain the correct output value. As a result, the delay can be estimated in unit of gates and by counting the number of gates needed. An example is given below.

$$F = X \oplus \mathrm{Y} \oplus \mathrm{Z} \oplus \mathrm{U}$$

$$Number\ of\ gates: 3. Total\ Delay: 3\ gate\ delays.$$

The Input Expansion function produces an ANF for each expanded input bit. The maximum number of gates in this set of expression is $2^m - 1$, which can be deduced by studying table 5 and table 6. This is the longest delay and also the delay of the Input Expansion function. Note that the gates required to determine each term is ignored; in other words, each term is assumed to be a direct input. Since the Input Expansion function for each input pair can be carried out in parallel, the delay in total is at least $2^m - 1$ for an n-bit design.

The Modulo Addition can be implemented in hardware using various topologies and one of the simplest methods is the Carry-Ripple Adder. In this topology, the Generate and Propagate signals are generated for each input bit pair. Both can be roughly estimated to be using a single gate. When counting the delays, it can be observed that all the individual Generate and Propagate signals are carried out at the same time. As a result, only 1 gate delay requires to be counted. The carry-ripple logic is determined by the group Propagate logic and group Generate logic. The group Propagate logic requires logic ANDing of multiple individual Propagate signals and the previous carries. This can be estimated to roughly $n - 1$ gates for an n-bit adder. The group Generate logic requires logic ORing of the group Propagate signal and the individual Generate signal. This is roughly 1 gate and $n - 1$ gates for an n-bit adder. In total, the delay is $2n - 2$ for an n-bit carry-ripple logic. Finally, another delay is added from the sum of the most significant bit. This is carried out with multiple single XOR gates in parallel; therefore, only 1 gate delay is needed. In summary, the Carry-Ripple Adder provides a delay of

$1 + 2n - 2 + 1 = 2n$ gates. For the addition of two expanded inputs, the delay is extended to $1 + 2n2^m - 2 + 1 = 2n2^m$.

The delay of the Output Compaction function can be estimated easily. Using the SOP form of the MUX function described in equation (7), the number of gates required is $2^m - 1$. For an n-bit design, the total delay of the Output Compaction function is still $2^m - 1$ as more than one MUX can be used in parallel.

In summary, the total delay of the new design in hardware is estimated to be:

$$2^m - 1 + 2n2^m + 2^m - 1 = (n + 1)2^{m+1} - 2$$

Table 18 provides a summary of the estimation.

	Delays (Number of Gates)
Input Expansion	$2^m - 1$
Modulo Addition	$2n2^m$
Output Compaction	$2^m - 1$
Total	$(n + 1)2^{m+1} - 2$

Table 19 - Hardware Delay Estimation

It is worth noting that, this calculation is based on the availability of all required resources. If the hardware resources are limited and they need to be re-used, the delay will change and more than one clock cycles will be required, when using a pipelined design. Architectural synthesis can be used to determine the optimized tradeoff between delay and resources [38].

6.3.2 Area

Continuing from the discussion above, the other limitation of the new design is the resource. The number of gates that will be used can be roughly estimated in a similar fashion as above. Since the maximum number of gates in the input expansion function is $2^m - 1$, the total number of gates used for a block of expanded input can be $2^m(2^m - 1)$. The assumption here is that each expanded input bit uses the $2^m - 1$

gates. Therefore, for two n-bit inputs, the area required for the Input Expansion function is $2n2^m(2^m - 1)$.

For the adder, one gate is needed for producing the Generate or Propagate signal for each input bit pair. As a result, there are total $2n2^m$ gates for the expanded inputs. Similarly, for the group Generate and Propagate signals, there are $2n2^m - 2$ gates in total. Finally, single bit addition is carried out in the form of a single XOR gate. Therefore, there will be $n2^m$ gates for summation. As a whole, the Modulo Addition consumes $2n2^m + 2n2^m - 2 + n2^m = 5n2^m - 2$ gates.

