Design of a Backdoored Block Cipher and its Evaluation

By
Humayun Ajmal

A thesis submitted to the faculty of Information Security Department, Military College of Signals, National University of Sciences and Technology, Rawalpindi in partial fulfillment of the requirements for the degree of MS in Information Security

February 2022

THESIS ACCEPTANCE CERTIFICATE

This is to certify that final copy of MS thesis written by **NS Humayun Ajmal** Student of **MSIS-18,** Reg.No **00000318940** of **Military College of Signals** has been vetted by undersigned, found complete in all respects as per NUST statutes / regulations / MS Policy, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS degree. It is further certified that necessary amendments as pointed out by GEC members and local evaluators of the scholar have also been incorporated in the said thesis.

Signature ______________________________

Name of Supervisor (**Asst Prof Dr. Fawad Khan**)

Date: ____________ 2022

Signature (HoD)______________________________

Date: ____________ 2022

Signature (Dean / Principal)___________________

Date: ____________ 2022

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification either at this institution or elsewhere. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and acknowledgements

Humayun Ajmal

February 2022

Dedication

"In the name of Allah, the most Beneficent, the most Merciful"

I dedicate this thesis to my late father, family, friends who supported and encouraged me during every step of this study and remained a source of inspiration and strength.

I owe everything presented in this study to my teachers and fellow students who guided me throughout this phase and shared their words of advice and encouragement to finish this study,

Acknowledgments

All praises to Allah for the strengths and His blessing in completing this thesis.

I would like to convey my gratitude to my supervisor, Dr. Fawad Khan for his supervision and constant support. His invaluable help, constructive comments and suggestions throughout the experimental and thesis works are major contributions to the success of this research. Also, I would thank my committee members; Asst Prof Dr. Shahzaib Tahir, and Assoc Prof Dr. Muhammad Faisal Amjad and Maj Wajahat Sultan for their support and knowledge regarding this topic.

Last, but not the least, I am highly thankful to my parents and family. They have always stood by my dreams and aspirations and have been a great source of inspiration for me. I would like to thank them for all their care, love and support through my times of stress and excitement.

Abstract

Symmetric key block ciphers are employed for data encryption widely in everyday applications. Block ciphers are considered robust and secure even if their structure is known globally and immune to several cryptanalysis attacks including linear and differential cryptanalysis. Data retrieval should only be possible with the help of data encryption key.

However, there may exist block ciphers whose structure is transparent, but may contain an inherent algebraic backdoor helping the designer to retrieve the data encryption key. Undiscoverable algebraic backdoors are hard to design because of the hidden mathematical structure employed to retrieve the key. Moreover, it is also interesting to explore statistical methods in order to determine an inherent mathematical backdoor.

In this thesis, we explored various backdoor embedding methodologies previously, and have employed the recent LowMC-M framework to design a backdoored cipher. Furthermore, we applied the standard NIST statistical suite tests against the backdoored cipher and the standard AES to explore which statistical methods might help to determine the underlying backdoor cipher

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since the beginning of this millennium, Information technology is one field that has witnessed a revolution. Researchers now agree that the world's most valuable resource is no more gold or oil, but data. Technology have become an essential component in almost every field, generating and contributing to the next gold reservoir, the Big Data**.** Considered as a repository of every sort of information, from financial to medical, health to classified to personal information, it is growing at an exponential rate every day.

However, with Big Data comes big responsibility. The responsibility to secure it during storage and during communication, so that it remains safe from hackers, eavesdroppers, commercial organizations and even hostile nations, willing to access it for their own interest. Cryptography has attempted to provide an answer to evade these attempts, even as early as around 1900 BC by Egyptians, as recorded in ancient history. Every organization, therefore, wants its information to be secure from eavesdroppers and it wishes to guarantee its users that their data is secure during every stage of its handling.

Cryptology has evolved from simple ciphers used by Egyptians and Romans to complex algorithms. Scientific community has researched and developed Cryptographic protocols / algorithms (like RSA, DHKE, DES, AES, RC4, SHA etc), augmented by employing a Symmetric (single key) or asymmetric (public key) cryptography system. The use of these crypto systems has been limited by performance issues; computational power and battery requirement etc when it comes to their deployment with IoT devices. Therefore, many light weight symmetric key ciphers have been proposed to address these issues.

Cryptology is broadly divided into two branches, i.e. Cryptography and Cryptanalysis. These fields are very closely related and, in a sense, complement each other. Cryptography deals with the development of cryptosystems whereas cryptanalysis ensures that these crypto systems remain free from vulnerabilities that may lead to a compromise. A crypto system may seem very strong and efficient, but researchers will strive to find out design flaws that may give away information and resultantly lead to a compromise. Cryptanalysis ensures that a cipher design is free from such vulnerabilities so that the encrypted data remains secure during all phases of communication and storage.

Breaking these ciphers and retrieving information has always seemed a tempting task. Hackers, Governments and Intelligence agencies have been trying since long to break the cryptographic means used for securing data. Possibility of brute-force effort to find the secret key virtually been ruled out by the use of complex algorithms resistant to cryptanalysis, larger key space and limited computational power.

Backdoors can provide solution to these problems. A backdoor is a way of by-passing encryption. Encrypted information can be recovered by anyone knowing the backdoor and not knowing the secret key.

Broadly speaking Backdoors are divided into two categories :-

i. Embedded in a product at key scheduling, sharing or at protocol level

ii. An algebraic backdoor which is implemented at the design level of cipher

Existence of backdoor in a cryptographic algorithm (DES, for instance) has always been a debatable question in academia and research community, further fueled by recent proof of concept of existence of a backdoor in NIST Dual_EC_DBRG and revelation by Edward

Snowden leaks about existence of Cryptographic backdoors in NSA algorithms. The news about Swiss firm AG Crypto, selling rigged machines capable of breaking the codes to 3rd world countries including India and Pakistan have been termed as the “intelligence coup of the century”.

Tweakable block ciphers provide one approach to answer this issue. While designing a block cipher, the designers can embed a backdoor that is not easy to detect. This will allow the designer to either retrieve the key or part of the key which will substantially reduce the brute-force effort. The backdoor is designed and embedded in such a fashion that even if its existence is known, it is computationally impossible to retrieve.

The research in this field gives rises to a general suspicion that most of the publicly available block ciphers can also have backdoors which have not been claimed by the designer in implementation. However, it is a complex battle between remaining secure and exploiting a system. A designer may consider his design secure and share it publicly for researchers to exploit it. A researcher may find a vulnerability and NOT declare it, creating a false sense of cryptographic security. Researchers continue to find new ways to attack a cryptosystem and come up with new ideas to break a cryptosystem.

1.1 Problem Statement

With the advancement and easy access to information, use of technology that was once considered highly sophisticated by ordinary people is no surprise. One of the major concern of LEAs and Government remains the use of encryption and covert channels by criminals, which helps them to dodge surveillance. Pakistan remained embroiled for over a decade in combating terrorism. Pakistan decided to ban use of VPNs [1] and social media applications including Telegram in 2011, since these applications were extensively used by terrorists owing to their ease of use and easy access to internet even in the remotest of areas.

Surveillance, therefore, remains a major contributor towards intelligence gathering. On the other hand, encryption remains a preferable choice for anyone desirous to hide from Government tailing. In order to remain one step ahead of these criminals, Governments want access to any covert communication that takes place between these criminals. Where breaking a cipher is not feasible in time sensitive scenarios, deploying a cryptographic backdoor in a cipher may seem to address the problem. The research proposes an AES-like block cipher based on SPN architecture, with a mathematical backdoor which helps to retrieve key or part of key that would help in deciphering ciphertext.

1.2 Motivation

Back doored cryptosystems can be employed by Government organizations, LEAs and intelligence agencies. A back doored cryptosystem based communication system (mail and messaging applications etc.) will help LEAs and intelligence agencies for law enforcement and national security requirements. Moreover, government organizations can keep an eye on policy implementation and ensuring communication security by randomly checking communication. Designing a block cipher with a mathematical backdoor is the main motive behind this study, which may help anyone knowing certain parameters in deciphering the Ciphertext without knowing the Secret Key at the expense of large computational effort. Key recovery, however, is not the goal of this research.

1.3 Research Objectives

This research focuses on following objectives:-

1.1.1 Developing a cipher similar to AES that contains a mathematical backdoor

1.1.2 Analyzing effectiveness of linear and differential crypt analysis

1.1.3 Applying statistical analysis to observe how conservative the algorithm is in terms of randomness

1.4 Contributions

The design of the new cipher will contribute to the existing research in following ways:

a. Develop an understanding of the cryptographic backdoors by providing detailed literature review

b. Develop an understanding of a block cipher, how linear and non-linear layers function, and discuss the impact of linear and differential cryptanalysis of such functions.

c. Design and working of non-linear layers (S-Box) of a block cipher which can help in understanding presence of backdoors in block ciphers.

d. Paving a way for future work with an endaveour to embed cryptographic backdoors in an AES-like block cipher.

1.5 Thesis Outline

The research work has been organized and distributed in following chapters:

- **Chapter 1**: Chapter 1 presents a brief introduction to Cryptographic backdoors. It also presents the problem statement, followed by motivation behind the research and enumerates research objectives. Lastly it highlights the contributions made through this research.
- **Chapter 2**: **Existing Research in Cryptographic Backdoors**.

- Chapter 2 presents an overview of the existing / recent research that has already been carried out in the field of cryptographic backdoors.
- **Chapter 3**: **Preliminary Background**. This chapter gives an insight of the preliminary background knowledge that is vital in understanding block cipher designs and cryptanalysis techniques used against them.
- **Chapter 4**: **Construction of A Block Cipher with An Embedded Algebraic Backdoor**. The chapter presents an introduction / brief description of the LowMC Block Cipher which form the basis of our backdoored design. It will also elaborate on underlying concepts of how an algebraic backdoor is embedded in LowMC cipher.
- **Chapter 5: The Designed Cipher – Security and Performance Analysis.** This chapter would review the security as well as the performance analysis of the cipher that we have designed. It will also present an analysis of how our cipher compares to AES.
- **Conclusion**

Chapter 2

Existing Research in Cryptographic Backdoors

This chapter highlights previous significant research work carried out in the field of Algebraic cryptographic backdoors. An idea of having a backdoor in block ciphers has been a catalyst for academia, however, it has proven to be a difficult task. Presently, there are two ways of implementing backdoors in block ciphers :-

a) Embedding a backdoor at protocol or key management level of the cipher

b) A cryptographic, mathematical or algebraic backdoor that is embedded in the design of a cipher and exposes the cipher to some form of cryptanalysis.

The first type of backdoor is the commonly used type which is easy to design. The second type, however, is difficult to implement and will form the base of this thesis.

2.1 Challenges in Backdoors implementation

Though the concept of mathematical backdoors has opened new avenues for public research, yet only limited work has been done in this field so far, primarily due to facts stated below :-.

1) It is difficult to embed secure backdoors in Block Cipher design since they are deterministic in nature and very less likely to evade the research by hackers and researchers while studying their design.

2) In order for a backdoor to be successful, the effort for key recovery and correspondingly information extraction from only ciphertext should practically be less than the brute-force effort.

3) Designers would not like to reveal the presence of any backdoor inside their design, since it will affect the confidence of users on the product. Once such backdoors are discovered, the researchers / organizations will not publish this fact due to any security / financial gains associated with it.

2.2 Research in the field of Backdoors

Eli Biham, in 1994, proposed the idea of attacking key scheduling algorithms [2] that may have inherent relationships between keys. He proved that these relationships can be used to carry out attack against a block cipher. The paper explores the concept that Key scheduling implementation in most of the block ciphers designs can be seen as a sequence of algorithms. Each of these algorithm computes and derives a particular round key or sub-key from sub keys of previous rounds. If the key scheduling algorithms for key derivation is same for all the rounds, then it is possible to shift given sub-key one round backwards to retrieve all the sub-keys and ascertain the relationship between the keys. Similarly, for another given key, all the valid sub-keys can be derived using the same relationship, called Related Keys. Therefore, if the structure of key scheduling algorithm is kept simple it will expose the cipher to Related key attacks. Both the attacks were mounted against LOKI89 and LOKI91, the two variants of LOKI block cipher. The authors displayed that related-key attacks do not exploit any vulnerability in DES and concluded how key-scheduling algorithm can seriously compromise the design of a cipher.

Adam Young and Moti Yung extensively studied cryptographic backdoors and in 1996 coined the term [3] “Kleptography” which they described as the study of stealing information from cryptosystems in a secure fashion. They explained that “subliminal channels” can be designed within a cryptosystem which would place a hacker at an advantage of retrieving information without the knowledge of users. The paper explains that the use of “black box” cryptographic devices with trusted internal structure can have backdoors which will enable hackers to steal the secrets in an invisible manner. Their paper proposed a “Secretly Embedded Trapdoor with Universal Protection” or SETUP, a trapdoor implementation technique which can be embedded in cryptosystems like El-Gamal, RSA, DSA etc and will leak encrypted secret key information. However, instead of containing an information leaking “subliminal” channel, the design would provide opportunities for the attacker to retrieve information from the output of the cipher.

Rijmen and Preneel [4] suggested several methods to embed trap-doors in block ciphers [3] in 1997. They defined partial trapdoors, which may either give partial information about the key or do not work for all keys. The paper discusses the design of a trapdoor in a *m x n* S-Box (where *n* represents the number of S-Boxes implemented in a round and *m* represents the number of input bits to an S-Box) by adding an extra function T(x) (which is the Trapdoor function) and is derived from the S-Box S(x). The researchers displayed that with the higher value of *m x n*, it is easier for the cipher to hide a trapdoor. The paper concludes that a trapdoor hidden in a 10 x 80 bits S-box is virtually undetectable, thereby verifying that the possibility of having a trapdoor hidden in a seemingly legitimate block cipher exists. Though the embedded backdoor remains undetectable, even when the general design and characteristics of the cipher are known, yet the paper emphasis that trusting a cipher whose design is secret is not advisable since it may contain a trapdoor which is hard to detect. The cipher design was subsequently broken by Wu et al. in 1998 [5], who concluded that the backdoor proposed by Rijmen and Preneel can be broken by applying linear cryptanalysis techniques proposed by Matsui **[**6**]**. The trapdoor can be discovered if its global design is known but not the parameters. In 1999 Paterson **[**7**]** proposed a DES-like cipher structure containing a backdoor, which was based the idea that when a round function acts on a message group, it can generate a group which can be imprimitive. In this case, the design of the cipher will contain an inherent weakness which can be exploited, allowing construction of a backdoor based on this weakness. Anyone knowing the backdoor will be able to retrieve the key with 2^{41} permutations. However, it concluded that the backdoor was easily detectable.

