Advanced Cryptographic System:
Design, Architecture and FPGA Implementation

Mohammed Falih Al-Gailani

Submitted in accordance with the requirements for the degree of

Doctor of Philosophy

Newcastle University

School of Electrical, Electronic and Computer Engineering

England, United Kingdom

May 2012

Abstract

The field programmable gate array (FPGA) is a powerful technology, and since its introduction broad prospects have opened up for engineers to creatively design and implement complete systems in various fields. One such area that has a long history in information and network security is cryptography, which is considered in this thesis. The challenge for engineers is to design secure cryptographic systems, which should work efficiently on different platforms with the levels of security required. In addition, cryptographic functionalities have to be implemented with acceptable degrees of complexity while demanding lower power consumption.

The present work is devoted to the design of an efficient block cipher that meets contemporary security requirements, and to implement the proposed design in a configurable hardware platform. The cipher has been designed according to Shannon's principles and modern cryptographic theorems. It is an iterated symmetric-key block cipher based on the substitution permutation network and number theoretic transform with variable key length, block size and word length. These parameters can be undisclosed when determined by the system, making cryptanalysis almost impossible. The aim is to design a more secure, reliable and flexible system that can run as a ratified standard, with reasonable computational complexity for a sufficient service time.

Analyses are carried out on the transforms concerned, which belong to the number theoretic transforms family, to evaluate their diffusion power, and the results confirm good performance in this respect mostly of a minimum of 50%. The new Mersenne number transform and Fermat number transform were included in the design because their characteristics meet the basic requirements of modern cryptographic systems.

A new 7×7 substitution box (S-box) is designed and its non-linear properties are evaluated, resulting in values of 2^{-6} for maximum difference propagation probability and $2^{-2.678}$ for maximum input-output correlation. In addition, these parameters are calculated for all S-boxes belonging to the previous and current standard algorithms. Moreover, three extra S-boxes are derived from the new S-box and another three from the current standard, preserving the same non-linear properties by reordering the output elements.

The robustness of the proposed cipher in terms of differential and linear cryptanalysis is then considered, and it is proven that the algorithm is secure against such well-known attacks from round three onwards regardless of block or key length.

A number of test vectors are run to verify the correctness of the algorithm's implementation in terms of any possible error, and all results were promising. Tests included the known answer test, the multi-block message test, and the Monte Carlo test.

Finally, efficient hardware architectures for the proposed cipher have been designed and implemented using the FPGA system generator platform. The implementations are run on the target device, Xilinx Virtex 6 (XC6VLX130T-2FF484). Using parallel loop-unrolling architecture, a high throughput of 44.9 Gbits/sec is achieved with a power consumption of 1.83W and 8030 slices for implementing the encryption module with key and block lengths of 16×7 bits. There are a variety of outcomes when the cipher is implemented on different FPGA devices as well as for different block and key lengths.

This thesis is dedicated to the memory of my beloved father,

who died during the final stage of my research.

To him I owe this thesis,

and without his support and encouragement

this study would never have been possible

Acknowledgements

My first and foremost thanks go to Allah for his blessings at every stage of my life.

I would like to express my deep appreciation and gratitude to my supervisor Professor S. Boussakta for his guidance, support and advice throughout the research project.

I would also like to thank my second supervisor Mr. Jeff Neasham for his assistance.

I am immensely thankful and appreciative to Professor B. Sharif, the Head of the School of Electrical, Electronic and Computer Engineering, for his support.

It is also my pleasure to thank all members of the School of Electrical, Electronic and Computer Engineering, Newcastle University, both academic and non-academic, who helped and supported me during my research.

I would like to give special thanks to the Ministry of Higher Education and Scientific Research in Iraq for their financial support.

Finally, and most importantly, I would like to express my deep appreciation to my parents, my wife, my family and my friends for their continuous support and endless patience, and for always encouraging me throughout these years. I am really grateful for everything they have done for me.

Declaration

I hereby declare that the work contained in this thesis has not been previously submitted, in whole or in part, for the award of any other academic degree or diploma. I also declare that, except where otherwise indicated, all material in this thesis is entirely my own work and I have fully cited and referenced all information and results that are not original to this work.

Table of Contents

List of Figures

List of Tables

List of Abbreviations and symbols

$\langle A \rangle_B$	A modulo B
$\oplus$	Bitwise XOR operation
$\boxplus$	Modular addition
$\odot$	Modular multiplication
1-D	One dimension
Ac	Column indices for addition array
ADT	Actual data transfer
AES	Advanced Encryption Standard
A_N	Number of addition operations
ANSI	American National Standards Institute
Ar	Row indices for addition array
BIC	Bit Independence Criterion
BitGen	Bitstream generation program
BN	Branch Number
CBC	Cipher Block Chaining mode
CFB	Cipher Feedback mode
CLB	Configurable Logic Block
CMOS	Complementary Metal Oxide Semiconductor
CMT	Clock Management Tile
CT	Ciphertext
CTR	Counter mode
D	Decryption
DC	Differential Cryptanalysis
DCM	Xilinx Digital Clock Manager
DES	Data Encryption Standard
DFT	Discrete Fourier Transform
DIF	Decimation In Frequency
DIT	Decimation In Time
D_{max}	Maximum value in the XOR distribution table
DPP_{max}	Maximum Difference Propagation Probability
DSP	Digital Signal Processing
E	Encryption
E^2PROM	Electrically Erasable Programmable Read-Only Memory
ECB	Electronic Codebook mode
ECC	Elliptic Curve Cryptography
EDK	Embedded Development Kit
FIFO	First-In First-Out
FIPS	Federal Information Processing Standard
FNT	Fermat Number Transform
FPGA	Field Programmable Gate Array
F_t	Fermat number
GF(•)	Galois Field
h	Hexadecimal number
HDL	Hardware Description Language
I/O	Input/Output

ib	Number of required input bytes per block
IDE	Integrated Development Environment
IDEA	International Data Encryption Algorithm
Im(•)	Imaginary part
Inv-NMNT	Inverse new Mersenne number transform
Inv-ST	Inverse shifts transform
IOC_{max}	Maximum Input-Output Correlation
IP	Initial Permutation
IP^{-1}	Inverse of the Initial Permutation
ISE	Xilinx Integrated Software Environment
ISP	In-System Programmable
IUT	Implementations Under Test
IV	Initial Value
JTAG	Joint Test Action Group
Ka	Accumulated Key
KAT	Known Answer Test
Kc	Cipher Key
Kr	Round Key
LC	Linear Cryptanalysis
LCs	Logic Cells
LFSR	Linear Feedback Shift Register
L_{max}	Maximum value in the linear approximation table
LSB	Least Significant Bit
LUT	Look-Up Table
MAC	Message Authentication Code
MCT	Monte Carlo Test
MDS	Maximum Distance Separable
ml	Number of elements in the message
MMT	Multi-block Message Test
M_N	Number of multiplication operations
M_p	Mersenne number
MSB	Most Significant Bit
MUX	Multiplexers
N	Transform Length
nb	Block size in bits
NBS	National Bureau of Standards
NCD	Native Circuit Description
NGD	Xilinx Native Generic Database file
NIST	National Institute of Standards and Technology
nM	Message length in bits
N_{max}	Maximum transform length
NMNT	New Mersenne Number Transform
nNz	Number of non-zero elements in the first column of the XOR distribution table
NTT	Number Theoretic Transform
nZ	Number of padded bytes
OFB	Output Feedback mode
OTP	One-Time Programmable
PB_{max}	Maximum probability bias
P-box	Permutation box
Pc	Column indices for permutation array
PC	Permuted choice