The MUX uses roughly $2^m - 1$ gates for each block. As a result, for an n-bit design, the estimated number of gates used is:

$$2n2^m(2^m - 1) + 5n2^m - 2 + n(2^m - 1) = 2n2^{m+1} + (4n)2^m - n - 2$$

Table 19 provides a summary of area estimation of the new design:

	Number of Gates
Input Expansion	$2n2^m(2^m - 1)$
Modulo Addition	$5n2^m - 2$
Output Compaction	$n(2^m - 1)$
Total	$2n2^{m+1} + (4n)2^m - n - 2$

Table 20 - Hardware Area Estimation

The number of gates that may be used is provided above; however, it is without any optimization. Since the new design is scalable, the user-defined parameter m can be optimized to fit in the specific resource constrained environment. The selection and optimization techniques are open.

7 The Use of New Design in Block Cipher

In previous Chapters, the new design is applied to stream ciphers. One may wonder whether the new design can be applied in a block cipher as well. One of the major differences between a generic stream cipher and a generic block cipher is that the cryptographic functions used in stream cipher do not need to be reversible. As mentioned in Chapter 2, this is because the stream cipher encrypts a plaintext by first generating a key stream and then mixing the key stream with the plaintext. The receiving end only needs to generate the same amount of key stream to reverse the mixing, or to decrypt, the ciphertext. On the other hand, the block cipher makes changes to the plaintext directly to produce the ciphertext. To decrypt, the receiving end needs to run the block cipher in reverse. Therefore, the question of the applicability of the new design in block cipher can be answered by finding out whether the new design can be run in reverse.

During decryption in the block cipher, the receiving end has knowledge of the output, all the control bits, and one of the inputs. Naturally, the decryption should follow the order of Output Compaction, Modulo Addition, and Input Expansion. The Output Compaction function as discussed in Chapter 3 is lossy. Even with the knowledge of the output and the output control bits, only n bits out of 2^{nm} bits can be recovered. This is not useful because there are many combinations of expanded inputs that can arrive to the same n bits. Therefore, a different approach is needed.

7.1 Encryption in Block Cipher

Uniqueness is needed in the recovered bits from the reverse Output Compaction function. This can be achieved on the encryption side by leaving trails in the output. A characteristic in addition is utilized and the below table is used for explanation.

Case	Input 1	Input 2	Previous Carry	Sum	SUM $\oplus$ Previous Carry

A	0	0	0	0	0
B	0	1	0	1	1
C	0	0	1	1	0
D	0	1	1	0	1
E	1	0	0	1	1
F	1	1	0	0	0
G	1	0	1	0	1
H	1	1	1	1	0

Table 21 - Characteristic of Addition

When looking at the sum only, more than one possible input combination can occur. For example, when sum is 0, both case A and case D are possible. Keep in mind that, in the decryption scenario, one of the inputs is known. Therefore, if one input is 0, cases from E to H are not valid. When two inputs are fixed, like in case A and C, only one possible carry is valid. Therefore, case A and C cannot be considered together. Similarly, both case B and C are possible when the sum is 1. One may discover that in both case A and D, or case B and C, the previous carry is different. To distinguish the inputs from the sum, the sum can be XORed with the previous carry. As a result, in the last column of table 17, case A and D are now different. The same applies to case B and C.

As a result, the new design needs to XOR the previous carry bit with the sum bit to leave a trail for decryption. In fact, the result of the “addition” simply becomes the XOR of the two input bits.

7.2 Decryption in Block Cipher

Now that the ciphertext has a clue, the known output control string will locate the position of each ciphertext bit in the corresponding sum. As mentioned before, each ciphertext bit is chosen from a 2^m-bit sum, which is the result of adding two expanded inputs. Also, for each pair of expanded inputs, one of the inputs and the two input control strings are known. Since the ciphertext bit is a XOR of two bits, located by the output control string, in the expanded inputs, the bit in the unknown expanded input is

just another XOR of the ciphertext bit and the bit in the known expanded input. Refer to Table 1 for the truth table of a XOR. This process can be explained below:

$$\text{Let } U_{n-1}, \dots, U_0 \text{ be an unknown expanded input string}$$

$$\text{Let } K_{n-1}, \dots, K_0 \text{ be a known expanded input string}$$

$$\text{Let } C_i \text{ be the ciphertext bit}$$

$$C_i = U_i \oplus K_i; \text{ therefore, } U_i = C_i \oplus K_i$$

Once the corresponding bit in the unknown expanded input is known, the real input can be obtained by checking the output control string against the input control string for the unknown expanded input. When the input control string is fixed, there are only two possible expanded inputs and they are the complement of each other. Also, the polarity of the input is reflected only at the position specified by the input control string. Other bits in the expanded input are of opposite polarity. Therefore, if the output control string, which specifies the position of the deciphered bit, is the same as the input control string for the unknown expanded input, the real input has the same polarity with the corresponding bit. Otherwise, the real input has the opposite polarity.