Patarin and Goubin studied new cryptosystem [8], which was based on the asymmetric cryptosystem " C* " and was proposed earlier in [9]. These cryptosystems were based on the idea of hidden s-box computations with a secret function known only to the designer. These functions were of one or two degrees. "C*" was concluded as a special case, however, it did not contain the algebraic properties of "C*", which subsequently led to its cryptanalysis [10]. Therefore, these ideas were taken forward to explore different cryptanalysis techniques that could exploit the algebraic characteristics of these ciphers and led to introduction of completely new cryptanalysis tools.

In 2002, Liskov, Moses, Ronald L. Rivest, and David Wagner [11] introduced the concept of Tweakable class of Block Ciphers. The idea was based on the concept of a nonce for OCB mode or IV in CBC mode. A conventional block cipher has two input components, a key, a plaintext and produces an output, the ciphertext. The idea behind was the address the deterministic behavior of block ciphers, where a plaintext will always produce the same ciphertext, if it is encrypted with the same key.

In 2013, Angelova, Vesela, and Yuri Borissov [12] studied design of block ciphers and highlighted the weaknesses of S-Boxes that have flaws in their design, despite fulfilling some design criteria and will result in very weak ciphers. It describes an attack that can exploit these weaknesses and help in recovering plaintexts in DES-like ciphers that have poorly or improperly constructed S-boxes. The paper discussed DES / triple-DES like ciphers in ECB mode with modified S-Boxes.

In Bannier, Arnaud, and Eric Filiol [13] the authors discussed the general design of block ciphers and proposes a block cipher embedded with a backdoor. The block cipher was based on an encryption system which is vulnerable to cryptanalysis as suggested by Matsui in [5] and which enables an attacker to retrieve the secret κ-bit key with a single plaintext / cipher text pair. Bannier, Arnaud, and Eric Filiol [14] proposed an AES-like cipher named BEA-1, which had an algebraic / mathematical backdoor which is implemented in the design of cipher and had following characteristics :-

- 80-bit Block size
- 120-bit Key size
- 11 rounds

The designers were able to recover 120-bits of key in 10 seconds with only 300 kb of plaintext and 300 kb of corresponding ciphertext. The authors of the backdoors claim that the backdoor still remains undiscoverable despite sharing its design.

2.3 Low Multiplication Complexity Ciphers

SPNs are constructed using a non-linear and a linear layer. Non-linear layers (S-Boxes) are cost heavy in terms of execution times and therefore effect the overall performance of the design. Low Multiplication Complexity or LowMC is a class of AES-like ciphers that is designed to achieve Low Multiplication Complexity by employing partial non-linear layers and a strong linear layer. A partial non-linear layer is designed in a fashion that S-Boxes only act on a part of the layer and not the complete layer. This helps reduce the computational overload as presented by Arnaud Bannier, Nicolas Bodin, and Eric Filiol [15].

In 2020, Peyrin, Thomas, and Haoyang Wang [16] introduced LowMC-M, a malicious instantiation of a LowMC variant. LowMC-M is based on a related-key attack and has an additional tweak input, as shared by [1] and [10], which helps recover the secret key using differential cryptanalysis. LowMC-M employs a partial non-linear layer in its design and compensates for the security using a strong Linear layer.

2.4 Conclusion

In 2006, NIST proposed an algorithm, Dual_EC_DBRG classified as a CSPRNG and became part of NIST SP 800-90A as a standard in 2007. However, in 2007, it was revealed [17] that the DBRG contained a design flaw that could be termed as a "trap door", a type of backdoor. The news story was exposed in 2013 by the newspapers, The Guardian [18], and *The New York Times* [19] while analyzing the memos shared by Edward Snowden, and commented that the design flaw was deliberately kept by NSA, allowing one having the secret NSA-points on the standard EC to reconstruct the secret key being used.

Similarly, the Washington Post in February 2020 [20] published a news story about Swiss firm AG Crypto, selling rigged machines capable of breaking the codes to 3rd world countries including India and Pakistan. The story has labelled this event as the "intelligence coup of the century", and explains how NSA established AG Crypto as a front-end company to sell rigged machines to third world countries.

There has always been a debate on existence of *Backdoors* in commercial / public cipher algorithms that no not claim existence of a backdoor otherwise. With the discovery of such backdoors, users have become more suspicious and careful, and has opened new avenues for researchers, who have been working hard to find any evidence of existence of a backdoor embedded by the designer.

Backdoors in block ciphers, therefore, can be embedded in their design or protocol level, the later being easily detectable. Designing a block cipher with a mathematical backdoor, however, is a difficult task since its discovery will seriously affect the credibility of the cipher and designers both. Apart from this, anyone discovering the backdoor can use it for its own benefit.

In subsequent chapters, we will discuss the underlying concepts involved in the design of a mathematical backdoor.

Chapter 3

Preliminary Background

Design of all the cryptosystem revolved around the Kerckhoffs's principle of cryptography which states that a cryptosystem must be secure if its design and everything, except the key, is a public knowledge. In other words, the strength of the cryptosystem will depend on the key and not its algorithm. Our thesis will be restricted to the *Symmetric Key* Cryptography only which is one of the two sub-domains of Cryptography.

3.1 The Symmetric Key Cryptosystems

The Symmetric Key Cryptosystems use identical key for doing the *encryption* and *decryption operation*. When a user encrypts *plaintext* by using a key, it must be *decrypted* with the same key. Therefore, the encrypting party ensures that all the parties who would be requiring to decrypt a *plaintext* must also be in possession of the *Secret Key*. Handling and distribution of Secret Key is beyond the scope of this thesis.

Symmetric Key Cryptosystems are further divided into *Block* and *Stream* Ciphers. In this thesis, we would focus our attention towards *Block Ciphers* only and after explaining the general primitives, will discuss *Tweakable Block Ciphers* which are pertinent to our thesis.

3.1.1 Block Ciphers and SPN Networks

A block Cipher generates permutations on a fixed length of bits, called a *Block*. These permutations are *indexed* or *controlled* by a *secret Key*. Consequently, a block Cipher will have two algorithms: an *encryption algorithm* and a *decryption algorithm*.

To keep it structurally simple and to increase its cryptographic strength, a *Block Ciphers* use a technique to iteratively update its internal state multiple number of times (as intended by the designer) after it has been initialized by a *plaintext*. This technique is called 'round function'. Again, the *Secret Key (K)* or the *Master Secret Key* is passed through a *Key Scheduling Algorithm* that will calculate a set of round keys, one to be used for each round, so that each round does not depend on the *Master Secret Key* and depends on the *round Key* only.

The design of round keys may follow one of the two design frameworks, a *Feistel Network and a SP-Network.* Here SP stands for *Substitution-Permutation.* Though the design of the round function may differ, yet the rest of the primitives will remain the same.

3.1.2 Tweakable Block Ciphers

A special type or variant of Block Ciphers is "*Tweakable Block Ciphers*", which were introduced by Rivest, Liskov and Wagner [21]. Here, the block cipher is designed to accepts an additional input value known as 'Tweak" and encrypts a message M which is controlled by two entities, the key 'K' and a "tweak" T. Both values act together to encrypt a message to produce a cipher text C. Therefore, we can represent a tweakable block cipher as

$$\tilde{E}(K,T,P) : \{0,1\}^k \times \{0,1\}^t \times \{0,1\}^n \rightarrow \{0,1\}^n$$

3.2 Components of a Tweakable Block Cipher

Classically, all Symmetric encryption systems are built using four components. However, in the thesis our design will revolve around a Tweakable Block Cipher and therefore we shall be dealing with an additional *Tweak Input* as well. We will describe these one by one before exploring each in detail.

A Typical Tweakable Block Cipher is illustrated in *Figure 3.1* :

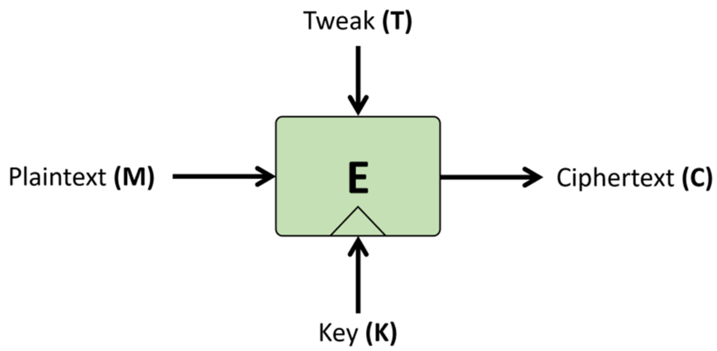


Figure 3.1: A TBC

3.2.1 The Plain Text (M)

The message, clear text or literal text forms the input to a cipher algorithm. This is the text that the sender wants to encrypt so that the adversary is not able to read it during communication or storage if he accesses it. The Cipher receives a fixed length of plaintext bits known as a *Block.* The length or size of the Block fed to the Encryption algorithm as an input will remain fixed for a given scheme of Block Cipher.

3.2.2 The Encryption / Decryption Algorithm (E)

The component of encryption system that takes the *Plain text message* (M) and generates a *Cipher Text* (C) under control of the *key* (K) is known as the *Encryption* (E) */ Decryption (D)Algorithm*. The Block Cipher, classically, will take a *Block of Input Text,* will perform Encryption (E) process, and creates a *cipher text* (C). This process will be done under control of the *key* (K).

3.2.3 The Master Key (K)

The master key (K) is a string of random characters which is known to only the sender and receiver. It is used to encrypt and decrypt the data and therefore, is kept secret.

1) The Key string is kept random so that it is difficult to guess and is long enough for making the brute-force (or trying all possible combinations) effort infeasible.

2) A ***Key scheduling algorithm*** is used to generate keys, since block ciphers use more than one round for encryption.

3.2.4 Cipher Text (C)

The result of the *Encryption Process* (E) is the *Cipher Text* (C). This is the output of a Block Cipher which, for an adversary, is unintelligible. Therefore, if an adversary wants to derive information from *Cipher Text* (C), it has to be reverted back to its original state using the *Decryption Process* (E).

3.2.5 The Tweak Input (T)

For the Tweakable Block Cipher to work, an additional input is also provided to the Block Cipher known as *Tweak Input* (T).

3.3 The Advanced Encryption Standard (AES)

The objective of our research is to design an AES-like SPN cipher, therefore we will explain the brief working of AES in this section, to develop an understanding of how AES works.

3.3.1 The AES Design and Working

The Advanced Encryption Standard (AES) is an *iterative, Substitution – Permutation Network* Cipher. Being an iterative Cipher, it has multiple rounds, the number of which are determined by the length of the *Key,* whereas the round function remains the same. Table 3.1 and the figure 3.2 shows the various key lengths and corresponding rounds.

Table 3.1 AES key lengths and rounds

Key Length	Number of rounds	Comments
128-bits	10	
192-bits	12	
256-bits	14	

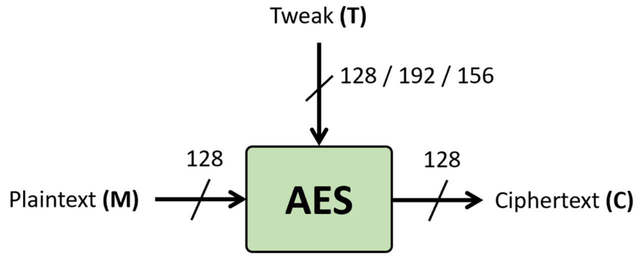


Figure 3.2 AES Block Diagram

The AES working is described in subsequent sections and shown in *Figure 3.3 :*

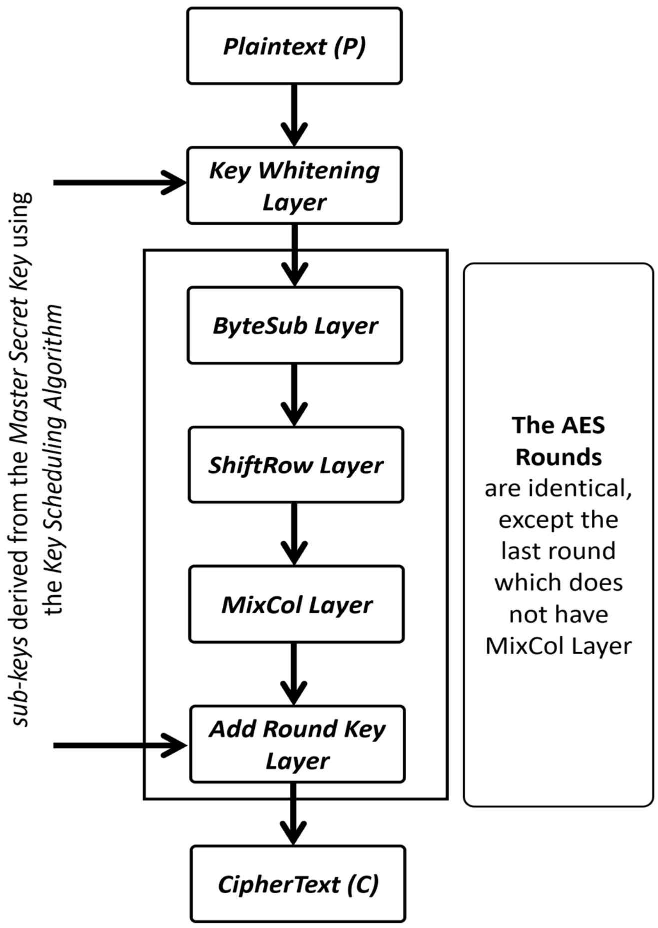


Figure 3.3 AES Operations Flowchart

3.3.2 Plain Text Handling by AES

AES is designed to perform computations on *Bytes*, rather than the *Bits.* Therefore a 128-bit data block in AES is treated as 16 bytes. AES further arranges these 16 bytes into a 4x4 matrix for processing in GF(2^8).

3.3.3 Rijndael's Key Scheduling Algorithm for Key Expansion

The same Secret Key is not used for every round, instead the *Master Secret Key* is used to determine a set of *Sub-Keys*. The series of "round keys", one key for every round, is calculated or derived using the *Rijndael's key scheduling algorithm.*

3.3.4 The Key Whitening Layer

The initially driven 128-bit Sub-Key or $K_{0,}$ (which is calculated from the *Master Secret Key* by using the *Key Scheduling Algorithm*, as explained in 3.3.2) is XORed with the state. This is done before the start of AES rounds operation to perform a *Key Whitening operation.*

3.3.5 The AES Rounds

The length of the *Master Secret Key* determines the number of rounds in AES (as discussed in Table 3.1 above). The number of rounds will be numbered from 0 to $N_r - 1$, where

N_r ε {10, 12, 14}.