PES	Proposed Encryption Standard algorithm
PGP	Pretty Good Privacy
PHT	Pseudo-Hadamard Transform
Pr	Row indices for permutation array
P_r	Probability
PROM	Programmable Read-Only Memory
PT	Plaintext
R	Maximum number of row in the array
RAM	Random Access Memory
Rcon	Round constant
rd	Maximum number of rounds
Re(•)	Real part
ROM	Read Only Memory
R_s	Robustness of the S-box
RSA	Public key algorithm with its inventor's initials
RTL	Register Transfer Level
s	Segment
S(•)	Element substitution
SAC	Strict Avalanche Criterion
S-box	Substitution box
SI(•)	Inverse element substitution
SoC	System-on-Chip
SOPC	System on a Programmable Chip
SPN	Substitution Permutation Network
SR	Shift Register
SRAM	Static Random-Access Memory
ST	Shifts transform
T	Transitional Transform
TDEA	Triple Data Encryption Algorithm
TPA	Throughput Per Area
UCF	User Constraints File
VA	Variable addition
VHDL	V: VHSIC, Very High Speed Integrated Circuit; HDL, Hardware Description Language
VS	Variable subtraction
W(•)	Element's Weight
XST	Xilinx Synthesis Technology
Zp	Percentage of the number of zero elements relative to the total
α	Kernel of 1-D Fermat number transform
β	Kernel of 1-D new Mersenne number transform
ϵ	Bias

List of Publications

1. M. F. Al-Gailani and S. Boussakta, "Evaluation of One-dimensional NMNT for Security Applications," in *7th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)* England, UK, 2010, pp. 715-720.
2. X. B. Yang, *et al.*, "A New Development of Cryptosystem Using New Mersenne Number Transform," in *7th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*, England, UK, 2010, pp. 701-705.
3. M. F. Al-Gailani, *et al.*, "Fermat Number Transform Diffusion's Analysis," in *IEEE GCC Conference and Exhibition*, Dubai, UAE, 2011, pp. 237-240.
4. M. F. Al-Gailani and S. Boussakta, "New Mersenne Number Transform Diffusion Power Analysis," in *American Journal of Engineering and Applied Sciences*, 2011, 4(4), pp. 461-469.

Chapter 1

Introduction

This chapter introduces the work described in this thesis. It is organised as follows: section 1.1 briefly introduces the background of the subject. The motivation for the research is explained in section 1.2, and its aim and objective are stated in section 1.3 while its contributions to knowledge are detailed in section 1.4. Finally an outline of the thesis is presented in section 1.5.

1.1 Background

The breakthrough of the field programmable gate array (FPGA) made by Xilinx in 1984, especially given the tremendous developments in silicon technology, has prompted engineers to make innovations in design, architecture and hardware implementation in many different fields such as communication devices, digital signal processing (DSP), cryptography and the system-on-chip (SoC).

Data security is of great interest in relation to most applications, especially those concerning personal and financial issues. It has been a hot topic for a long time, particularly in today's societies living in a world of ubiquitous computing. Computers are everywhere, millions and millions of items of data are transferred every second using many different applications. The confidentiality associated with a large number of

these applications needs to be protected. Some applications, for instance, require a disclosure of credit card information to a website for online transactions such as in internet shopping, ATM operations and bank transfers.

Cryptography is a tool that has been extensively used for many years [1], although at its inception this was confined to diplomatic, military and intelligence services, cryptography can be used nowadays by everyone to protect the confidentiality of their data transmitted over insecure channels as well as to provide authenticity and data integrity.

However, due to significant developments in cryptanalytic techniques and in processing power, existing algorithms used in cryptography are increasingly susceptible to attacks, such as side-channel attacks, for example. Therefore, as a response, new methods are continuously emerging in order to achieve the desired level of requirements, regarding for example: security, speed, platform, complexity, and power consumption.

1.2 Motivation for the Research

The previous standard algorithm, the Data Encryption Standard (DES) [2], was once very secure. However, due to developments in processing power and parallel processing technologies, this algorithm became quite vulnerable to exhaustive key search attacks, as the key length of the algorithm was considered to be short. Since the algorithm was designed for a fixed block size and key length, an alternative algorithm was essential. As a result, the National Institute of Standards and Technology (NIST) proposed in 1997 the Advanced Encryption Standard (AES) to replace the DES. In the current standard AES algorithm [3], the block size is double that of the DES of 128-bit, and the key length has expanded from 56 to 128-bit and could even support 192 and 256-bit. Even though this algorithm is currently believed to be secure, risks of vulnerability still exist due to the continual development in the field of cryptanalysis as well as increasing networking and processing power. Sooner or later, this algorithm will be broken, since it was designed for a fixed block size and limited key lengths, in my opinion the same action that has been taken before with the previous standard algorithm, an announcement for a replacement algorithm, is expected to be taken with this standard should the block size or key lengths become insufficient to suit the security requirements. Consequently, a reasonable solution would be to design an algorithm in such a way that it works efficiently on different block sizes and key lengths while

adhering to the required level of security. In other words it would be a revision-free algorithm, ensuring practical usage for the proposed lifespan. In addition, the suitability of the algorithm for implementing on different platforms should be taken into consideration within the design.

1.3 Aim and Objectives

The aim of the present work is to design an efficient block cipher that meets security requirements and then propose an architecture for the investigated cipher to be implemented in a configurable hardware platform. The design of the algorithm is based on the following criteria: simplicity, in order facilitate ease of analysis and implementation; immunity, so as be resilient to known attacks; flexibility, in accepting variable key lengths, block lengths and word lengths; and efficiency, so as to be implemented on a wide range of platforms with reasonable complexity and speed.

1.4 Contributions

The major contributions of the research work can be summarised as follows:

- The new Mersenne number transform (NMNT) and the Fermat number transform (FNT), as well as the mix column transform used in the current state-of-the-art AES algorithm are analysed and their diffusion power are determined based on the branch number technique. The diffusion power of the NMNT and FNT are further investigated to diagnose every possible case, especially those vulnerable ones to know how to avoid them in the design. These analyses are conducted using proposed techniques based on probabilities.
- All S-boxes of the previous standard DES as well as the current standard AES are analysed. Accordingly, their non-linear properties represented by maximum difference propagation probability (DPP_{max}), maximum input-output correlation (IOC_{max}) and robustness are computed after building up XOR distribution and linear approximation tables. Thereafter, a new S-box is generated by following the same procedures used in the construction of the AES S-box. In addition, other S-boxes are derived from the AES S-box and from the new S-box which have the same non-linear properties but where the order of the output elements with different offsets is changed.
- A new parameterised symmetric-key block cipher is proposed based on the substitution permutation network (SPN) and number theoretic transform (NTT).

The cipher is designed in such a way that it optionally can calculate the word length as well as the appropriate block size this is equivalent to the key length efficiently depending on the processor register length and message length. By undisclosed such parameters, cryptanalysis becomes almost impossible. In addition, a solution is proposed for the algorithm based on the NMNT to solve a problem that might arises due to the modulus value being a power of two minus one, where the zero value and the M_p value are retrieved as a zero. In addition, a simple action is undertaken in the substitution boxes (S-boxes) to overcome such a possible problem.