In summary, the procedure for encryption and decryption in the block cipher can be seen below.

Encryption:

Known: Two inputs, two input control strings, and output control string

1. Input Expansion (Two Inputs, Two input control strings)
2. Modulo Addition
3. Leave Trails
    a. Sum $\oplus$ Previous Carry Based on output control string
4. Output Compaction

Decryption:

Known: One input, two input control strings, output control string, and ciphertext

For i = 1 to N for N-bit ciphertext:

1. Input Expansion (One input, input control string of known input)
2. Corresponding bit = Ciphertext bit $\oplus$ Expanded input bit
    a. Based on output control string
3. If(input control string of unknown input == output control string)
    a. Unknown input bit = Corresponding bit
4. Else
    a. Unknown input bit = ~Corresponding bit

End For

Figure 16 - Encryption and Decryption of New Design in Block Cipher

7.3 Limitations of the New Design in Block Cipher

The application of the new design in a generic block cipher is quite different from the case in stream cipher. One extra step is required in the encryption phase while a different set of logic is required in the decryption phase. This is mainly due to the lossy nature of the Output Compaction function. However, some drawbacks have been exposed.

Performance and resource requirements are challenged when the new design is used in a block cipher. The time for the extra steps and logic required for encryption and

decryption are added on top of the time to perform the three functions: Input Expansion, Modulo Addition, and Output Compaction. Similarly, extra hardware is required.

The algebraic degree has also been compromised in this case. As mentioned before, leaving the trails means simply XORing the two expanded input bits. The added algebraic degree from the previous carry bit in the Modulo Addition is cancelled out. The algebraic degree of the final output only contains the degree of the Input Expansion and the Output Compaction. From Chapter 4, this can generalized to $\mathrm{m} + \mathrm{m} = 2\mathrm{m}$. Though it is much lower than the case in stream cipher, it is still better than the traditional Modulo Addition.

8 Conclusion

8.1 Summary

In this Project, a new design is proposed. It consists of a user-defined scalable structure of expansion and compaction that can increase the operational complexity but still keep the data width of the design. The structure utilizes a scalable Input Expansion function, a Modulo Addition, and a scalable Output Compaction function. The functions used in this design are chosen to be scalable. An algebraic degree of $2\mathrm{m} \rightarrow \mathrm{m}(1 + 2^{\mathrm{m}}(\mathrm{n} + 1))$ for an n-bit design is produced while the degree of the corner case is n, with an added cost of 2^{mn}. Another advantage of the design is that the functions can be substituted to achieve other objectives as long as they provide expansion or compaction. The new design can be easily applied in a stream cipher as two case studies are provided and analyzed. The application in block cipher is also discussed. Speed and Area are limitations of the new design. In software, an extra 4.7% of the original run time is needed to produce a single key stream bit. In hardware, the delay in speed is estimated using gate delays while the extra resource required is estimated by counting the number of extra gates. Overall, the proposed design provides a new module in cryptography that can help defend against Algebraic Attack.

8.2 Future Work

Future work may include the optimization of the user-defined parameter under different performance and resource constraints. Also, the realization of the design in software and hardware can be optimized as well. Other functions can be developed to substitute the existing functions in the Input Expansion and Output Compaction functions to provide different or extra cryptographic features. Meanwhile, further Algebraic cryptanalysis may be done to discover the possibility of creating extra independent equations with lower degrees [39] [40].

References

[1] A. Braeken, B. Preneel, “On the Algebraic Immunity of Symmetric Boolean Functions,” in *Progress in Cryptography – INDOCRYPT 2005*, Springer Berlin Heidelberg, pp. 35-48.

[2] C. Carlet *et al*, “Algebraic Immunity for cryptographically significant Boolean functions: analysis and construction,” *IEEE Trans. Inf. Theory*, vol. 52, no. 7, pp. 3105-3121, Jul. 2006.