Except the last round, AES round functions comprise of following transformations in the order given below. We will not go into the detailed working of these operations :-

3.3.5.1 Byte Substitution Layer (ByteSub or S-Box Layer)

Byte Substitution Layer is the non-linear layer that would apply identical S-Box permutation to every bit of the state and non-linearly transforms the state by making use of special lookup tables that have special mathematical properties.

3.3.5.2 Shift Rows

The Shift Row operation shifts or rotates (in a cyclic manner) the i^{th} row of the state by i bytes, where $i = 0, 1, 2, 3$ (the number of row). The process is shown in Figure 3.4.

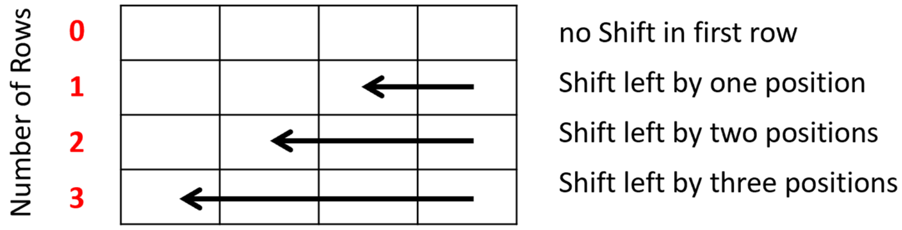


Figure 3.4 AES Shift rows operation

3.3.5.3 Mix Column

The Mix Column function operates on each column of the state. It starts by treating each individual column as a vector and multiplying it with a 4x4 fixed matrix.

3.3.5.4 Round Key Addition Layer

Towards the end of every round, a *Round Key Addition operation* is performed which existing stat is XORs with the *Round Key*. As discussed earlier, *Round keys* are driven from the Master Secret Key through Key Scheduling Alorithm.

3.3.6 The Cipher Text (C) and its Decryption

The output of the AES is a block of encrypted data that can be securely stored and communicated over an un-encrypted network, provided the *Secret Key* is kept secure.

Being a SP-Network based, the decryption operation is merely an inverse of the encryption operation and layers along with their order are inverted.

3.4 Cryptanalysis

The strength of a cipher lies in the *Secret Key*. Even if every detail of the cipher is public and its design and working is known, the cipher should be considered secure till the time the *Secret Key* is not known. Therefore, the objective of an adversary is to find a way to recover the *Secret Key*. The adversary can achieve this in a many ways.

The attacker is assumed to have gained access to the Cipher and it can submit queries in the form of Plaintexts / Ciphertexts and receive a corresponding reply (Ciphertext for plaintext and a plaintext for ciphertext). An attacker can launch an attack in a *white-box context* where the internal state or the design of the cipher is known to the attacker, or a *black-box context* where the internal state or the design of the Cipher is not known to the attacker. In the later, the adversary is dependent on encryption / decryption queries only. The table 3.2 describes the various scenarios in a black-box context that an adversary may use to launch an attack.

Table 3.2 Attack Scenarios

Ser	Type of attack	Description
1.	*Ciphertext only attack or Known Ciphertext attack*	During this scenario, the attacker or adversary has access to ciphertexts only.
2.	*Chosen Ciphertext Attack (CCA)*	In CCA, the adversary obtains the plaintexts against a set of ciphertexts for further analysis
2a.	*Adaptive CCA*	In this type, after receiving the plaintexts for the chosen Ciphertext attack and analyzing them, the adversary can request for plaintexts for additional Ciphertexts.
3.	*Known plaintext Attack*	Here, the adversary possesses certain pairs of plaintexts along with their generated ciphertexts
4.	*Chosen Plaintext Attack (CPA)*	The adversary can select random plaintexts which are required to be encrypted and obtains their corresponding cipher texts

		after encryption
4a.	*Adaptive chosen-plaintext attack*	In this model, after receiving the cipher texts for the chosen plaintext attack and analyzing them, the adversary can request for ciphertexts for additional plaintexts.

The attacker can choose to exploit *Keys* as well. As discussed earlier, an iterative cipher employs a *Key Scheduling Algorithms* for generating round keys. The attacker, therefore, can attack using *Original Secret Key* or the *Round Keys*. This is done in either of the following manners [22]: -

1) **Single-key attack**: the attacker can only make queries to the cipher by making use of the *master key K*.
2) **Related-key attack**: Both Original secret key K, as well as a related key K_1, can be used to make queries using to the cipher. In case the Cipher makes use of a weak or simple Key Scheduling Algorithm, it is easy for the attacker to determine the relationship and derive further keys.
3) **Chosen-Tweak attack**. The attacker can also analyse the tweak input, determine and use a relation between tweaks for attacking a Cipher by selecting a tweak value. This is known as *Chosen-tweak attack*.

Symmetric Key Block Ciphers, inherently being deterministic in nature, are susceptible to a number of attacks. A number of bits, or a block, when encrypted with a key, will always produce the same cipher text. This characteristic of a block cipher makes it vulnerable to a number of cryptanalysis techniques, the most common being the *Differential* and *Linear Cryptanalysis*. These attacks, when combined, form the basis of new type of attacks e.g. Boomerang attack and meet-in-the-middle attacks.[23].

3.4.1 Differential Cryptanalysis

Eli Biham was the first to introduce Differential Cryptanalysis in 1991 [24]. This type of cryptanalysis is a *chosen cryptanalysis attack*, where the probability of existence of a *differential*, i.e. an input-out difference pair is exploited.

For instance, let us consider a Cipher which has input $X = [X_1\ X_2\ X_3 \ldots. X_n]$ and consequent output *as* $Y = [Y_1\ Y_2\ Y_3 \ldots\ Y_n]$. Then for each plaintext input X^*, there will exist a corresponding Ciphertext output, Y^*.

Consider X' and X'' and two inputs such that their corresponding outputs exist as Y' and Y''. Then the difference between the two inputs is given by

$\Delta X = X' \oplus X''$ i.e. the input difference equals the XOR of X' and X''

with $\oplus$ being the bit-wise XOR of the two n-bit input values, And therefore

$\Delta X = [\Delta X_1\ \Delta X_2 \ldots\ldots \Delta X_n]$

Where $\Delta X_i = X_i' \oplus X_i''$, where X_i' *and* X_i'' represent the *i-th* bit of X_i' *and* X_i''.

Similarly

$\Delta Y = Y' \oplus Y''$

And therefore

$\Delta Y = [\Delta Y_1\ \Delta Y_2 \ldots\ldots \Delta Y_n]$

Where $\Delta Y_i = Y_i' \oplus Y_i''$, where Y_i' *and* Y_i'' represent the *i-th* bit of Y_i' *and* Y_i''.

Therefore, in a truly random cipher, the probability that given a particular ΔX is given as input and a certain output difference ΔY exists, will be $1/2^n$, (where *n* is number of bits in *X)*. However, this will not be true in every case and in some cases, the probability of occurrence will be high. A pair of input-output differences $(\Delta X, \Delta Y)$ with high probability of occurrence is known as a *differential*.

For constructing the complete *differential* of a SPN Cipher having multiple rounds with *Plaintext Input* as *X* and *Ciphertext Output* as *Y*, we calculate the *Differential Characteristics* of each round (i.e. input and output differences). For this, we shall examine the properties of individual S-Boxes of the cipher to determine the *Differential* (ΔX, ΔY) with highest occurring probability. This is done by calculating a *Difference Distribution Table (DDT)* for each S-Box, from where the probability of occurrence of a specific ΔY against a specific ΔX is determined. For each S-Box in a round, the ΔY with highest probability of occurrence is chosen for a given ΔX. This way, the non-zero ΔY bits from a round relates to non-zero ΔX bits of the next round. This gives us a high-probability difference from *start* (*input) of the Cipher* to the *input of the last round*. This is termed as *Constructing Linear Approximations for the Complete Cipher.*

Being a *Chosen-Plaintext Attack*, the adversary is allowed to select a pair of inputs X' and X'' so that a particular ΔX is satisfied. Moreover, the attacker knows that what values of ΔY value would occur with high probability therefore, he is at liberty to choose input pairs to get the corresponding ΔX.

We also have to consider the fact that we are to find the key for this Cipher. It is concluded that the *Key* will not effect the input difference and will cancel out when XORed.

After a *Differential Characteristic* for *R-1 round* has been determined in a R-round Cipher, we can attempt to recover key bits of the subkey of last round which we can term as *Target Partial Round Key*. Since a single S-Box in a specific round receives a small portion of the state, its output (ciphertext) can be brute-forced by XORing the Ciphertext with all the values of the *Target Partial Round Key.* The resultant *vector* or *Ciphertext* value is passed back through the all the respective S-Box. We will do this for all plaintext / ciphertext pairs and a counter for each recovered *Target Partial Sub Key* is kept. This value is incremented if the *linear expression* is found to be true after a *Partial Decryption* using a *Target Partial Round Key*. After all the *Target Partial Sub Keys* have been tested, the counters are checked. Thy *key* whose counter is found to deviate the greatest from half of the Plaintext / Ciphertext numbers is presumed to be the *Target Partial Sub Key* bits.

3.4.2.1 Linear Cryptanalysis

In subsequent paragraphs we will explain the Linear Cryptanalysis attack against a Substitution-Permutation Network Cipher.

Matsui discovered that the plaintext, cipher text and sub-key bit share a high probability relationship which can be expressed in the form of Linear Equations. The attack works with an assumption that an attacker has access to a random set of plaintexts and their corresponding ciphertexts, with a clause that he has no control over selection of plaintexts that are being used. Thus this is a *Known Plaintext Attack*.

First, we try to find out a linear approximation in our SPN Cipher to the last round. We start with a *portion of the cipher* and expresses it form of a linear expression (linearity being a binary XOR operation). The equation can be written as:

$$X^* \oplus Y^* = 0$$

The linear equation is formed in following manner

$$X_{i1} \oplus X_{i2} \oplus \ldots\ldots. \oplus Y_{j1} \oplus Y_{j2} \oplus \ldots\ldots\ldots \oplus Y_{jm} = 0$$

where

Xi represents the i-th bit of the input $X = [X_1, X_2, \ldots]$

Y_j represents the *j-th bit* of the output $Y = [Y_1, Y_2, \ldots]$.

This equation is representing XOR of u input and v output bits

So basically, we find expressions like equation above that have a high or low probability of occurrence. The existence of linear expressions of above form with a high or low probability is an indicator of poor randomization abilities and can subsequently be exploited in the *Linear Cryptanalysis attack*.

To proceed, we select random values of u and v, and place them in the equation above. The probability that the above equation will hold will be exactly ½. The deviation from this probability value is known as *Linear Probability Bias,* and forms the basis of a Linear Cryptanalysis Attack.

The expressions that are highly linear are constructed by taking into account the input and output bits of an S-Box and find out *linear vulnerabilities* in a S-Box. So for a S-Box that handles input $X = [X_1, X_2, X_3, X_4]$ and output $Y = [\ Y_1, Y_2, Y_3, Y_4]$, we shall examine all linear approximations and compute Linear Probability Bias for each.

Therefore, for one particular *linear equation,* by applying all the 16 input values of the S-Box and examining the corresponding output, we will find out the *probability bias,* i.e. the number of times this expression holds true. We can formulate the *Linear Approximation Table* of the S-Box by using all of its *linear approximations.*

Later, we can concatenate the linear approximations of multiple S-Boxes together so that we can come up with a linear expression which only contains the bits from plaintext and input bits from the last round.

The Key-recovery process is similar to what is done in Differential Cryptanalysis. We find out the *R-1 round linear approximation* for a *R-round* Cipher and with a large *probability bias,* which makes it possible for us to launch an attack to recover last round key bits or *Target Partial Sub Key.* Rest of the process is the same as the key recovery process of Differential Cryptanalysis.

3.5 Backdoors in Block Ciphers

So far, we have also discussed the general structure of Block Ciphers and focused on its classic example; the AES. In order to defeat encryption, researchers and hackers’ resort to tools like *Linear* and *Differential Cryptanalysis*, which forms basis of other type of attacks like *Boomerang Attack* etc. Privy of the fact, the designers of a crypto system take into account these cryptanalysis techniques during the design phase of the cipher and try to make the Cipher as much resistant to modern day cryptanalysis techniques as possible.

Considering their public use, it becomes equally cumbersome for the law enforcement agencies to decrypt any captured encrypted communication which made use of publicly available strong cryptosystems, like AES for example. So, over a period of time, thought was given to have a simpler method, like a backdoor, which is only known to designers, Governments and law enforcement agencies and would give them control over how encryption systems work. However, they would work best if they are only known to the designers.

Backdoors are regarded as the best way to implement cryptographic control over Ciphers. Theoretically, they require far less effort than brute-forcing a cipher. Consequently, they are ideal for use by governments and law enforcement agencies who want to control or by-pass encryption.

As mentioned in Introduction earlier, embedding a backdoor in block ciphers is a challenging task since exploit randomness in computations is difficult due to their deterministic behavior. Moreover, researchers and hackers both are always at the lookout for exploring a loophole in an encryption algorithm which can help in recovery of sensitive data and circumvent encryption, thus exploitation.

Broadly speaking when it comes to implementation, backdoors are categorized into two main types:-

- A backdoor can be embedded in a system at either the key scheduling, generation, distribution or management phase. These are more suited methods, being easier to implement.
- An algebraic backdoor which is implemented at the mathematical design level of cipher. These are considered difficult to implement and not much significant work exists in this field. A mathematical backdoor should assist its designer (or anyone who is in knowledge of a mathematical backdoor) in an effective cryptanalysis and help in recovering the key on a modern-day computer with limited plaintext / ciphertext pairs)

3.5.1 Characteristics of Backdoors in Ciphers

In order to have a strong cipher with an Algebraic backdoor, it is assumed that it will fulfill certain requirements, which make the design practical and workable [16]. These include :-

- Even if the general form of the cipher or internal design is known to an adversary, retrieving backdoor information should still remain computationally difficult.
- The security of backdoor (effort involved in recovering the backdoor) should be equivalent to that of cipher. In other words, retrieving the backdoor should be as difficult as brute-forcing the cipher, otherwise the security of backdoor will defeat the strength of cipher.
- The backdoor should perform an attack that is deemed practical or provide such an information that would significantly reduce the brute force effort for the designer.

Merely reducing 2^{256} to 2^{128} may seem great advantage theoretically, but it would still remain practically infeasible.

- Finally, the designed block cipher with an embedded backdoor must be protected in the classical sense, that is, it should not be vulnerable to state-of-the-art cryptanalysis techniques.