- Three categories of test vectors are run to verify the correctness of the algorithm's implementation in terms of any possible error. These tests include the known answer test (KAT), the multi-block message test (MMT), and the Monte Carlo test (MCT).
- The resistance of the suggested cipher towards differential and linear cryptanalysis is considered.
- Efficient hardware architectures for the proposed cipher are designed and implemented using FPGA technology.

1.5 Outline of the Thesis

This thesis is devoted to designing and implementing a secure cryptographic system, and it discusses the various stages of the design in realising the planned target. The thesis is organised as follows:

- The first part of Chapter 2 presents a general background of the field of cryptography, starting with basic definitions of the terminology used in this field, followed by a classification of cryptography, then possible network structures. Next, three well-known block ciphers are described, and after that the modes of operation of block ciphers are explained and types of cryptanalytic attack are outlined. The second part of the chapter discusses the background of the FPGA, programmable connectivity technologies, the design flow, and categories of design architecture.
- Chapter 3 explains the NTTs involved in the design for the purpose of enhancing diffusion. The chapter gives detailed descriptions of these transforms, and the necessary analyses are carried on to evaluate their diffusion power.

- Chapter 4 addresses the subject of confusion, which is accomplished in the design by the incorporation of a substitution box (S-box). The chapter begins by describing the methods available for constructing S-boxes. Then, the mechanisms for analysing them to determine their non-linear properties are discussed. In addition the non-linear properties of all S-boxes used in the current and previous standard algorithms are computed. Finally, new S-boxes are generated based on the AES S-box.
- Chapter 5 is dedicated to the design of the proposed cipher, its encryption and decryption algorithms, and the generation of the cipher key and round keys. The theoretical complexity of the algorithm is also determined. In addition, a number of test vectors are applied to the algorithm to verify the correctness of its implementation.
- The robustness of the proposed cipher towards several types of attack, especially differential and linear cryptanalysis, is discussed in Chapter 6.
- The hardware implementation of the cipher is described in Chapter 7, which is based on the FPGA system generator, where encryption and decryption circuits are designed for block sizes of 16×7 and 32×7 bits. In addition, the complexity of the cipher for different architectures is computed.
- Chapter 8 outlines the achievements that have been made in this research. In addition, recommendations are suggested concerning further work.
- The results of analyses of diffusion power based on probabilities for both the NMNT and FNT are presented in Appendices A and B, respectively.
- The results of the complete sets of test vectors run on algorithm based on the NMNT are illustrated on Appendix C.

Chapter 2

Background

This chapter provides a general background to cryptography and the FPGA. It is organised as follows: section 2.1 defines the basic terminology used in the field of cryptography, followed in section 2.2 by a classification of cryptography. Section 2.3 then demonstrates the network structures of block ciphers. Three important examples of block ciphers are then described in detail in section 2.4. The mode of their operation is explained in section 2.5, and a list of possible attacks is provided in section 2.6. The FPGA is illustrated in section 2.7 and section 2.8 discusses the relevant literature. Finally the conclusions of the chapter are drawn in section 2.9.

2.1 Basic Definitions

The basic definitions of the main terms used in cryptography are presented in this section as follows:

- Cryptology:

  This word is a combination of the Greek words *kryptos* and *logos*, and means hidden study [4]. It is the science concerned with secure communication, including two areas: cryptography and cryptanalysis.

Cryptography involves techniques designed to maintain the confidentiality of data [5]. Using cryptography sensitive information can be stored or transmitted across insecure channels without revealing its contents to anyone other than the intended recipient, where access is usually controlled by a secret key. The objective of cryptography is to provide confidentiality, data integrity, authentication and non-repudiation [6].

Cryptanalysis is the science concerned with techniques used to retrieve encrypted information without having access to the key, or to attempt to recover the key so that all information encrypted with it can be retrieved. Such techniques are also known as hacking or attacking.

- Plaintext:

  This may also be referred to as *cleartext*, and is the original intelligible information that is required to be kept confidential.

- Ciphertext:

  This is the unintelligible or gibberish form which is the result of the processing of plaintext.

- Encryption

  Encryption is the process of converting a plaintext into a ciphertext in an attempt to keep information confidential to anyone except those possessing the right key.

- Decryption

  Decryption is the inverse of encryption; a process of transformation used to return the encrypted data to its original intelligible form.

- Cipher

  A cipher is a complete system that encompasses both encryption to conceal the meaning of information, and decryption to reveal the content of the original data under the control of a key.

- Cryptographic System

  This is also referred to as a cryptosystem, consisting of the following: a plaintext space *PT*, a ciphertext space *CT*, a cipher key space *Kc*, a key generation algorithm, encryption algorithm $CT = E_{Kc}(PT)$, and decryption algorithm $PT = D_{Kc}(CT)$.

2.2 Classification of Cryptographic Systems

Cryptographic systems are categorised depending on the number of keys used by senders and receivers. If they use different but related keys the system is referred to as asymmetric or public-key cryptography. The two keys involved are a private key which should kept secret and a public key which is published. For the purposes of confidentiality, a sender encrypts a message using the receiver's public key. The receiver then decrypts the encrypted message using his private key. For authentication purposes, the sender encrypts a message using his private key, providing digital signatures. The receiver then decrypts the encrypted message using the sender's public key. These two steps can be combined to provide both confidentiality and authentication. The security of a public-key cryptosystem is based on the difficulty of solving certain mathematical problems rather than on substitution and transposition as based on the conventional cryptography. For instance, the RSA algorithm [7] is based on integer factorisation, where its difficulty is to factor a product of two large prime numbers. The ElGamal algorithm [8, 9] is based on a discrete logarithm, and its difficulty is to find the discrete logarithms over finite fields GF(P^m). Finally, Elliptic Curve Cryptography (ECC) [10, 11] is based on elliptic curve groups over finite fields. Where there are two possible types of finite fields: a prime field GF(P); and a binary field GF(2^m). The difficulty here is to find the discrete logarithm of an elliptic curve element *'k'* with respect to a given base point *'B'* and their product. The advantage of ECC compared to RSA is that the former can offer the same level of security with a much smaller size of key, providing less computational complexity.

If, on the other hand, the sender and receiver share the same key, the system is referred to as symmetric or conventional cryptography. The security of conventional cryptography rests in the key itself, which should be agreed securely between parties before establishing a session. Table 2.1 compares the key sizes (in bits) of various algorithms belonging to different categories in relation to the computational effort required by cryptanalysis [12].

Table 2.1: Key sizes comparable with reference to cryptanalysis computational efforts

Symmetric scheme	ECC-based scheme	RSA
56	112	512
112	224	2048
128	256	3072
256	512	15360

Conventional cryptography is classified into two types based on the way the plaintext is processed. The first type is called the stream cipher, which processes one input element at a time. It is developed as an approximation to the action of the one-time pad [4], which XOR the plaintext with a complete random keystream. Stream ciphers are much faster than those in the second group, the block ciphers, and do not lead to error propagation, so that a flip-bit error only affects that bit which gives the stream cipher a major advantage over a block cipher. For a slip-bit error, the error is propagated following that point. The stream cipher starts by generating a keystream (a sequence of bits), usually using a mechanism for generating binary bits called a Linear Feedback Shift Register (LFSR) or a combination of LFSRs, where the seed of the registers is most often the secret key. The keystream generated should be non predictable and looks random, where a statistical technique such as Chi-squared analysis [13] is usually applied to provide a quantitative measure of randomness. Encryption is accomplished by combining a keystream with a plaintext, mostly with bitwise XOR operation. The RC4 algorithm [14] is a prominent example of a stream cipher.