[3] C. Carlet, K. Feng, “An Infinite Class of Balanced Functions with Optimal Algebraic Immunity, Good Immunity to Fast Algebraic Attacks and Good Nonlinearity,” in *Advances in Cryptography – ASIACRYPT 2008*, Springer Berlin Heidelberg, pp. 425-440.

[4] F. Armknecht, M. Krause, “Constructing Single- and Multi-output Boolean Functions with Maximal Algebraic Immunity,” in *Automata, Languages, and Programming*, Springer Berlin Heidelberg, 2006, pp. 180-191.

[5] Y. Nawaz, K.C. Gupta, G. Gong, “Algebraic Immunity of S-Boxes Based on Power Mappings: Analysis and Construction,” *IEEE Trans. Inf. Theory*, vol. 55, no. 9, pp. 4263-4273, Sep. 2009.

[6] C. Adams, “Designing against a class of algebraic attacks on symmetric block ciphers,” *Applicable Algebra in Engineering, Communications, and Computing*, vol. 17, no. 1, pp. 17-27, Apr. 2004.

[7] C. Adams, S. Tavares, “Designing s-boxes for ciphers resistant to differential cryptanalysis,” *Proceedings of the 3rd Symposium on the State and Progress of Research in Cryptography*, pp. 181-190, Feb. 1993.

[8] A. Menezes, P. van Oorschot, S. Vanstone, “Stream Ciphers,” in *Handbook of Applied Cryptography*. CRC Press, 1997, pp. 200-201.

[9] B. Schneier, “Psuedo-Random-Sequence Generators and Stream Ciphers,” in *Applied Cryptography*. 2nd ed. Wiley Computer Publishing, 1996, pp. 323-324.

[10] H. Wu, “Cryptanalysis and design of stream ciphers,” Ph.D. thesis, Dept. Elect. Eng., Katholieke Universiteit Leuven, Belgium, 2008.

[11] M. Matsui, “Linear Cryptanalysis Method for DES Cipher,” in *Advances in Cryptography – EUROCRYPT’93*, Springer Berlin Heidelberg, 1994, pp. 386-397.

[12] E. Biham, A. Shamir, “Differential cryptanalysis of DES-like cryptosystems,” *Journal of Cryptography*, vol. 4, no. 1, pp. 3-72, Jan. 1991.

[13] J. Patarin, “Hidden Fields Equations (HFE) and Isomorphisms of Polynomials (IP): Two New Families of Asymmetric Algorithms,” in *Advances in Cryptography – EUROCRYPT’96*, Springer Berlin Heidelberg, 1996, pp. 33-48.

[14] J. Patarin, “Cryptanalysis of the Matsumoto and Imai Public Key Scheme of Eurocrypt’88,” in *Advances in Cryptography – EUROCRYPT’95*, Springer Berlin Heidelberg, 1995, pp. 248-261.

[15] N. Courtois, W. Meier, “Algebraic Attack on Stream Ciphers with Linear Feedback,” in *Advances in Cryptography – EUROCRYPT 2003*, Springer Berlin Heidelberg, 2003, pp. 345-359.

[16] N. Courtois, “Algebraic Attack on Combiners with Memory and Several Outputs,” in *Information Security and Cryptography – ICISC 2004*, Springer Berlin Heidelberg, 2004, pp. 3-20.

[17] N. Courtois, J. Pieprzyk, “Cryptanalysis of Block Ciphers with Overdefined System of Equations,” in *Advances in Cryptography – ASIACRYPT 2002*, Springer Berlin Heidelberg, 2002, pp. 267-287.

[18] W. Meier, E. Pasalic, C. Carlet, “Algebraic Attacks and Decomposition of Boolean Functions,” in *Advances in Cryptography – EUROCRYPT 2004*, Springer Berlin Heidelberg, 2004, pp. 474-491.

[19] N. Courtois, B. Debraize, “Algebraic Description and Simultaneous Linear Approximations of Addition in Snow 2.0,” in *Information and Communications Security*, Springer Berlin Heidelberg, 2008, pp. 328-344.

[20] S. Fischer, W. Meier, “Algebraic Immunity of S-Boxes and Augmented Functions,” in *Fast Software Encryption*, Springer Berlin Heidelberg, 2007, pp. 366-381.