3.5.2 Security Notions

For a Backdoor to be achieve both the purposes, i.e. being practical and secure at the same time, it must adhere to following security notions:-

1) ***Undetectability***. The Backdoor must be embedded in a manner that it would remain undetected. Thus, Undetectability represents the inability of researchers and hackers to comprehend that a covert Backdoor exists in the Cipher.
2) ***Undiscoverability.*** This notion represents the inability of researchers and hackers to find a hidden Backdoor embedded in a block cipher, even if they somehow know that a hidden Backdoor has been embedded in the cryptographic algorithm.
3) ***Untraceability.*** This notion states that if an adversary uses the backdoor to launch an attack, no information about the existence or working of Backdoor is revealed.
4) ***Practicability.*** The last and the most important security notion is the Practicability. It defines that when an entity is in knowledge of the Backdoor and it intends to launch an attack for key recovery, it should be practical and should allow the key recovery without much effort.

Above in view, it is therefore considered that designing a backdoor which meets above criteria is virtually impractical, and not much of a research exists in this field and the topic remains of extreme interest for academia.

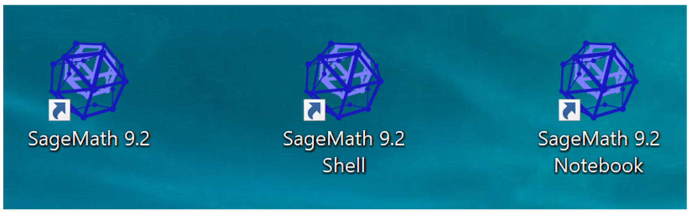
3.5.3 The SageMath Tool

Lastly, we will briefly introduce the SageMath Tool. The SageMath is an *Open-source*, mathematical system licensed under GPL. It is a library which is constructed on top of many other free and open-source libraries like NumPy, matplotlib etc. These libraries can be accessed through a Command-Line Interface (CLI) which is based on *Python*, a renowned programming language.

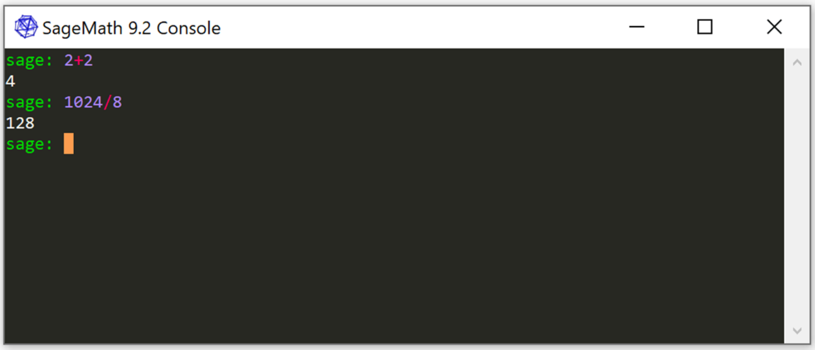
SageMath can be downloaded from *https://www.sagemath.org/download-windows.html* and will work on any 64-bit windows (windows 7 onwards) or from GitHub (https://github.com/sagemath/sage-windows/releases).

3.5.3.1 SageMath Components

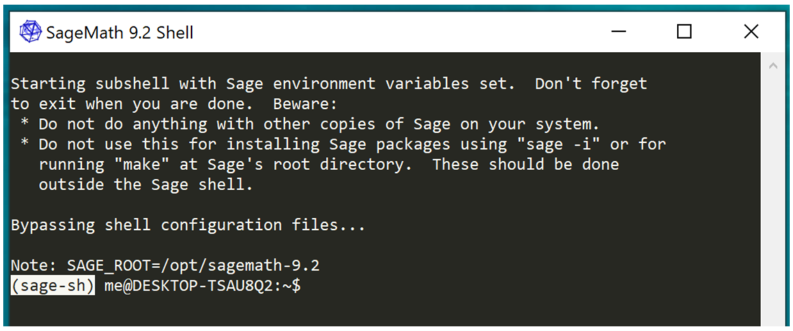
A normal SageMath installation can be run through three desktop / start menu shortcuts. The normal convention is SageMath <version>.



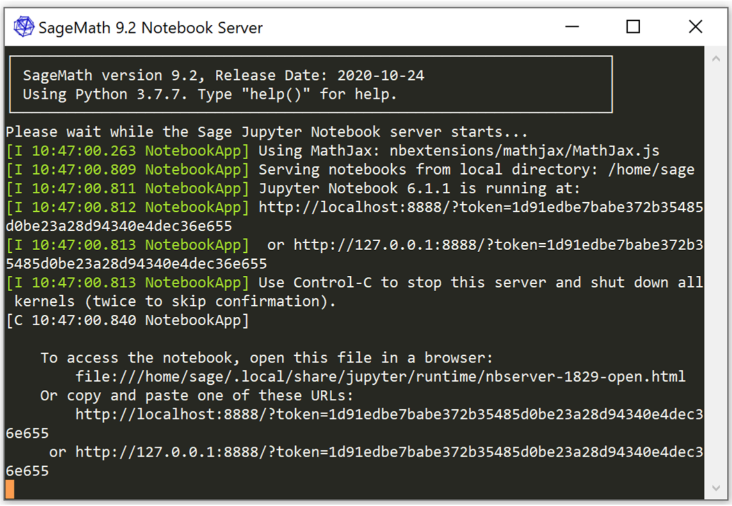
1) **SageMath 9.2** . The basic ***Sage:*** command Prompt can be accessed through the SageMath console. It is a CLI where we can enter commands and execute them. For example, we can simply write 2 + 2 on the command line and SageMath Console will sum these up and give 4 as output in the next line.

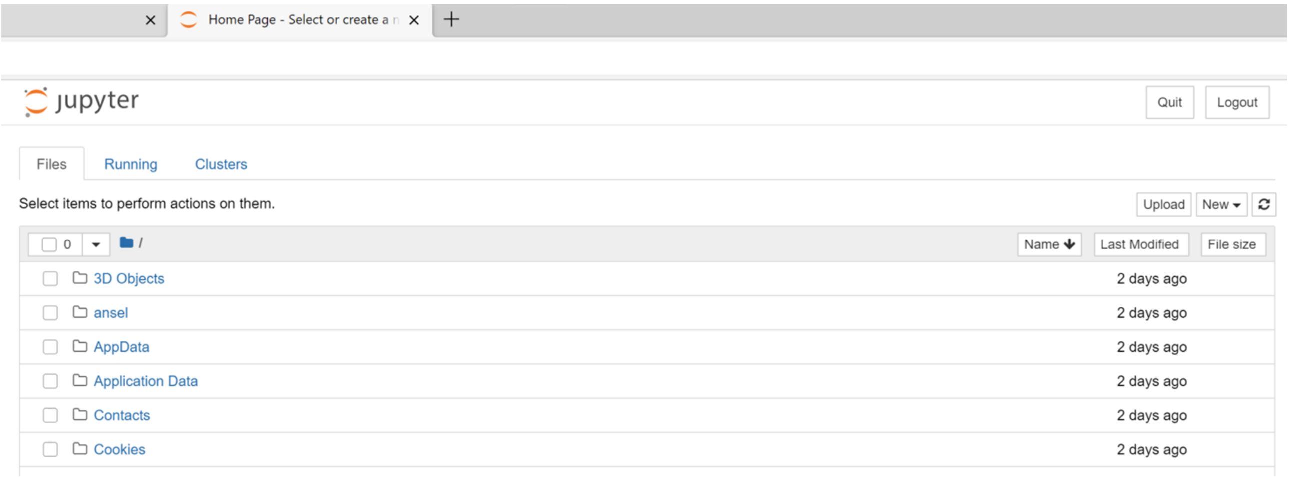


2) **SageMath 9.2 Shell**. This shortcut a *bash-shell* which is intended for advanced users accustomed to use SageMath in a UNIX-like environment (Linux, for example).



3) **SageMath 9.2 Notebook.** The SageMath 9.2 Notebook starts a *Jupyter NoteBook Server* which can run *Jupyter Notebooks* in a *Sage Kernel* (i.e. we can run *Sage* inside *Jupyter*). Running the Notebook will execute the *Notebook Server* in a command-line environment and open the NoteBook in our default browser.





3.5.3.2 Utility

For this thesis, we shall be making use of SageMath to write our code for the Cipher. The coding is done in *Python* and reason for selecting SageMath is its built-in collection of libraries which would otherwise require to be imported in Python.

3.6 Conclusion

Brute-force and cryptanalysis are two tools that are employed for breaking a cipher and recovering key information. During the design phase, designers use Cryptanalysis tools to analyses the system and try to find out design vulnerabilities which may help in recovering key. Addressing these vulnerabilities will not only strengthen the cipher, but make the task of attacker more difficult. Similarly, for a law enforcement agency, this task will be equally difficult considering longer keys and ciphers that are resistant to cryptanalysis.

Researchers are now considering embedding backdoors in the ciphers. To recapitulate, a backdoor is a hidden way of bypassing security in a cryptographic algorithm with an aim to facilitate the designer (or anyone who is in knowledge of Backdoor) in key recovery. Implementation of a backdoor may be easy, but keeping it secure so that hackers and researchers do no discover it and use it for breaking the cipher is a cumbersome and difficult task.

This research focuses around the design of a framework that can be used to embed a mathematical or algebraic backdoor in an *AES like, SPN based tweakable block cipher*. We will start with a broader overview of the cipher and then discuss each component threadbare.

Chapter 4

Block Cipher with an Embedded Algebraic Backdoor

After having gone through the preliminary background in the preceding chapters, we shall discuss the framework design of a backdoored block cipher and its code.

Designing a weak cipher susceptible to cryptanalysis (linear or differential) has an inherent weakness since anyone can exploit these weaknesses and retrieve the secret key. Such a design will not contribute to a practical solution where the backdoor (cryptanalysis) is not only *discoverable* but *detectable* and *traceable*. Therefore, we need a design that is workable and exhibits strong cryptographic properties and resists cryptanalysis. In short, it should behave like any other cipher, but has a secret backdoor element that is known; however, the secret element value is not easily discoverable or retreivable. This secret backdoor will help an attacker (who is in knowledge of the backdoor) in key recovery.

For designing the backdoored cipher, we employ the LowMC-M framework [16]. The framework generally generates the parameters for a traditional LowMC cipher with embedded backdoor. However, the concrete framework does not contain the underlying design code to make the backdoored cipher. We have designed and coded a cipher which is compatible with the LowMC-M framework and initiates by taking parameter values generated by the framework. The encryption / decryption algorithm running inside the LowMC-m code will be explained where required. Main emphasis will be on the explanation of concept, theory and coding being done to embed the Backdoor inside the Block Cipher.

The LowMC uses randomly generated matrices for *Whitening Key* and *Linear Layer matrix* multiplication. This is essential because of the following reasons: -

1) If the underlying matrices are fixed, it will make the cipher deterministic, leading to correct decryption of the ciphertext. This is also the well-known fact for AES as well in which the underlying matrix employed in encryption and decryption operations is fixed.

2) Fixed Matrices or values (without any mathematical justification) leads to a suspicion that the designer might have embedded a mathematical backdoor in the design by specifically choosing these matrices. Due to the underlying design of the LowMC cipher for being lightweight, every time we instantiate the cipher, the random parameters are generated, which are then fixed by both the sending and receiving parties for encryption / decryption.

4.1 The Generic LowMC Framework

Our Cipher is designed based on the LowMC class of Block Ciphers which is a *Low Multiplication Complexity Cipher*. This family of Block Ciphers belongs to the SPN architecture and utilizes a partial non-linear layer.

4.1.1 The design Overview

Based on a conventional SPN design, the LowMC cipher comprises of an initial *Key Whitening* stage followed by a round function which is iterated *r* number of times. The general diagram of the LowMC is illustrated in *figure 4.1.* Each layer will be described in subsequent paragraphs.

4.1.2 LowMC-M Framework: The LowMC with a Backdoor

The LowMC-M framework is a variant of LowMC framework. The LowMC-M framework transforms the LowMC into a tweakable block cipher (TBC) with *hidden high-probability* *differential characteristics* which embeds a Backdoor. The additional input of the TBC is controlled by a Tweak value which is added before the rounds as *Whitening Tweak* and during each round as *round tweak* [26]. The general form of the LowMC and LowMC-M framework is illustrated in *figure 4.1*.

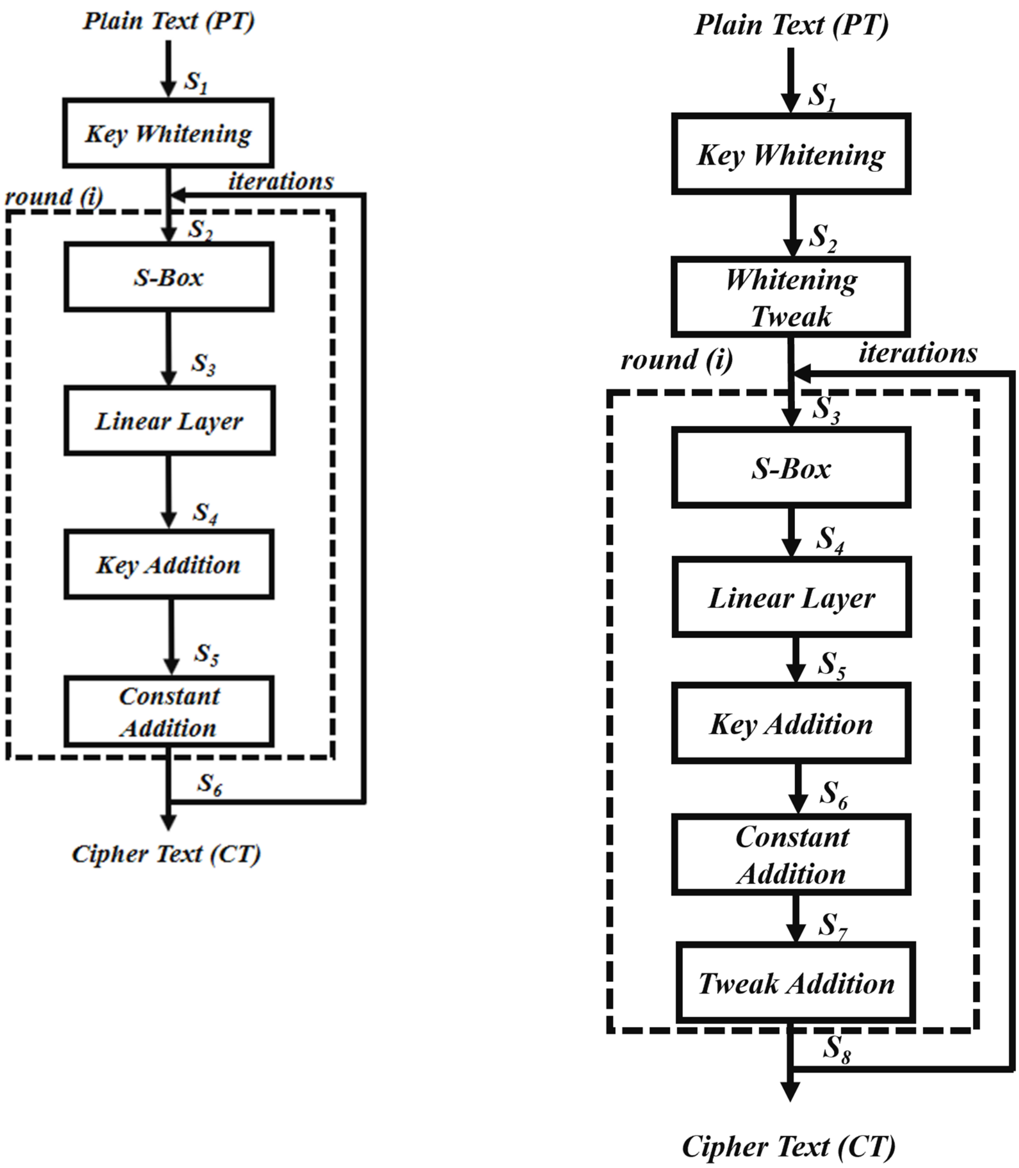


Figure 4.1 The Block Diagram of LowMC (left) as compared to LowMC-M (right)

4.2 Tweakable Block Ciphers

The LowMC-M instantiation is designed basing on a *Tweakable Block Cipher* (TBC) with a *partial nonlinear layer*. The TBC is designed as such so that the *Tweak Value* and the *partial nonlinear layer* are used to embed differential characteristics over a number of rounds. With the knowledge of the *Tweak Value*, it is easy for an attacker to recover full or part of *Secret Key*. The *Tweak Value,* therefore, acts as the backdoor.