The second group are block ciphers. In this scheme instead, of processing a single element at a time all block elements are processed simultaneously under the control of a secret key to generate a ciphertext block of equal size.

Two basic operations are used for transforming plaintext into ciphertext. Substitution maps the elements onto one another, and transposition relocates the elements. The cipher usually consists of multiple stages of these operations, yielding what is known as a product cipher.

In his famous paper on secrecy systems, Shannon [15] highlighted that the fundamental principles in designing a secure block cipher are confusion and diffusion. In simple terms, confusion consists of complicating the relationship between the statistics of plaintext bits, ciphertext bits and key bits. By preventing the properties of key and plaintext bits from being reflected in the corresponding ciphertext bits, this is achieved mostly by applying substitution operations. Diffusion aims to dissipate the statistical structure of the input into as much of the output statistics as possible, by spreading out the influence of each individual plaintext bit over many ciphertext bits. This can be accomplished using transposition operations during which after a number of rounds, each ciphertext bit should be a function of all input bits. The main disadvantage of this scheme is error propagation, since of these dependence a single ciphertext bit error may causes many errors in the corresponding plaintext block.

A block cipher can be implemented using various modes of operation depending on the application, and certain modes of operation make a block cipher in effect work as a stream cipher. These modes of operation are explained in detail in section 2.5.

Many algorithms are block ciphers, such as AES [3, 16], Blowfish [17], Camellia [18], CAST [19, 20], DES [2], IDEA [21, 22], MARS [23], RC5 [24], RC6 [25], SAFER [26], Serpent [27], Skipjack [28] and Twofish [29]. Some of these are described in sections 2.4 and 2.8.

Both conventional and public-key cryptographies can be combined to produce what is known as a hybrid system which can benefit from both sets of properties; those of the latter concerned with sorting the distribution of a secret key and those of the former used to secure rapid data processing [5, 6].

2.3 Block Cipher Network Structures

There are different types of network structures, all of which are designed according to Shannon's [15] principles of confusion and diffusion, so as to ensure data confidentiality after a number of rounds. The most well-known network structures for a block cipher are listed below:

2.3.1 Feistel Networks

This most widely used scheme was proposed by Feistel [30] in 1973 using the concept of the product cipher, which iterates the execution of a few simple components to achieve strongly secure results. Since then the Feistel structure has been utilised for different well-known block ciphers; for instance, Blowfish [17], CAST [19, 20], DES [2], FEAL [31], GOST [32], Khufu and Khafre [33], LOKI [34] and RC5 [24].

The inputs to the encryption algorithm, as shown in Figure 2.1, are a block of plaintext and a key. The plaintext block is split into two halves of equal length; only one half is modified at each round. Then the two halves are swapped and enter the following round. After processing the last round, the two halves are combined to produce the ciphertext. In this scheme, substitution is performed via the round function *F*, and permutation is achieved through swapping the two halves. An important feature of the Feistel network is that the algorithm is invertible, and hence both encryption and decryption are processed in the same algorithm taking into consideration the order of the keys. This is unlike the round function which needs not be invertible.

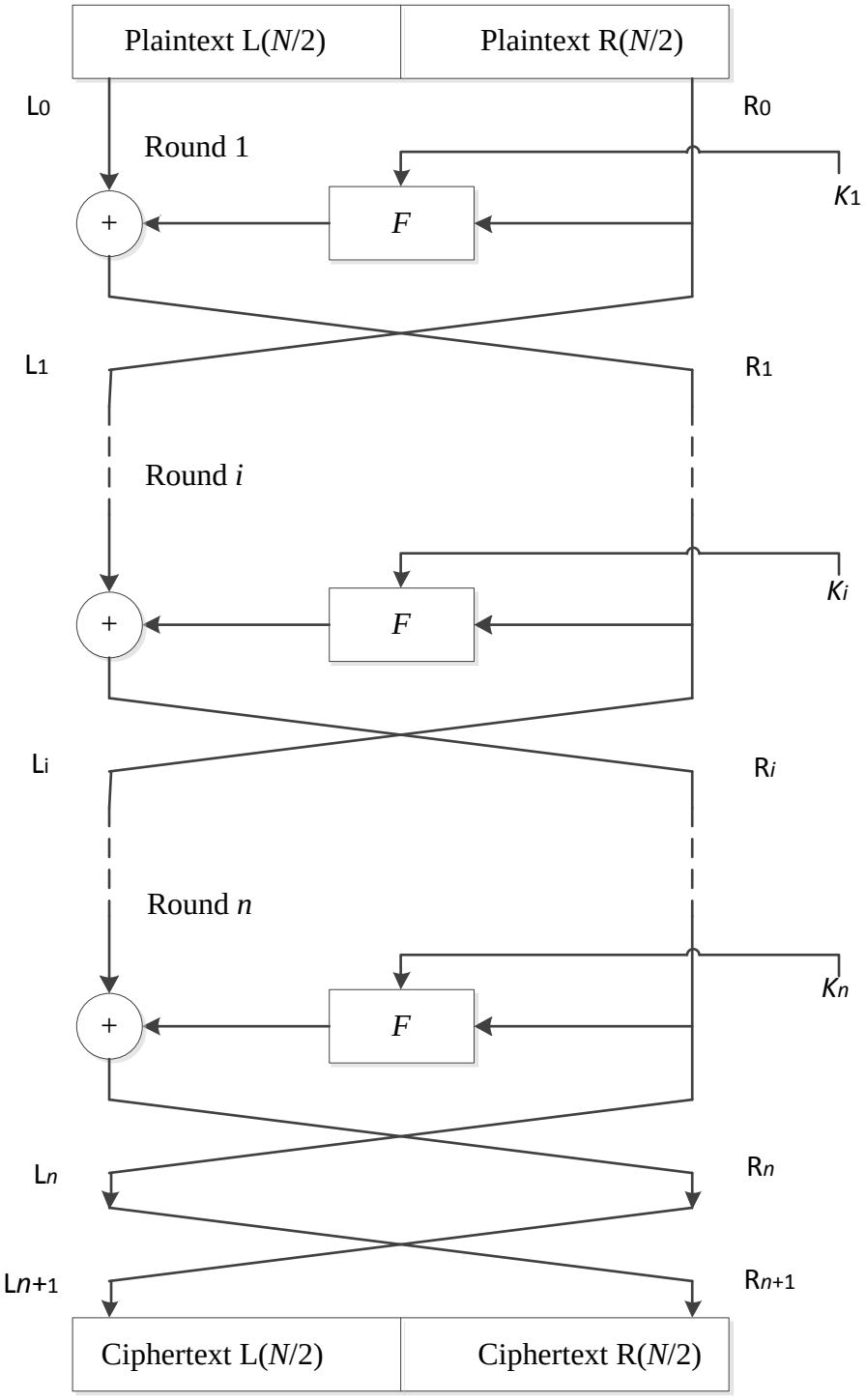


Figure 2.1: Feistel Network Structure

2.3.2 Generalised Feistel Networks

In the Feistel network, the input is split into two halves of equal length. However, in generalised Feistel networks [35, 36], some structures known as unbalanced Feistel networks may be involved where the input is either split into two halves of different lengths or split into more than two. For the latter case, different schemes are available

where a network can be constructed with one or more round functions and their inputs can be from one or more than one source, as shown in Figures 2.2-2.5. Examples of block ciphers that are based on generalised Feistel Networks: the CAST-256 [20] which uses the type-1 structure; RC6 [25], based on type-2; and MARS [23], type-3.