[21] C. Adams, “Constructing Symmetric Ciphers Using the CAST Design Procedure,” in *Selected Areas in Cryptography*, Springer US, 1997, pp. 71-104.

[22] B. Scheier *et al*, *The Twofish encryption algorithm: a 128-bit block cipher*, New York, NY, Wiley, 1994.

[23] C. Burwick *et al*, “MARS – a candidate cipher for AES,” IBM Corp., Rep., 1998.

[24] P. Hawkes, G. Rose, “Primitive specification and supporting documentation for SOBER-t32 submission to NESSIE,” *Proceedings of the first open NESSIE workshop*, 2000.

[25] P. Ekdahl, T. Johansson, “A New Version of the Stream Cipher SNOW,” in *Selected Area in Cryptography*, Springer Berlin Heidelberg, 2003, pp. 47-61.

[26] “Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. Document 2: ZUC Specification,” Rep. version 1.6, Jan. 2011.

[27] O. Billet, H. Gilbert, “Resistance of SNOW 2.0 Against Algebraic Attacks,” in *Topics in Cryptography – CT-RSA 2005*, Springer Berlin Heidelberg, 2005, pp. 19-28.

[28] N. Weste, D. Harris, “Datapath Subsystems,” in *CMOS VLSI DESIGN*, 4th ed., Boston, MA, Addison-Wesley, 2011.

[29] R. Merkle, “Fast Software Encryption Functions,” in *Advances in Cryptology-CRYPTO’90*, Springer Berlin Heidelberg, 1991, pp. 477-501.

[30] R. Zhang, L. Chen, “A block cipher using key-dependent S-box and P-boxes,” *IEEE ISIE 2008*, pp. 1463-1468, Jul. 2008.

[31] J. Peng, S. Jin, “Designing Key-Dependent S-Boxes Using Hyperchaotic Chen System,” in *Proceedings of the International Conference on Information Engineering and Applications (IEA) 2012*, Springer London, 2013, pp. 733-740.

[32] S. Harris, C. Adams, “Key-Dependent S-Box Manipulations,” in *Selected Areas in Cryptography*, Springer Berlin Heidelberg, 1999, pp. 15-26.

[33] C. Cid, M. Robshaw, “The eSTREAM Portfolio in 2012,” ECRYPT, Rep. version 1.0, Jan. 2012.

[34] C. D. Canniere, “TRIVIUM: A Stream Cipher Construction Inspired by Block Cipher Design Principles,” in *Information Security*, Springer Berlin Heidelberg, 2006, pp. 171-186.

[35] L. R. Simpson *et al*, “LILI Keystream Generator,” in *Selected Areas in Cryptography*, Springer Berlin Heidelberg, 2001, pp. 248-261.

[36] A. S. Raj, C. Srinivasan, “Analysis of Algebraic Attack on TRIVIUM and Minute Modification to TRIVIUM,” in *Advances in Network Security and Applications*, Springer Berlin Heidelberg, 2011, pp. 35-42.

[37] I. Simonetti, J. C. Faugere, L. Perret, ”Algebraic attack against trivium,” in *Proceedings of the First International Conference on Symbolic Computation and Cryptography*, SCC, vol. 8, pp. 95-102, 2008.

[38] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*, 1st ed., McGraw-Hill, 1994.

[39] F. Armknecht, “On the Existence of low-degree Equations for Algebraic Attacks,” IACR Cryptographic ePrint Archive, 2004.

[40] X. M. Zhang, J. Pieprzyk, Y. Zheng, “On Algebraic Immunity and Annihilators,” in *Information Security and Cryptography – ICISC 2006*, Springer Berlin Heidelberg, 2006, pp. 65-80.

[41] N. Courtois, J. Patarin, “About the XL Algorithm over *GF*(2),” in *Topics in Cryptography – CT-RSA 2003*, Springer Berlin Heidelberg, 2003, pp. 141-157.

[42] N. Courtois, “Higher Order Correlation Attacks, XL Algorithm and Cryptanalysis of Toyocrypt,” in *Information Security and Cryptography – ICISC 2002*, Springer Berlin Heidelberg, 2002, pp. 182-199.

[43] N. Courtois *et al*, “Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations,” in *Advances in Cryptography – EUROCRYPT 2000*, Springer Berlin Heidelberg, 2000, pp. 392-407.