4.2.1 Tweak Value

In order for the tweakable block cipher to work, a *tweak value* is given as an additional input to the cipher during its various stages. The addition of same tweak value in all stages of the cipher will be not serve the purpose and will cancel out during the Cryptanalysis. Figure 4.2 shows the block diagram of how *Tweak values* are added in LowMC-M.

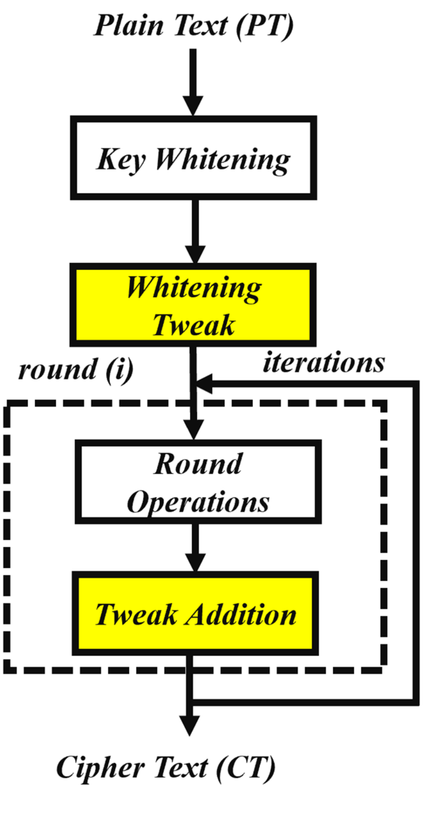


Figure 4.2 Tweak Addition in the LowMC Framework

The tweak value is generated from a randomly selected tweak pair using an *extendable-output function* (XOF). XOF, as its name suggests, is a variation of a HASH function which can produce an output of a desired length.

Since we intend using this as the backdoor, therefore it is assumed that it is known to the designer who, however, is not in possession of *Secret Key* and wants to retrieve it.

Following steps are involved in tweak generation phase of the cipher, which are depicted in Figure 4.3:-

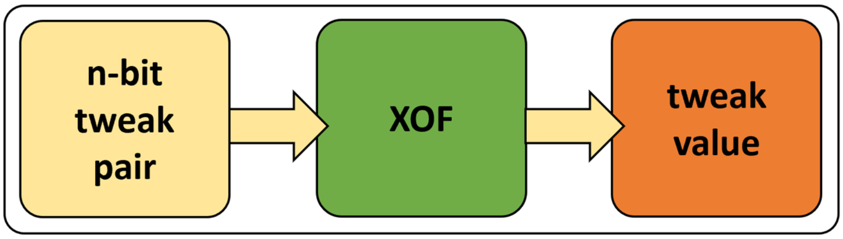


Figure 4.3 Tweak Value Generation

1. A *n-bit* random tweak pair (t1 & t2) is selected. This tweak pair is used to generate the *Tweak Value*.
2. Each *Tweak Value* (t1 & t2) is fed to an extendable-output function (XOF) which in turn generates a *Hash Value*. An XOF can generate a desired length output, which can be used to generate the *Tweak Schedule*.
3. The output of the XOF is XORed to generate the *Master Tweak Value.*

Master Tweak Value
1. Select n-bit tweak values (tweak_1 & tweak_2). These values are random
2. Compute following:
XOF(tweak_1) → $\mathbf{t_1}$
XOF(tweak_2) → $\mathbf{t_2}$
3. Evaluate the difference
$t_0 = \Delta t = t_1 \oplus t_2$
4. The evaluated value is the t_0 Master Tweak Value

4.2.2 The Tweak Schedule

The code uses SHAKE128 as XOF, therefore the length of input i.e. tweak_1 = tweak_2 = 128. The XOF uses these 128-bit vectors to generate t_1 and t_2. The length of t_1 and t_2 is fixed such that the whitening and round tweaks can be derived from it. The output of the XOF is depicted in figure 4.4 below:-

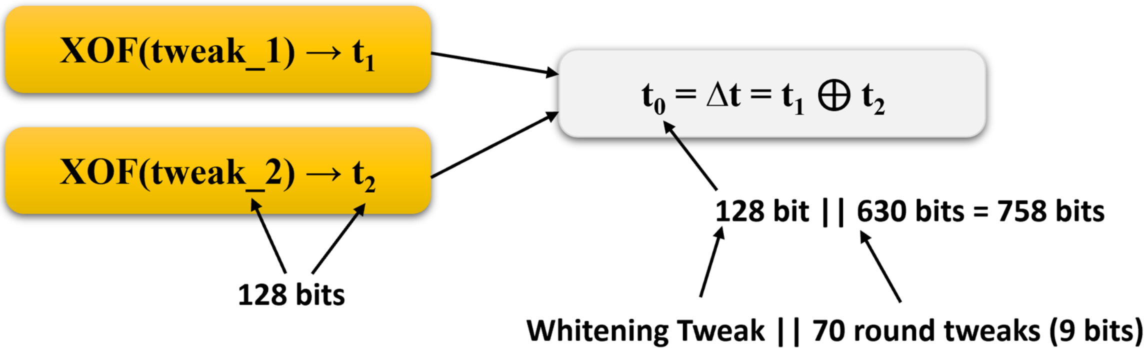


Figure 4.4 Output of XOF

4.3 The Cipher Parameters and Instantiation

Unlike the conventional ciphers, the LowMC instantiation is not fixed and user is at liberty to choose parameters of his own choice to instantiate the Cipher. For the purpose of this thesis, we select parameters as stated in table 4.1 for instantiation.

Table 4.1 Instantiation of LowMC-M

Parameter	**Symbol**	**Size (bit)**	**description**
Key Size	$k \in \{0,1\}^s$	128	the *Key size*
Block Size	$p \in N$	128	The block size is denoted by p = *plaintext*
Size of S-Box	$n \in N$	3	the input size of *S-box*
Number of S-Box	$m \in N$	30	the number of *S-box applied in each rounds*
Rounds	$r \in N$	70	the number of *rounds*
Non-linear size	$s \in N$	90	Size of non-linear Layer is denoted by $s = mn$

It is worth noting that using different combinations of instantiation parameters would result into different security strength owing to varying number of rounds and S-Boxes in the linear layer.

4.4 The Plaintext (*p)*

The block cipher takes a *128-bit* block of data as input, which will be transformed by the block cipher into the cipher text of the same length, i.e. *128-bit*.

4.5 Whitening Key (k_w)

The first step of the cipher is *key whitening* stage or layer. The *whitening* and the *round keys* both are generated by using a *key scheduling algorithm*, that derives these keys from the *master secret key*. We will generate random keys for using with the LowMC-M framework cipher and save them for the encryption and decryption round. All parameters are generated by LowMC-M framework based on the underlying LowMC cipher design.

The Whitening key is a *n x k* matrix which is required to be generated by the *key scheduling algorithm.* However, in our case, this is generated as a random matrix of size *p x k.* The state (*plaintext*) vector is multiplied in GF(2) with the *n x k* whitening key matrix generated earlier. The result product of the matrix multiplication is a *n-bit* vector.

In this layer, the *128-bit* input text block or the state S_1 is multiplied by a *128 x 128 bit* binary matrix and the output of this layer S_2 is a *128-bit* binary vector.

4.6 The Whitening Tweak

In the next layer, the state S_2 is XORed with the 128-bit *Whitening Tweak* value, which is driven from the *Tweak Schedule.* The output of the Whitening Tweak layer is a 128-bit vector S_3.

4.7 The Round Function

The SPN based cipher will have *r* number of rounds (where r $\in 1, 2, 3, \ldots$). We shall consider a round function at round *i*, (where i $\in \{1, 2, 3, \ldots r\}$). The previous round will be referred to as r_{i-1} and next round as r_{i+1}.

Therefore, at the start of round r_i, the state x_i will be output from r_{i-1}. At the start of the first round, output of *Whitening Tweak Layer* i.e. S_3, is available to round function as input.

4.8 The Non-Linear Layer (S_i)

The non-linear or the S-box layer comprises of *m* number of *n-bit* s-boxes that are identical and applied onto the state. Here, the S-boxes are not applied to the complete state, but only to a portion of state. This type of non-linear layer is termed as a partial non-linear layer.

In a classical SP Network based block cipher, the Linear (Li) and Non-Linear (Si) are applied to the complete state during every round. In 2013, Gerard et al [xx] presented the concept of *partial non-linear layers*. The non-linear state (Si) only acts on a part of the state only. Assume that we are using a cipher where in its design it utilizes *m* number of s-boxes in each layer having a block size of *3* bits for each s-box, then the size of non-linear layer $s = 3m$ where $s < p$.

Therefore, if we look at our parameters above, since state p = 128, and a partial non-linear layer has m = 30 s-boxes in each layer with the size of each s-box as 3, therefore the size of partial non-linear layer s = 3m = 30 x 3 = 90. Moreover, clearly $s < p$ since 90 < 128. This means that out of the *128-bits* of the state S_4, only 90 bits will be transformed by the s-boxes (thus the partial non-linear layer) and remaining (n – s) 38-*bits* will pass without any change.

4.9 Handling round keys in rounds with Partial nonlinear layers

In LowMC, we observe that the partial nonlinear layer will act on *s-bits* of the state and remaining *(n-s) bits* will pass through the layer unchanged. This can be used to optimize *key generation*. If we split the round key is into two parts, i.e. $k_i'^{(0)}$ and $k_i'^{(1)}$ such that size of $k_i'^{(0)} = s$ and $k_i'^{(1)} = (n - s)$, then the $k_i'^{(0)}$ part will be XORed with data that has been transformed by s-box and $k_i'^{(1)}$ remains unaffected. Thus, if the *round key* layer follows the *S-Box Layer,* then it can be moved up and combined with the k_{i-1} of the previous layer.

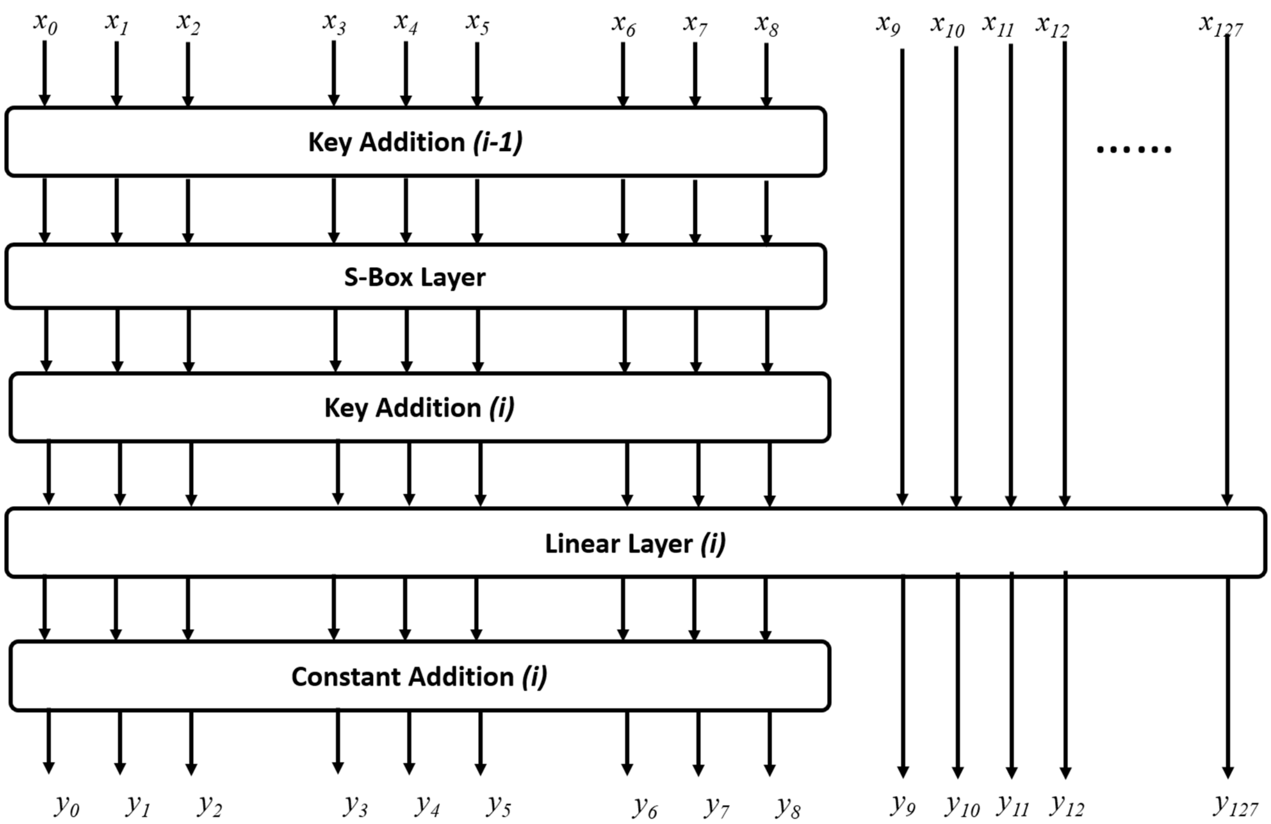


Figure 4.5: *Representation of LowMC with key size equal to S*

4.10 The S-box layer design

The S-box being used in LowMC is a *3-bit* S-box, which is shown in Figure 4.6. An S-box can be realized in terms of a look up table, where the substitutions are carefully designed after evaluating Boolean functions and it is ensured that they satisfy certain security criteria. The S-Box is designed on following Boolean functions.

$$S\ (x_0, x_1, x_2) = (x_0 \oplus x_1 x_2,\ x_0 \oplus x_1 \oplus x_0 x_2,\ x_0 \oplus x_1 \oplus x_2 \oplus x_0 x_1)$$

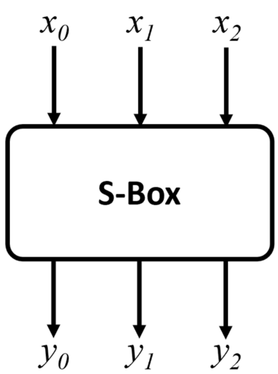


Figure 4.6 The Sbox of the Non-Linear Layer

From the Boolean code above, the lookup table for the S-Box is given below.