The scheme can be homogenous, which means that the round function is the same in all rounds. Conversely, the scheme can be heterogeneous [35], such that the round function is not identical in all rounds. The advantage of this scheme lies in the difficulty of constructing its characteristics with high probability throughout all rounds, due to the alterations in round functions between the rounds. This provides high resistance to attacks, but implementing such schemes is costly and analysing them is complicated.

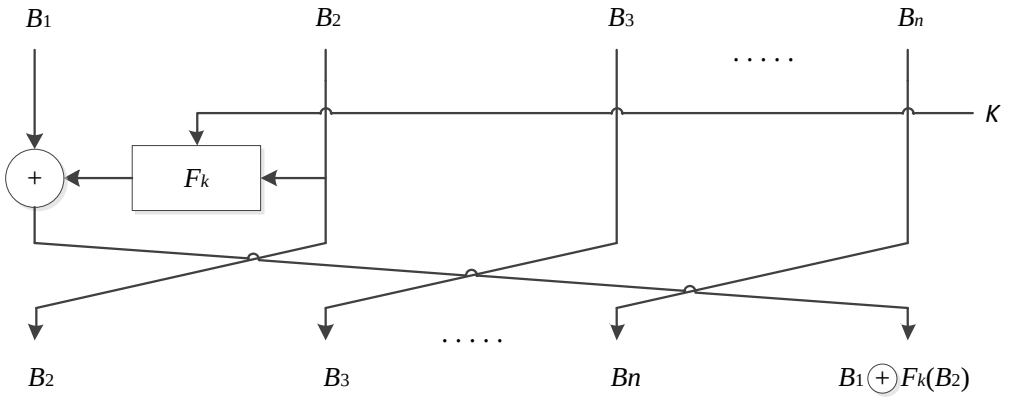


Figure 2.2: Type-1 Generalised Feistel Network Structure

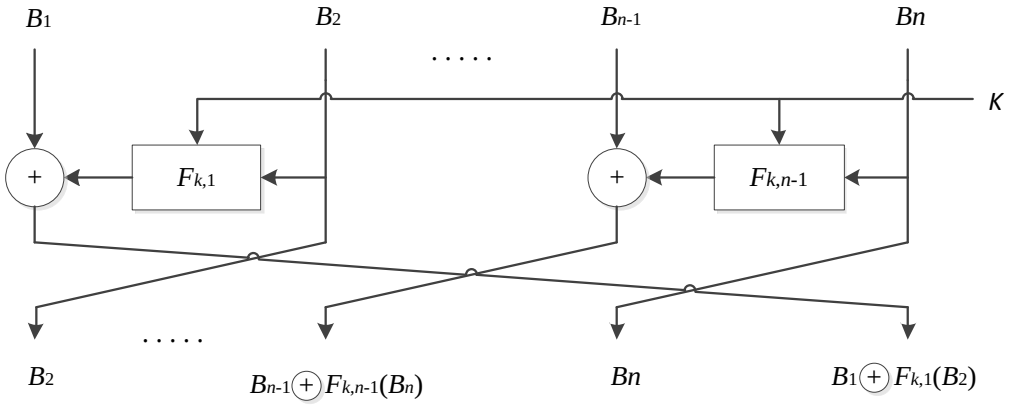


Figure 2.3: Type-2 Generalised Feistel Network Structure

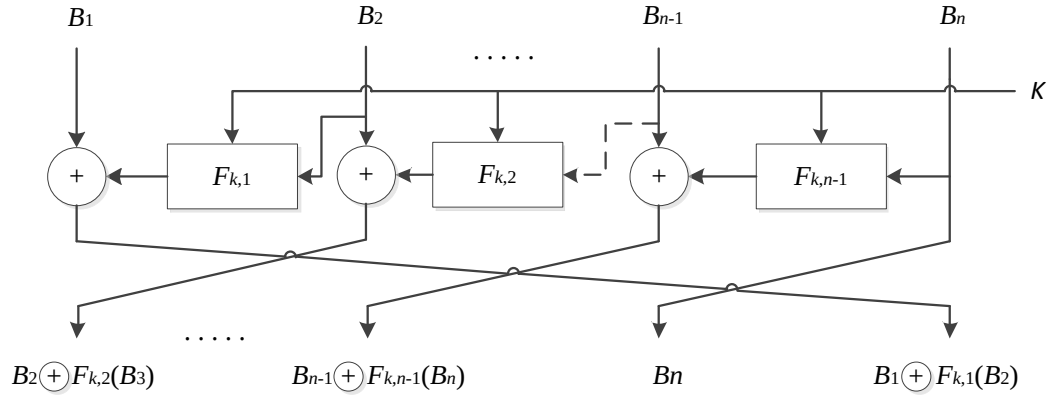


Figure 2.4: Type-3 Generalised Feistel Network Structure

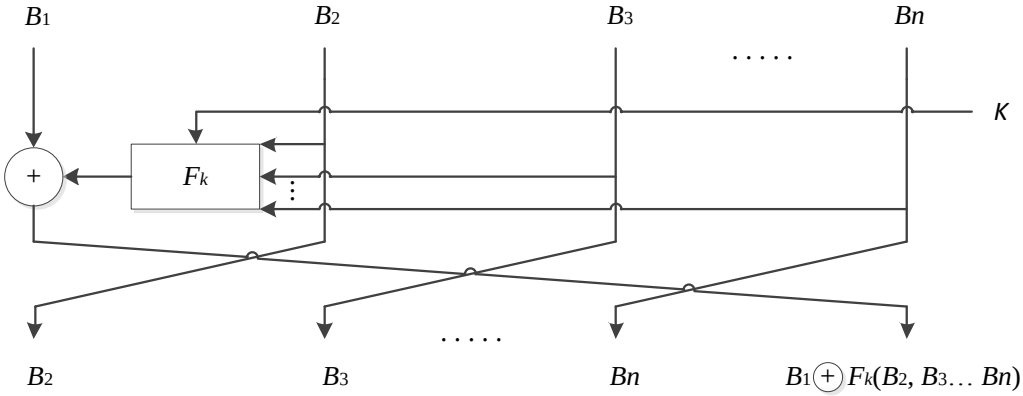


Figure 2.5: Unbalanced Feistel Network Structure

2.3.3 Substitution-Permutation Networks

The substitution-permutation network (SPN) directly implements Shannon's [15] principles of confusion and diffusion. It accepts a block of plaintext and a key as inputs, processing the plaintext by alternating the layers of substitution (S-boxes) and permutation (P-boxes) throughout the rounds to produce a ciphertext. Figure 2.6, illustrates the general structure of the scheme.