Table 4.2: Lookup table for S-Box

Input			Output		
x_0	x_1	x_2	y_0	y_1	Y_2
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	1	1	0
1	0	0	1	1	1
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	0	1	0

From the parameter settings of our cipher, three *3-bit* S-boxes will be act on the first *9* bits (*S2)* in the first round, since we are using the partial non-linear layer. The remaining *n-s* bits will be identity. Therefore, for input x_i ($i \in \{1, 2, 3, \dots n\}$) and corresponding output y_i ($i \in \{1, 2, 3, \dots n\}$), the partial non-linear is shown in Figure 4.7:

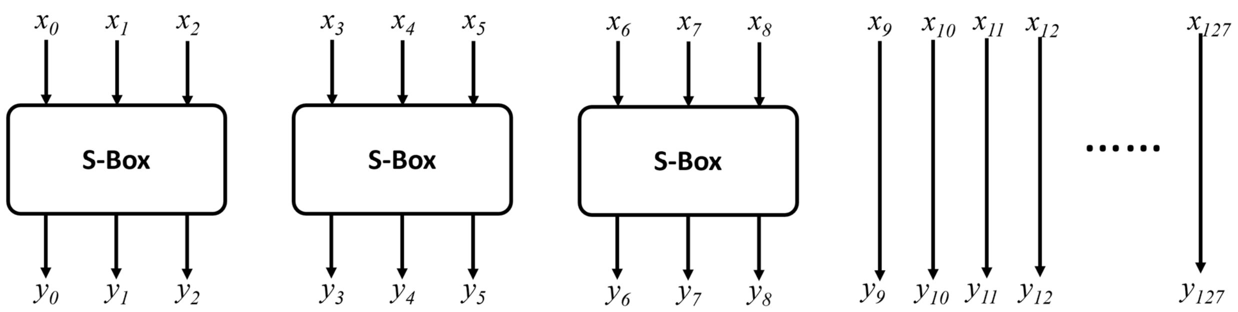


Figure 4.7 The partial non-linear layer

4.11 Round key Addition (k_{ri})

For a *r-round* cipher, (where r $\in 1, 2, 3, \dots$), *r* number of round keys are required to be generated by the *key scheduling algorithm.* As already concluded above, the size of the round key k_i (where i $\in \{1, 2, 3, \dots r\}$), will be equal to *S* which is the nonlinear size of the nonlinear layer. Therefore, each round key is a *s x k* matrix which is required to be generated by the *key scheduling algorithm.* However, in our case, this is generated as a random matrix of size *s x k.*

The state x_i *(n-bit vector)* is split into $x^{(0)}$ and $x^{(1)}$, s.t. $x_i = (x_i^{(0)} \| x_i^{(1)})$ where $x^{(0)}$ is multiplied with the *s x k* round key k_i (where $i \in \{1, 2, 3, \dots r\}$) in GF(2). The resultant product of the matrix multiplication is a *s-bit* vector, which is concatenated with the identity $x_i^{(1)}$ to generate an *n-bit* vector S_5, which is the output of this layer.

4.12 Round Constant Addition (*RC_i*)

In this Layer, the State S_5 is split into $x^{(0)}$ and $x^{(1)}$, s.t. $x_i = (x_i^{(0)} \| x_i^{(1)})$ where $x^{(0)}$ is XORed with a randomly generated S-bit round constant vector. The resultant vector is concatenated with $x_i^{(1)}$ to generate an *n-bit* vector S_6, which is the output of this layer.

4.13 Round Tweak Addition

The state S_6 again is split into $x^{(0)}$ and $x^{(1)}$, s.t. $x_i = (x_i^{(0)} \| x_i^{(1)})$ where $x^{(0)}$ is XORed with a S-bit *Round Tweak*, which is generated by the Tweak Schedule using XOF. The resultant vector is concatenated with $x_i^{(1)}$ to generate an *n-bit* vector S_7, which is the output of this layer.

4.14 Linear Layer (*L_i*)

In the LowMC, the state is multiplied with an invertible randomly selected $n \times n$ binary matrix L_i in the Linear Layer.

However, in LowMC-M the Linear Layer Matrix is not chosen randomly, but is generated to embed differential characteristics in every round, one round after the other. This means that the Linear Layer Matrix is generated separately for every round.

4.15 Differential Characteristics and Differential Cryptanalysis

If we recall, in-order for a Key recovery attack to work, Differential Cryptanalysis is performed to recover the *s-bit* subkey k_r for the last round first, say round *r*. This is done when differential characteristics exist over *r-1* rounds. After recovering the subkey bits, the cipher is reduced to r-1 rounds and the next (previous round) key i.e. k_{r-1} is recovered. For recovering the next round ket K_{r-2}, *r-2* differentials would exist and will be exploited. So far, we have discussed existence of 3 differential characteristics, one for round *r*, *r-1* and *r-2* each.

So, when we intend embedding *a-differential characteristics* over *i-numbers of rounds* of the cipher, we would have to consider the initial *i-1* rounds and design the linear layer matrices accordingly. Note that Linear Layer Matrix L_i of round *I* will not be designed as effect on S-Boxes of last round *i* is not required. However, if we design the cipher requires the differential characteristics to be extended to one more round, then we would be required to design the linear layer matrix of round *i* as well.

4.16 State and Linear Matrix Multiplication - Notation

When carrying out the differential cryptanalysis, the difference during the *i-th* round before it is changed by the Linear Layer is represented by X_i. The Linear Layer Matrix L_i can be partitioned into 4-sub matrices while denoting its *k-th* row by [k,*] and is shown below:-

$$L_i = \left[\begin{array}{c|c} L_i^{00} & L_i^{01} \\ \hline L_i^{10} & L_i^{11} \end{array}\right] \quad \textit{Where} \quad \left[\begin{array}{c|c} L_i^{00} \in GF(2)^{s \times s} & L_i^{01} \in GF(2)^{s \times (n-s)} \\ \hline L_i^{10} \in GF(2)^{(n-s) \times s} & L_i^{11} \in GF(2)^{(n-s) \times (n-s)} \end{array}\right]$$

As discussed earlier, we know that during a round, partial non-linear layer will acts on a fragment of the state, i.e. $x^{(0)}$, and

$$f_i(x) = L_i\,(S_i\,(x^{(0)})||x^{(1)})$$

With this notation,

<u>For Non-linear part</u>

L_i^{00} will correspond to $x^{(0)}$

L_i^{01} will correspond to $x^{(1)}$

<u>For Linear part</u>

L_i^{10} will correspond to $x^{(0)}$

L_i^{11} will correspond to $x^{(1)}$

4.17 Generating the Linear Layer Matrices

As discussed earlier, we will generate the Linear Layer Matrix for use during the first round, after a tweak difference has been pre-computed. Later the Linear Layer matrices are generated round by round along with the differential characteristics. During this process, the differential characteristics will be integrated and will extend from one round to the next round.

4.18 Conclusion

The Generic LowMC-M is a malicious variant of the light weight block Cipher LowMC and has an embedded Backdoor. The LowMC Cipher has been modified into a *Tweakable Block Cipher* to carry the malicious backdoor. The LowMC-M framework also provides a python code which provides insight to the cipher working. The framework does not contain fixed values and generates random parameters every time it is run. By doing this, the designer of the framework have tried to eliminate this doubt that the code carries a backdoor.

The limitations of this framework is that it does not contain a concrete code and can be embedded in any design.

Chapter 5

The Designed Cipher –Performance and Security Analysis

In this chapter we will be discussing the security and performance analysis of our design. As we have already discussed, malicious tweak pair that is used to generate parameters and the differences of the plaintext used is the Backdoor. For anyone else, the cipher acts as a normal AES-like *Tweakable Block Cipher*, which performs encryption and decryption operations. The Code uses partial non-linear layers and can be instantiated using different parameters.

5.1 Performance Analysis of Our Code

The performance analysis of the LowMC was carried out by Peyrin, Thomas, and Haoyang Wang [16]. They used the AVX2 instruction set for Intel Haswell processor and concluded that single encryption usually costs 10,000 to 30,000 cycles. However, this calculation was dependent on the instantiation parameters used to with the cipher along with the block size being used. Our tests were conducted on a laptop with intel Core i7 6700 processor operating at 2.6 GHz. The encryption and decryption operations with parameters selected were comparable to AES as far as running time is concerned. The parameters selected included:-

- n = 128-bits (Block size)
- k = 128 bits (key size)
- s = 90 bits (Non Linear Size)
- XOF = 128
- r = 14 (rounds)

- number of differentials selected = 2

A wide range of parameter combinations can be used for instantiating LowMC within the framework of LowMC-M framework. However, every time the code is instantiated, a unique malicious tweak pair is required to be used to initiate a new embedded differential characteristic.

5.2 Security Analysis of the Code

We have subjected our Code to NIST test suite [27] to test the randomness of binary sequence produced by the cryptographic random number generators used by our code. The tests conducted along-with their output is shown in table 5.1

Table 5.1 – NIST Statistical Test Suite

Test Number	**Nomenclature of Test**	**Description**	**Result**	
			AES	**Our Code**
1.	Frequency (Monobit) - Test	Check proportion of zeros and ones in a sequence and to ascertain whether number of zeros and ones is the same as would be expected in a truly Random Number Generator	**Non Random**	Random
2.	Runs - Test	Ascertain total number of runs (uninterrupted sequence of identical bits) occurring in a sequence	Random	Random
3.	Repeated occurrence Test	Check ratio of 1s in a n-bit block	Random	Random
4.	Binary Matrix Rank Test	the rank of disjoint sub-matrices of the entire sequence are tested. This tests for linear dependence among fixed length substrings of the sequence given for tests.	Random	**Non Random**
5.	Lengthiest	Find the lengthiest occurrence	Random	Random

	sequence of 1s in a Block	(uninterrupted sequence) of 1s within a n-bit Block, that would be required for a qualifying for Random Sequence		
6	Discrete (Spectral) Fourier Transform Test	Detect periodic features (i.e., repeating sequences that occur near each other) in the tested sequence that would indicate a deviation from the assumption of randomness.	Random	Random
7.	Overlapping Template Matching Test	Test the number of occurrences of specified target strings that have already been defined	Random	**Non Random**
8.	Non-overlapping Template Matching Test	Identify generators that produce large occurrences of a given non-periodic (aperiodic) m-bit pattern	Random	Random
9.	Maurer's "Universal Statistical" Test	Detect if the sequence could be considerably compressed without causing any loss of information.	**Non Random**	**Non Random**
10.	Linear Complexity Test	Detect if sequence is complex enough so it can be considered as random	Random	**Non Random**
11.	Approximate Entropy Test	Compare the frequency of occurrence of two consecutive overlapping blocks of adjacent lengths (x and x+1). It is tested against the result that expected for a random sequence	Random	Random
12	Cumulative Sums (Cusum) Test (forward	Checks the cumulative Sum of the partial sequences in the given sequence. It checks if it is too large	**Non Random**	Random

	and reverse)	or too small relative to the expected behavior required for random sequences		
13	Serial Test	Checks the number of the 2m m-bit overlapping patterns, and determines if it is almost the same as would occur in a random sequence.	Random	Random
14	Random Excursions Variant Test	The purpose of this test is to detect if the expected number of visits to various states in the random walk exists or otherwise	Random	Random
15	Random Excursions Test	Checks the number of visits to a particular state within a cycle. It checks if that visits are different from that occurring in a random sequence	Random	Random

The tests where the result of our code differ from that of AES may explored from the prospects of detecting a backdoor.

The screenshot of results displayed by running these tests on AES are shown in Figure 5.1 and on our code are shown in Figure 5.2

```
Type of Test                                      P-Value                Conclusion
01. Frequency Test (Monobit)                      0.004358379639419098   Non-Random
02. Frequency Test within a Block                 0.019618805035413085   Random
03. Run Test                                      0.7757216483704583     Random
04. Longest Run of Ones in a Block                0.028437960302943124   Random
05. Binary Matrix Rank Test                       0.6937201409888837     Random
06. Discrete Fourier Transform (Spectral) Test    1.0                    Random
07. Non-Overlapping Template Matching Test        0.755111809050058      Random
08. Overlapping Template Matching Test            0.020733036101736886   Random
09. Maurer's Universal Statistical test           -1.0                   Non-Random
10. Linear Complexity Test                        0.919688853865076      Random
11. Serial test:
                                                                                              0.6546158277261692     Random
                                                                                              0.4985307552967052     Random
12. Approximate Entropy Test                      0.48868890744866955    Random
13. Cummulative Sums (Forward) Test               0.008716759278838183   Non-Random
14. Cummulative Sums (Reverse) Test               0.006097288223432275   Non-Random
15. Random Excursions Test:
                                State        Chi Squared           P-Value                Conclusion
                                -4           9.428571428571429     0.09314328483293266    Random
                                -3           7.640000000000001     0.17722382548124416    Random
                                -2           4.296296296296297     0.507591586738595      Random
                                -1           3.6666666666666665    0.598332188093073      Random
                                +1           5.666666666666666     0.34001609192154897    Random
                                +2           3.2181069958847734    0.6664008664604748     Random
                                +3           6.4736                0.26282230899245085    Random
                                +4           6.1460502568374284    0.2922659339963548     Random
16. Random Excursions Variant Test:
                                State        COUNTS                P-Value                Conclusion
                                -4.0         1                     0.5853789284609616     Random
                                -3.0         3                     0.6985353583033387     Random
                                -2.0         3                     0.6170750774519738     Random
                                -1.0         2                     0.24821307898992362    Random
                                +1.0         7                     0.7728299926844475     Random
                                +2.0         7                     0.8676323347781927     Random
                                +3.0         11                    0.5186050164287256     Random
                                +4.0         12                    0.5126907602619235     Random
                                +5.0         15                    0.3864762307712327     Random
```

Figure 5.1 Result of NIST Statistical Test Results when run on AES

```
Type of Test                                      P-Value                Conclusion
01. Frequency Test (Monobit)                      0.07709987174354183    Random
02. Frequency Test within a Block                 0.07709987174354183    Random
03. Run Test                                      0.02282616178257479    Random
04. Longest Run of Ones in a Block                0.05116734662089044    Random
05. Binary Matrix Rank Test                       -1.0                   Non-Random
06. Discrete Fourier Transform (Spectral) Test    0.8711314915971565     Random
07. Non-Overlapping Template Matching Test        0.9999999556775546     Random
08. Overlapping Template Matching Test            nan                    Non-Random
09. Maurer's Universal Statistical test           -1.0                   Non-Random
10. Linear Complexity Test                        -1.0                   Non-Random
11. Serial test:
                                                                                              0.4989610874592239     Random
                                                                                              0.49853075529672125    Random
12. Approximate Entropy Test                      1.0                    Random
13. Cummulative Sums (Forward) Test               0.15419957289619884    Random
14. Cummulative Sums (Reverse) Test               0.08411878628402439    Random
15. Random Excursions Test:
                                State        Chi Squared           P-Value                Conclusion
                                -4           16.857142857142858    0.004778851602361653   Non-Random
                                -3           1.8103999999999998    0.8747081612331625     Random
                                -2           2.1604938271604937    0.8265214837628534     Random
                                -1           1.0                   0.9625657732472964     Random
                                +1           8.0                   0.1562356275777222     Random
                                +2           10.814814814814815    0.05517783198207664    Random
                                +3           7.7                   0.17356267022817284    Random
                                +4           0.5714285714285714    0.989273996570703      Random
16. Random Excursions Variant Test:
                                State        COUNTS                P-Value                Conclusion
                                -9.0         2                     0.8638317428547264     Random
                                -8.0         2                     0.8551321405847059     Random
                                -7.0         1                     0.7686247922021466     Random
                                -6.0         1                     0.7491191330005953     Random
                                -5.0         1                     0.7236736098317631     Random
                                -4.0         2                     0.7892680261342813     Random
                                -3.0         6                     0.7518296340458492     Random
                                -2.0         6                     0.6830913983096087     Random
                                -1.0         3                     0.7236736098317631     Random
```

Figure 5.2 Result of NIST Statistical Test Results when run on our Code

5.3 Security Analysis of the Backdoor

5.3.1 Undetectability

An entity should not be able to distinguish between an instance of LowMC-M that does not contain a backdoor from an instance that is embedded with a backdoor. If we recall our code, the instance of LowMC-M that contains a backdoor will have embedded differential characteristics that are generated round by round by specially designed linear layer matrices using the distinct tweak pairs. The Linear Layer matrices, therefore, are the only difference between the two instances.

We have also discussed earlier that while extending the differentials from one round to the next, we create a set of linear equations and try to look for some solution. We have also highlighted in the previous chapter that these parameters are dependent on the tweak pair and the *sub-tweak differences*. Now, in order for the backdoor to be embedded in LowMC-M undetected, a tweak pair used for constructing differential characteristics is not recommended to be used again. The backdoor, therefore, is undetectable, provided the tweak pairs are not reused.

5.3.2 Practicability

As far this property is concerned, once the designer is aware of the backdoor, only a little data and computation effort will be needed to launch a comprehensive key recovery attack. In this case, the backdoor is claimed to practical by its designers.

5.3.3 Untraceability

The user can detect the malicious tweaks while querying using the chosen-tweaks attack model. Since the designer would need to make a few queries before launching an attack, a user can also find out the malicious tweak pair by brute forcing the queries. The backdoor, therefore, is traceable.

5.3.4 Undiscovereability

Undiscoverability is the inability of an attacker to recover the backdoor. However, in this case, the backdoor i.e. the tweak pair, tweak difference, sub-tweak differences and tweak differentials are fully protected by the XOF (SHAKE128). Recovering the tweak pair or the other deterministic tweak differentials should be as difficult as recovering the Key by brute force. The backdoor, therefore, is undiscoverable.

5.3 Conclusion

Our design within the LowMC-M Framework provides a practical and efficient approach to embedding a backdoor in an AES-like *Tweakable Block Cipher*.

Chapter 6

Conclusion

LowMC-M is a Framework for embedding Malicious Backdoor in a Block Cipher. The framework is based on LowMC (Low Multiplicative Complexity) variants of Tweakable Block Ciphers. The designers of this framework proposed a mathematical backdoor and claimed its effectiveness and practicability by embedding deterministic differential characteristics in cipher rounds and recovering the secret key by differential cryptanalysis. However, the limitation with the framework was its practical manifestation. The designers of the framework kept it generic, instantiating it with random values every time the code was run, so that any suspicion of a backdoor could be averted.

In this thesis, we designed an *AES-like tweakable block cipher* based on the LowMC-M framework. The cipher performed encryption and decryption operations successfully and was subjected to *NIST statistical test suite* for testing randomness. The Cipher exhibited behavior similar to AES and the results of the test were found comparable to AES.

Embedding a backdoor in a block cipher is a challenging task when it comes to incorporating a backdoor in its design. On the other hand, protocol level implementation is easy but discoverable. Our Backdoor is a mathematical backdoor which is embedded in the design of the cipher and is based on the LowMC-M framework which allows key recovery using the Differential Cryptanalysis.

References

[1] Pakistan to ban encryption software | Pakistan Defence published online https://defence.pk/pdf/threads/pakistan-to-ban-encryption-software.127633/

[2] Biham, Eli. "New types of cryptanalytic attacks using related keys." Journal of Cryptology 7.4 (1994): 229-246.

[3] Young A., Yung M. (1997) Kleptography: Using Cryptography Against Cryptography. In: Fumy W. (eds) Advances in Cryptology — EUROCRYPT ’97. EUROCRYPT 1997. Lecture Notes in Computer Science, vol 1233. sSpringer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-69053-0_6

[4] Rijmen V., Preneel B. (1997) A family of trapdoor ciphers. In: Biham E. (eds) Fast Software Encryption. FSE 1997. Lecture Notes in Computer Science, vol 1267. Springer, Berlin, Heidelberg. https://doi.org/10.1007/BFb0052342.

[5] Wu, H., Bao, F., Deng, R.H., Ye, Q.Z.: Cryptanalysis of Rijmen-Preneel Trapdoor Ciphers. In: Ohta, K., Pei, D. (eds.) Advances in Cryptology – ASIACRYPT’98. LNCS, vol. 1514, pp. 126–132. Springer, Heidelberg, Germany (1998)

[6] M. Matsui, ”Linear cryptanalysis method for DES cipher”, Advances in Cryptology, Proceedings Eurocrypt’93, LNCS 765, T. Helleseth, Ed., Springer-Verlag, 1994, pp. 386-397.

[7] Paterson, K.G.: Imprimitive Permutation Groups and Trapdoors in Iterated Block Ciphers. In: Knudsen, L.R. (ed.) Fast Software Encryption – FSE’99. LNCS, vol. 1636, pp. 201–214. Springer, Heidelberg, Germany (1999)

[8] Patarin, Jacques, and Louis Goubin. "Asymmetric cryptography with S-Boxes Is it easier than expected to design efficient asymmetric cryptosystems?." International

Conference on Information and Communications Security. Springer, Berlin, Heidelberg, 1997.

[9] Tsutomu Matsumoto, Hideki Imai, Public quadratic polynomial-Tuples for efficient Signature Verification and Message - Encryption, EUCROCRYPT’88, Springer-Verlag, pp. 419-453.

[10] Jacques Patarin, Cryptanalysis of the Matsumoto and Imai public Key Scheme of Eurocrypt’88, CRYPTO’95, Springer-Verlag, pp. 248-261

[11] Liskov, Moses, Ronald L. Rivest, and David Wagner. "Tweakable block ciphers." Annual International Cryptology Conference. Springer, Berlin, Heidelberg, 2002.

[12] Angelova, Vesela, and Yuri Borissov. "Plaintext recovery in des-like cryptosystems based on s-boxes with embedded parity check." Serdica Journal of Computing 7.3 (2013): 257p-270p.

[13] Bannier, Arnaud, and Eric Filiol. "Partition-Based Trapdoor Ciphers." In Partition-Based Trapdoor Ciphers. IntechOpen, 2017.

[14] Bannier, A., Filiol, E.: Mathematical Backdoors in Symmetric Encryption Systems - Proposal for a Backdoored AES-like Block Cipher. arXiv preprint arXiv:1702.06475 (2017)

[15] Arnaud Bannier, Nicolas Bodin, and Eric Filiol. Partition-based trapdoor ciphers. Cryptology ePrint Archive, Report 2016/493, 2016. http://eprint.iacr.org/2016/493.

[16] Peyrin, Thomas, and Haoyang Wang. "The MALICIOUS Framework: Embedding Backdoors into Tweakable Block Ciphers." Annual International Cryptology Conference. Springer, Cham, 2020.

[17] Dan Shumow and Niels Ferguson. On the possibility of a back door in the NIST SP800-90 Dual_Ec_Prng. CRYPTO 2007 Rump Session, August 2007. http://rump2007.cr.yp.to/15-shumow.pdf

[18] https://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security

[19] https://www.nytimes.com/2013/09/06/us/nsa-foils-much-internet-encryption.html?pagewanted=all&_r=0.

[20] https://www.washingtonpost.com/graphics/2020/world/national-security/cia-crypto-encryption-machines-espionage/.

[21] Liskov, Moses, Ronald L. Rivest, and David Wagner. "Tweakable block ciphers." Annual International Cryptology Conference. Springer, Berlin, Heidelberg, 2002.

[22] Wang, Qingju. "Design and Cryptanalysis of Symmetric Key Primitives." (2016).

[23] Sinkov, Abraham, and Todd Feil. Elementary cryptanalysis. Vol. 22. MAA, 2009.

[24] Biham, Eli, and Adi Shamir. "Differential cryptanalysis of DES-like cryptosystems." Journal of CRYPTOLOGY 4.1 (1991): 3-72.

[25] M. Matsui, Linear Cryptanalysis Method for DES Cipher, Abstracts of EUROCRYPT'93, pp. W112–W123, May 1993.

[26] Rechberger, C., Soleimany, H., Tiessen, T.: Cryptanalysis of Low-Data Instances of Full LowMCv2. IACR Transactions on Symmetric Cryptology 2018(3), 163–181 (2018)

[27] Rukhin, Andrew, et al. "A statistical test suite for random and pseudorandom number generators for cryptographic applications," NIST Special Publication 800-22 (revised May 15." (2002).