The S-box maps a fixed number of bits into other bits, and the mapping is conducted by bijection (one-to-one) to ensure invertibility. Chapter 4 describes and analyses a number of S-boxes. The P-box spreads the output bits of each S-box to as many S-boxes inputs as possible during the following rounds. The current industry standard, the AES, is based on this scheme as are other well-known block ciphers such as Safer [26], Serpent [27], Shark [37] and Square [38].

In this type of network, decryption is achieved by applying the algorithm as well as the round keys in reverse order. The latter are derived by means of a key schedule algorithm, and in addition, inverse versions of both the S-boxes and the P-boxes are used.

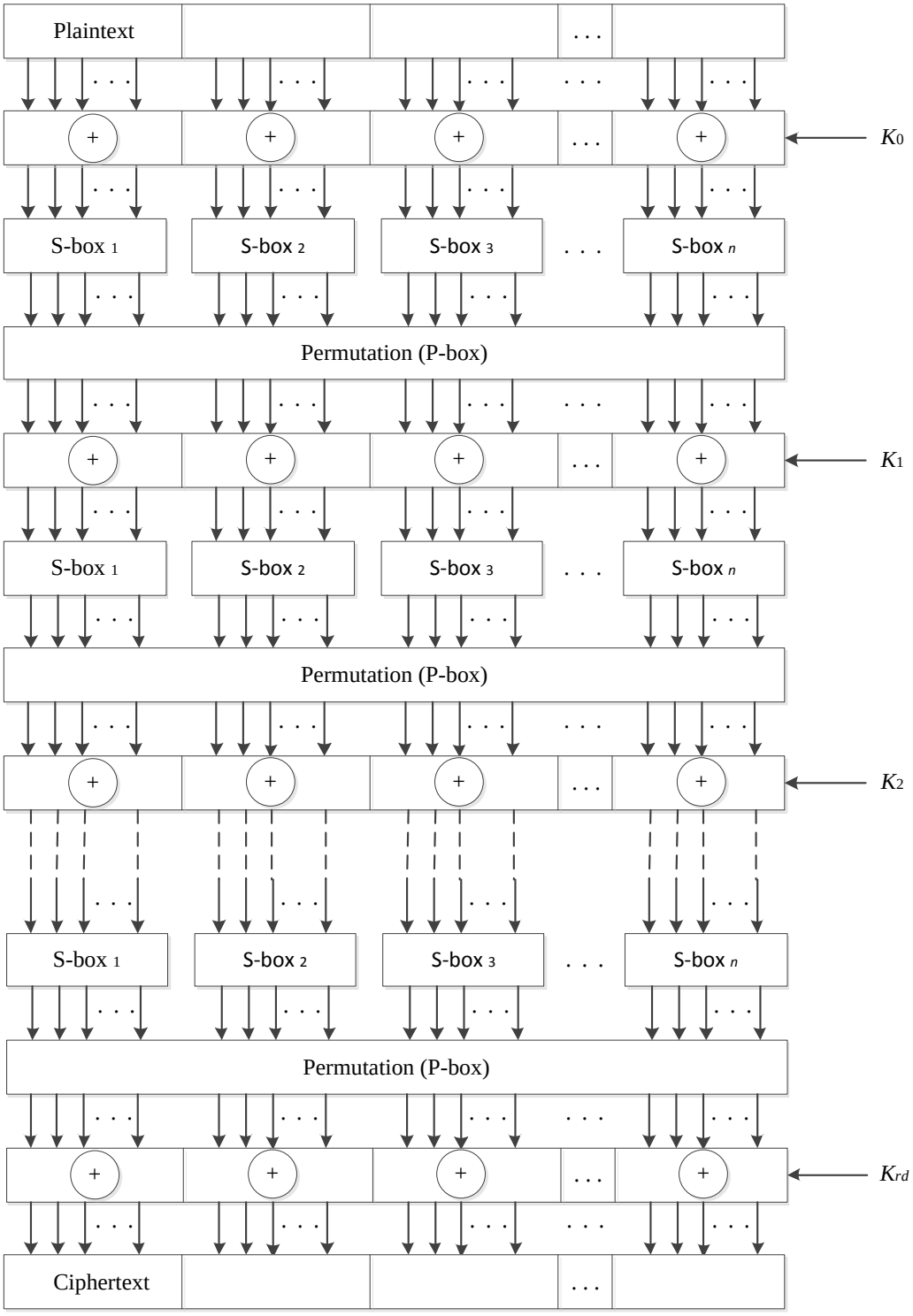


Figure 2.6: Substitution-Permutation Network

2.4 Examples of Block Ciphers

In this section, three well-known block ciphers are described in detail; two of which, namely the DES and AES are the former and the current industry standards, respectively.

2.4.1 Data Encryption Standard

The DES was adopted in 1977 by the US. National Bureau of Standards (NBS), now known as the NIST, as a Federal Information Processing Standard (FIPS PUB 46) for unclassified government communications [2]. It was also approved in 1981 by the American National Standards Institute (ANSI) as a private-sector standard (ANSI X3.92) [39].

The DES is a block cipher based on the Feistel network structure, which encrypts and decrypts blocks of data of 64-bit size under the control of a key 56-bit in length. The 56-bit of the key are extracted from a 64-bit string, while the remaining 8-bit are used for detecting errors among the bytes of the key. This involves a parity check, which is achieved by setting the least significant bit (LSB) of each byte such that the resulting parity of that byte is odd.

The algorithm, as shown in Figure 2.7, starts by initial permutation (*IP*) followed by 16 identical key dependent rounds of transformation ignoring the final swap. The ciphertext is then produced after passing the output through a final permutation which is the inverse of the initial permutation (IP^{-1}). Since the design is based on the Feistel structure, both encryption and decryption use the same algorithm except in terms of the order of the round keys.

After initial permutation the input is equally split into two halves, which are both processed through subsequent rounds according to equations 2.1 and 2.2, respectively.

$$L_i = R_{i-1} \tag{2.1}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \tag{2.2}$$

where L and R stand for the left and right half of the data, respectively,
$\oplus$ denotes bitwise XOR operation (bit-by-bit addition modulo 2).

The round transformation consists of four layers, which are expansion permutation, round key addition, element substitution and finally permutation. This round transformation works only on the right-hand half of the data.