Appendix

Main.py

```
import numpy as np
from pyfinite import ffield
import BitVector


def sub3(num):
    ###### Substitution. Kindly Check Image Attached
    if num == 0:
        return np.array([1,1,1])
    elif num == 1:
        return np.array([1,0,0])
    elif num == 2:
        return np.array([0,1,1])
    elif num == 3:
        return np.array([0,1,0])
    elif num == 4:
        return np.array([0,0,0])
    elif num == 5:
        return np.array([0,0,1])
    elif num == 6:
        return np.array([1,0,1])
    else:
        return np.array([1,1,0])

def sBox(arr):
    ####### Checks and converts Inserted Array to 16 Bits


    ####### Slices the Least Significant 9 bits for 3 bit Byte
Substitution

# FAWAD, this needs to be changed
    l1 = arr[0:3]
    l2 = arr[3:6]
    l3 = arr[6:9]
    rem = arr[9:]
    ####### Converts to Decimal
    l1 = int("".join(str(x) for x in l1), 2)
    l2 = int("".join(str(x) for x in l2), 2)
    l3 = int("".join(str(x) for x in l3), 2)
    ####### Calls Sub 3 method in Line 4. Returns 3 bit numpy Array
    l1 = sub3(l1)
    l2 = sub3(l2)
    l3 = sub3(l3)
    ####### Appends and substituted bits and unchanged 7 bits
    rem = np.append([l1,l2,l3],rem)
    return rem


def ECipher(pt, key, rounds, blocksize, LMatrices, roundkey_matrices,
constants, tweakdifferences, nonLsize):
```

```
#print('PT = :')
#print(pt)
###### Retrieves Whitening Data From Data Class
wk = np.array(roundkey_matrices[0])
###### Matrix Multiplication of Whitening Key and Key
wk = np.mod(np.dot(wk,key),2)
#print('WK = :')
#print(wk)
###### Retrieves tweak diff From Data Class
wt = np.array(tweakdifferences[0][0])

#pt = pt^wt^wk

wk = int("".join(str(x) for x in wk), 2)
wt = int("".join(str(x) for x in wt), 2)
pt = int("".join(str(x) for x in pt), 2)

pt = pt^wt^wk
pt = np.array([int(x) for x in bin(pt)[2:].zfill(blocksize)])

for i in range(rounds):

    ####### SBox Subsitution. Go to Line 22
    #print(pt)
    pt = sBox(pt)
    #print(pt)
    #pt = np.array([int(x) for x in bin(pt)[2:].zfill(32)])

    ####### Retrieves Data Per Round
    LLM = np.array(LMatrices[i])
    RK = np.array(roundkey_matrices[i+1])
    RC = np.array(constants[i])
    RT = np.array(tweakdifferences[0][i+1])
    #print(LLM)
    #print(RK)

    ####### LLM Matrix Multiplication with Altered Plain Text
    # FADDI
    pt = np.mod(np.dot(LLM,pt),2)


     ####### Converting 9x16 Round Key to 9x1 Round Key with Key
     Matrix Mul
    RK = np.mod(np.dot(RK,key),2)
    #print(pt)
    #print(RT)
    #print(RC)
    #print('Round Key = :')
    #print(RK)
    #Aux = np.bitwise_xor(Aux,RT)

    apt = pt[0:9]
    rem = pt[9:]
    xr =  RK ^ RC ^ RT
    apt = int("".join(str(x) for x in apt), 2)
    xr = int("".join(str(x) for x in xr), 2)
```

```
    apt = apt ^ xr
    apt = np.array([int(x) for x in bin(apt)[2:].zfill(nonLsize)])
    pt = np.append(apt,rem)


return pt

def invsub3(num):
###### Substitution. Kindly Check Image Attached
if num == 0:
return np.array([1,0,0])
elif num == 1:
return np.array([1,0,1])
elif num == 2:
return np.array([0,1,1])
elif num == 3:
return np.array([0,1,0])
elif num == 4:
return np.array([0,0,1])
elif num == 5:
return np.array([1,1,0])
elif num == 6:
return np.array([1,1,1])
else:
return np.array([0,0,0])

def invsBox(arr):
####### Slices the Least Significant 9 bits for 3 bit Byte
Substitution
l1 = arr[0:3]
l2 = arr[3:6]
l3 = arr[6:9]
rem = arr[9:]

####### Converts to Decimal
l1 = int("".join(str(x) for x in l1), 2)
l2 = int("".join(str(x) for x in l2), 2)
l3 = int("".join(str(x) for x in l3), 2)
####### Calls Inverse Sub 3 method in Line 4. Returns 3 bit numpy
Array
l1 = invsub3(l1)
l2 = invsub3(l2)
l3 = invsub3(l3)
####### Appends and substituted bits and unchanged 7 bits
rem = np.append([l1,l2,l3],rem)
return rem
def __invert_lin_matrix(LLM, blocksize):
mat = LLM
inv_mat = []
for i in range(blocksize):
temp_bv = BitVector.BitVector(intVal=0, size=blocksize)
temp_bv[i] = 1
inv_mat.append(temp_bv)
# Transform to upper triangular matrix
row = 0
for col in range(blocksize):
if (not mat[row][col]):
```

```
    r = row + 1
    while ((r < blocksize) and (not mat[r][col])):
    r += 1
    if (r >= blocksize):
     continue
     else:
     temp = mat[row]
                mat[row] = mat[r]
                mat[r] = temp
                temp = inv_mat[row]
                inv_mat[row] = inv_mat[r]
                inv_mat[r] = temp
        for i in range(row + 1, blocksize):
            if (mat[i][col]):
                mat[i] = mat[i] ^ mat[row]
                inv_mat[i] = inv_mat[i] ^ inv_mat[row]
        row += 1

            # Transform to inverse matrix
    for col in range(blocksize, 0, -1):
        for r in range(col - 1):
            if (mat[r][col - 1]):
                mat[r] = mat[r] ^ mat[col - 1]
                inv_mat[r] = inv_mat[r] ^ inv_mat[col - 1]

    return inv_mat


def DCipher(pt, key, rounds, blocksize, LMatrices, roundkey_matrices,
constants, tweakdifferences, nonLsize):
    gf = ffield.FField(rounds)
    #key = np.array([int(x) for x in bin(key)[2:]])
    #key = key.transpose()
    for i in range(rounds, 0,-1):


        RK = roundkey_matrices[i]
        RC = constants[i-1]
        RT = tweakdifferences[0][i]
        LLM = np.array(LMatrices[i-1])                          #
Compute its inverse (Invertable matrix Remaining)
        RK = np.mod(np.dot(RK,key),2)

        apt = pt[0:9]
        rem = pt[9:]
        xr =  RK ^ RC ^ RT
        apt = int("".join(str(x) for x in apt), 2)
        xr = int("".join(str(x) for x in xr), 2)
        apt = apt ^ xr
        apt = np.array([int(x) for x in bin(apt)[2:].zfill(nonLsize)])
# fills values to the left as desired, can be cross checked as well
        pt = np.append(apt,rem)


        bv = []
        #LLM = np.linalg.inv(LLM)
        #LLM = np.mod(LLM,2)
```

```
        #LLM = LLM.astype(int)

        # FADDI

        #LLM = inverseMatrix(LLM, blocksize)
        #print(LLM)

        for vec in range(blocksize):
            bv.append(BitVector.BitVector(bitlist = LLM[vec].tolist()))
        LLM = __invert_lin_matrix(bv,blocksize)
        pt = np.mod(np.dot(LLM,pt),2)


        ####### Inverse S Box
        pt = invsBox(pt)

    wk = np.array(roundkey_matrices[0])
    wk = np.mod(np.dot(wk,key),2)
    wt = np.array(tweakdifferences[0][0])


    wk = int("".join(str(x) for x in wk), 2)
    wt = int("".join(str(x) for x in wt), 2)
    pt = int("".join(str(x) for x in pt), 2)

    pt = pt^wt^wk
    pt = np.array([int(x) for x in bin(pt)[2:].zfill(blocksize)])

    return pt
```

lowmc_mc.py

```
'''

This program generates an instance of LowMC-M. Since SHAKE128 is considered, so the key size is fixed to 128 bits for security concern.

'''

from sage.all import *

from SHAKE128 import *

from Main import *

import numpy as np    #numpy is a library

import pickle


blocksize = 16

keysize = 16

tweaksize = 16

sboxsize = 3               # sbox size
```

```
m = 3                      # number of sboxes

nonLsize = sboxsize*m      # non-linear size

rounds = 14                # number of rounds

num_dc = 1                # number of differential characteristics to be embedded


def generate_Kmatrix():

    roundkey_matrices = [] #has been defined empty

    #Generate the whitening key

    while True:       # loop till the time it is true

        mat = np.random.randint(0,2,size = (blocksize,keysize))

# Generates a random number 01001 matrix ... size  (128, 128)

        Mat = matrix(GF(2),mat)                       #

        if rank(Mat) == min(blocksize,keysize):

# what does this mean

            break     # break loop if condition is met

    roundkey_matrices.append(mat.tolist())

    np.shape(round)


#Generate the round keys

    for r in range(rounds):

        while True:

            mat = np.random.randint(0,2,size = (nonLsize,keysize))     # size (9, 128)  Generates it for 70 rounds

            Mat = matrix(GF(2),mat)

            if rank(Mat) == min(nonLsize,keysize):

                break

        roundkey_matrices.append(mat.tolist())

    return roundkey_matrices


def generate_constants():

    cons = []

    for r in range(rounds):

        con = np.random.randint(0,2,size = nonLsize)   # Round constant will only have one row of size = nonLsize
```

```
        cons.append(con.tolist())
    return cons


def generate_tweakdifferences():
    subtweakdiff_set = []
    tweak_set = []
    for i in range(num_dc):
        subtweaks1 = [0] * (rounds+1) # generate row of 71 x 0 = [0 0 0
0 ....]
        subtweaks2 = [0] * (rounds+1)


#*************TWEAK GENERATION**************
# It can be chosen by the user alternatively, both size and value


        tweak1 = list(np.random.randint(0,2,size=tweaksize))
# one row of size 128
        tweak2 = list(np.random.randint(0,2,size=tweaksize))
#*******************************************
        tstring1 = shake128(tweak1, blocksize+rounds*nonLsize)
# one row of size 758
        tstring2 = shake128(tweak2, blocksize+rounds*nonLsize)
        subtweaks1[0] = tstring1[:blocksize]
# one row of size 128 extracted from above 758
        subtweaks2[0] = tstring2[:blocksize]
        for r in range(rounds):
            subtweaks1[r+1] =
tstring1[blocksize+r*nonLsize:blocksize+(r+1)*nonLsize]
# for every round, 9 next bits are extracted from above
            subtweaks2[r+1] =
tstring2[blocksize+r*nonLsize:blocksize+(r+1)*nonLsize]

# 758 - 128 - (70*9) = 0
        subtweak_differences = []
        for r in range(rounds+1):
            subtweak_differences.append([subtweaks1[r][j] ^
subtweaks2[r][j] for j in range(len(subtweaks1[r]))]) # bitwise XoR
```

```
# Subtweaks XoR for 70 rounds not including 128 bit first key
         subtweakdiff_set.append(subtweak_differences)
        tweak_set.append([tweak1,tweak2])
    return subtweakdiff_set, tweak_set


def generate_Lmatrix(differences, tweakdiff, r):   # Function called in line 148 and 152
    Length = len(differences)
    Nonzero = [0]*sboxsize


    # This is to ensure that an i-round deterministic differential characteristic will active all the Sboxes in round i+1
    if r >= (rounds-1-num_dc):
        for i in range(m):
            while True:
                Nonzero[i] = np.random.randint(0,2,size=m)
                if sum([Nonzero[i][j]^tweakdiff[Length-1][r+1][j+m*i] for j in range(sboxsize)]) != 0:
                    break


    Set = []
    for t in range(nonLsize):
        extra_column = []
        for i in range(Length):
            extra_column.append([tweakdiff[i][r+1][t]])


        if r >= (rounds-1-num_dc):
            extra_column[-1][0] = Nonzero[t//m][t%m]


        augmented_mat = (np.append(differences, extra_column, axis=1)).tolist()
        Mat = matrix(GF(2),augmented_mat)
        Set.append(Mat.right_kernel().basis_matrix())


    while True:
```

```
        Matrice = []
        # Generate the first (nonLsize) rows
        for t in range(nonLsize):
            while True:
                tmpvec = random_vector(GF(2),len(list(Set[t])))
                if (tmpvec*Set[t])[-1] == 1:
                    Matrice.append(list((tmpvec*Set[t])[:-1]))
                    break
     # Generate the left (blocksize-nonLsize) rows
      # Last non-linear rows after 9th row
        for i in range(blocksize-m*sboxsize):
            Matrice.append(list(np.random.randint(0,2,size=blocksize)))
        mat = matrix(GF(2),Matrice)
        if rank(mat) == blocksize:
            return Matrice


def generate_DC():
    roundkey_matrices = generate_Kmatrix()  #Generate key matrices
    constants = generate_constants()    #Generate round constants


    tweakdifferences, tweak_set = generate_tweakdifferences()   #Generate
tweak pairs and its corresponding sub-tweak differences
    BS_differences = [[] for _ in range(rounds)] # Difference before Sbox
transformation in each round
    AM_differences = [[] for _ in range(rounds)] # Difference after
matrix multiplication in each round
    LMatrices = []  # Linear matrices
    plaintext_differences = []  # The plaintext difference is input
difference of the differential characteristic to be embedded, it can be
chosen by the user along with the first sub-tweak difference.


    for i in range(num_dc): # Generate plaintext difference
        plaintext_differences.append(tweakdifferences[i][0][:nonLsize] +
list(np.random.randint(0,2,size=blocksize-nonLsize)))
            # For 14 number of differentials, it generates PT difference
as: tweakdifferences from 0 to 9 + list (119 values)
```

```
    for i in range(num_dc): # Compute the difference between the
plaintext difference and the first sub-tweak difference

        BS_differences[0].append([plaintext_differences[i][j] ^
tweakdifferences[i][0][j] for j in range(blocksize)])

            # Fawad xor of plaintext_differences and tweakdifferecnes -->
Ctrl + F tweakdifferences


    #*************GENERATE ROUND DIFFERENCE**************

    # Building (num_dc) differential characteristics, the number of
rounds ranges from (rounds-1) to (rounds-1-num_dc+1)


    # For round 69 to round 55 - 56 seen from file generated.

    #****************************************************


    for r in range(rounds-1):                  # 1 to 69

        if r <= (rounds-1-num_dc):             # if r <= 55

LMatrices.append(generate_Lmatrix(BS_differences[r],tweakdifferences,r))
# Generate linear matrix

        current_num_dc = num_dc

        elif r > (rounds-1-num_dc):

            LMatrices.append(generate_Lmatrix(BS_differences[r][:-
1],tweakdifferences,r))   # Generate linear matrix

 current_num_dc = rounds-r-1

 for i in range(current_num_dc):        # For remaining I guess

 AM_differences[r].append(list(matrix(GF(2),LMatrices[r]) *
vector(GF(2),BS_differences[r][i])))

BS_differences[r+1].append([AM_differences[r][i][j] +
tweakdifferences[i][r+1][j] for j in range(nonLsize)] + \
AM_differences[r][i][nonLsize:])


    # Generate the last linear matrix

    while True:

        mat = np.random.randint(0,2,size = (blocksize,blocksize))

        Mat = matrix(GF(2),mat)

        if rank(Mat) == blocksize:

            break
```

```
LMatrices.append(mat.tolist())
with open('matrices_and_constants.txt', 'w') as matfile:
    s = 'Linear layer matrices\n\n'
    for r in range(rounds):
        s += '\nround ' + str(r) + ':\n'
        for row in LMatrices[r]:
            s += str(row) + '\n'

    s += '\nKey matrices\n\n'
    for r in range(rounds+1):
        s += 'round ' + str(r) + ":\n"
        for row in roundkey_matrices[r]:
            s += str(row) + "\n"
    s += '\nRound constants\n\n'
    for r in range(rounds):
        s += str(constants[r]) + '\n'
    s += '\nRound-Tweaks\n'
    s += str(tweakdifferences) + '\n'

    matfile.write(s)
with open('Differential Characteristics.txt','w') as dcfile:
    s = 'Differential Characteristics\n\n\n'
    for i in range(num_dc):
        s += '\ndifferential ' + str(i+1) + ':\n'
        s += 'length: {} rounds\n'.format(rounds-i-1)
        s += 'tweak pair:\n'
        s += str(tweak_set[i][0]) + '\n'
        s += str(tweak_set[i][1]) + '\n'
        s += 'plaintext difference:\n'
        s += str(plaintext_differences[i]) + '\n'
        s +=  'differences before SB:\n'
        for r in range(rounds-i):
                  s += 'round {:3} '.format(r+1) +
 str(BS_differences[r][i]) + '\n'
```

```
        dcfile.write(s)

     return LMatrices, roundkey_matrices, constants, tweakdifferences

def main():
```

#If you would like to generate new values then UNCOMMENT the Creqation BLock of Code. Otherwise this will work on the same values.

```
#Creation Block START

#    PT = np.random.randint(0,2,size=blocksize)

#    Key = np.random.randint(0,2,size=blocksize)

#    print('PT :')

#    print(PT)

#    LMatrices, roundkey_matrices, constants, tweakdifferences =
generate_DC()

#    with open('items.pkl','wb') as f:

#        pickle.dump([rounds, blocksize, LMatrices, roundkey_matrices,
constants, tweakdifferences, nonLsize],f)

#    f.close()

    #Creation Block END

    analysis()


if __name__ == "__main__":

    main()
```

generate.py

```
#def main():

 #   for i in range(100):
        #PT = np.array([0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1])
        #Key =np.array([1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1])
  #     PT = np.random.randint(0,2,size=blocksize)
  #      Key = np.random.randint(0,2,size=blocksize)
        #print('PT :')
        #print(PT)
    #    LMatrices, roundkey_matrices, constants, tweakdifferences =
generate_DC()
     #   CT= ECipher(PT,Key, rounds, blocksize, LMatrices,
roundkey_matrices, constants, tweakdifferences, nonLsize)
        #print('CT')
      #  print(CT)
        #res= DCipher(CT,Key, rounds, blocksize, LMatrices,
roundkey_matrices, constants, tweakdifferences, nonLsize)
        #print('PT')
        #print(res)
if __name__ == "__main__":
    main()
```