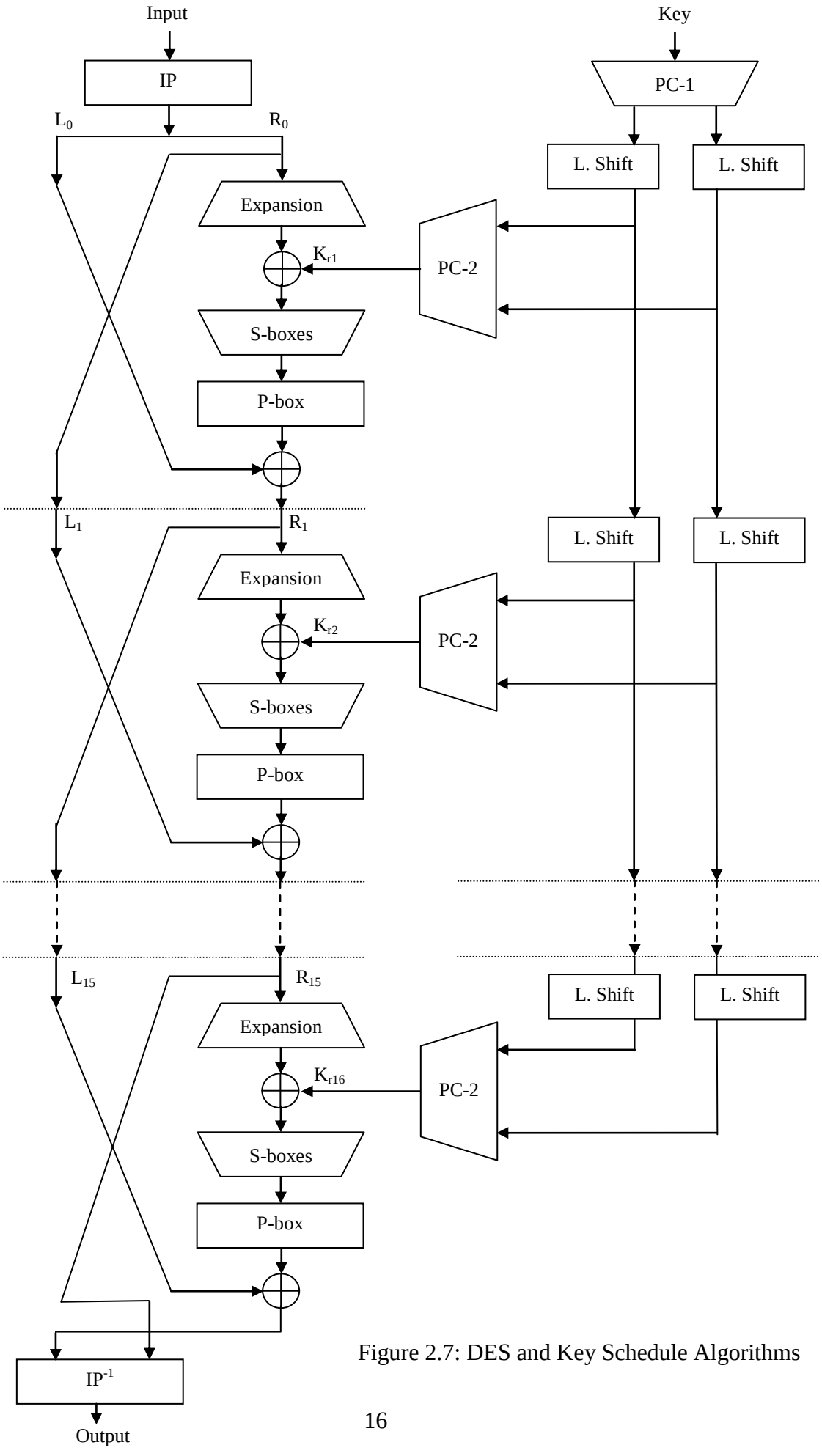


Figure 2.7: DES and Key Schedule Algorithms

The expansion permutation layer is used to expand the right half of the data from 32 to 48-bit. Expansion is achieved by duplicating and permutating the outer bits of every 4-bit, as illustrated in Figure 2.8. This layer improves the avalanche effect by rapidly spreading the dependency of the output bits on the input bits [5].

The output from the expansion permutation layer is XORed with the 48-bit round key. Sixteen different 48-bit round keys are generated from the 56-bit key via the key schedule algorithm. This operation is achieved by first ignoring the parity bit from each byte of the 8-byte of the key. Then the remaining 56-bit are permutated and subsequently split into two halves. Next, each half is left-shifted in a circular manner by either one offset for rounds 1, 2, 9 and 16, or two offsets for the other rounds. After that a 48-bit round sub-key is chosen out of the 56-bit. These two operations are known permuted choice (*PC*) or compression permutation, as a subset of data is chosen after permuting the all [5].

The next step after the key addition layer is the element substitution layer. Eight different 6×4 S-boxes are used, and thus the 48-bit are converted into eight 6-bit groups. The S-box conducts non-linear mapping, the six input bits to the S-box are mapped into four output bits. The S-boxes are analysed in detail in Chapter 4. The final layer of the round functions is permutation, which permutates the 32-bit resulting from the mapping.

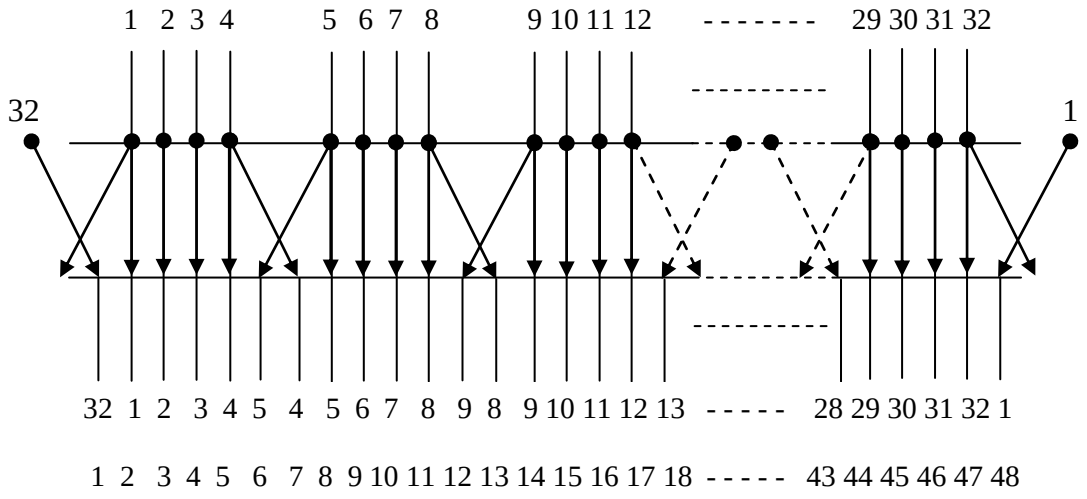


Figure 2.8: Expansion Permutation

Security of the DES

The key length of the DES algorithm is considered short in terms of processing power nowadays, and the code can be broken using an exhaustive key search attack that decrypts the encrypted message with all possible key spaces using 2^{56} or on average 2^{55} combinations to recover the right key. A variety of attacks against the DES are described in the literature. For instance, differential cryptanalysis can break the cipher with a complexity of 2^{47} of chosen-plaintext [40], and linear cryptanalysis can succeed with an availability of 2^{43} known plaintext-ciphertext pairs.

Triple DES

One of the variants of the DES algorithm is the Triple DES algorithm, also known as the Triple Data Encryption Algorithm (TDEA), which IBM suggested to improve the security of the DES algorithm by increasing the length of the key without altering the algorithm. The improvement is achieved by repeating the procedures three times using two or three different keys. Here, encryption and decryption are processed according to equations 2.3 and 2.4, respectively, where *E* and *D* refer to normal single DES encryption and decryption.

The TDEA is described in ANSI X9.52 [41], and has been recommended for use instead of the DES due to the vulnerability of the latter to exhaustive key search attacks.

$$CT = E_{k_3}\left(D_{k_2}\left(E_{k_1}(PT)\right)\right) \tag{2.3}$$

$$PT = D_{k_1}\left(E_{k_2}\left(D_{k_3}(CT)\right)\right) \tag{2.4}$$

where *CT*, *PT*, *E*, *D* and *k* stand for ciphertext, plaintext, encryption, decryption, and key, respectively.

The ANSI X9.52 standard identifies there possible keying options as follows:

1. The three keys k_1, k_2 and k_3 are independent.
2. Keys k_1 and k_2 are independent and $k_3 = k_1$.
3. The values of the three keys are the same ($k_1 = k_2 = k_3$), and are equivalent to the single DES.

The security of the system is thereby enhanced, since the exhaustive key search now requires 2^{168} attempts to break the system if all keys are independent, or 2^{112} if two of the keys are independent (as in point 2 above), which is clearly much harder than with just 2^{56} as in the single DES.

2.4.2 International Data Encryption Algorithm

The origin of the International Data Encryption Algorithm (IDEA) was the Proposed Encryption Standard (PES) algorithm [42], which was designed in 1990 as an alternative to the DES algorithm. The PES algorithm was further improved a year later after occurrences of successful differential cryptanalysis [43]. The improved algorithm, called the IPES [21], reflects much more resistant to such an attack, and it was later renamed the IDEA in 1992 [22]. Although this algorithm has not been chosen as a standard, it has been very popular as part of the Pretty Good Privacy (PGP) [44] designation.

The IDEA is an iterated block cipher that processes blocks of 64-bit size under the control of a key 128-bit in length. It consists of eight identical rounds (where the inner word swap in the last round is omitted) followed by key addition/multiplication operations; and encryption and decryption use the same algorithm apart from the order of their keys. Figure 2.9 illustrates the structure of one round of the IDEA. The input block is divided into four 16-bit words. In each round four 16-bit words, X_1-X_4, are processed with round keys each with six 16-bit words, K_1-K_6, producing four 16-bit output words, $Y1$-$Y4$. Only three operations are used namely: ($\oplus$) bitwise XOR operation, ($\boxplus$) addition modulo 2^{16}, and ($\odot$) multiplication modulo $2^{16}+1$ (where the number 0 is represented as the number 2^{16}). The ciphertext is obtained by combining the four 16-bit output words after having processed the plaintext through the eight rounds and the final key operations.

The round keys are generated from the cipher key using the round key schedule algorithm. Simple operations are used and a total of 52 16-bit words are required, 6 words in each round and 4 at the final transformation. The first eight 16-bit words are filled with the cipher key, and then eight 16-bit words are generated for every 25-bit left rotating the cipher key bits until all of the required keys have been generated.

Security of the IDEA

Although exhaustive key search attacks on the IDEA are considered impractical, as they would require the processing of on average 2^{127} decryption steps to recover the key, other types of attacks may succeed more rapidly in breaking the reduced version of IDEA and up to 5 rounds. For instance, differential cryptanalysis can work on 2.5 rounds [45], differential-linear attack on 3 rounds [46], truncated differential attack on 3.5 rounds [46] and an impossible differential attack on 4.5 rounds [47].

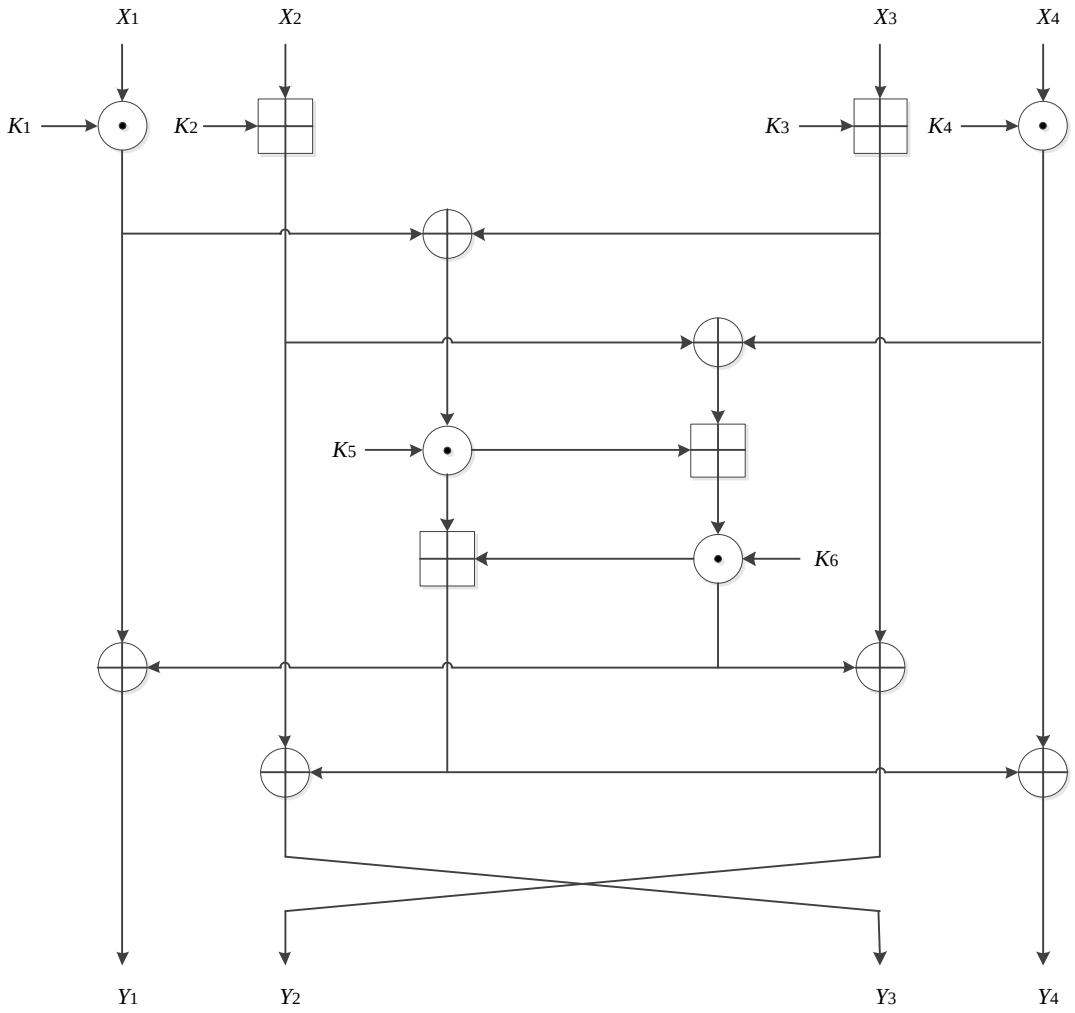


Figure 2.9: IDEA One Round Flow diagram

2.4.3 Advanced Encryption Standard

The AES algorithm [3] was announced by the NIST in the U.S. FIPS Publication 197 [16] on November 26, 2001. Replacing the old DES and the triple-DES, it was the culmination of a 5-year standardisation process in which fifteen competing designs were submitted and evaluated according to their levels of security, cost and capacity for implementation. In the event, Rijndael was chosen as the AES algorithm.

The AES is an iterated symmetric-key block cipher based on the SPN structure, and it processes data blocks of 128 bits using a cipher key 128, 192, or 256 bits long. The cipher starts by initial key addition followed by 10, 12, or 14 repeated rounds of transformation for key lengths of 128, 192, or 256 bits, respectively, as shown in Figure 2.10. All operations are performed on a 4×4 array of bytes, called state. The round transformation consists of a sequence of four different transformations, called steps, which are SubBytes, ShiftRows, MixColumns, and AddRoundKey.

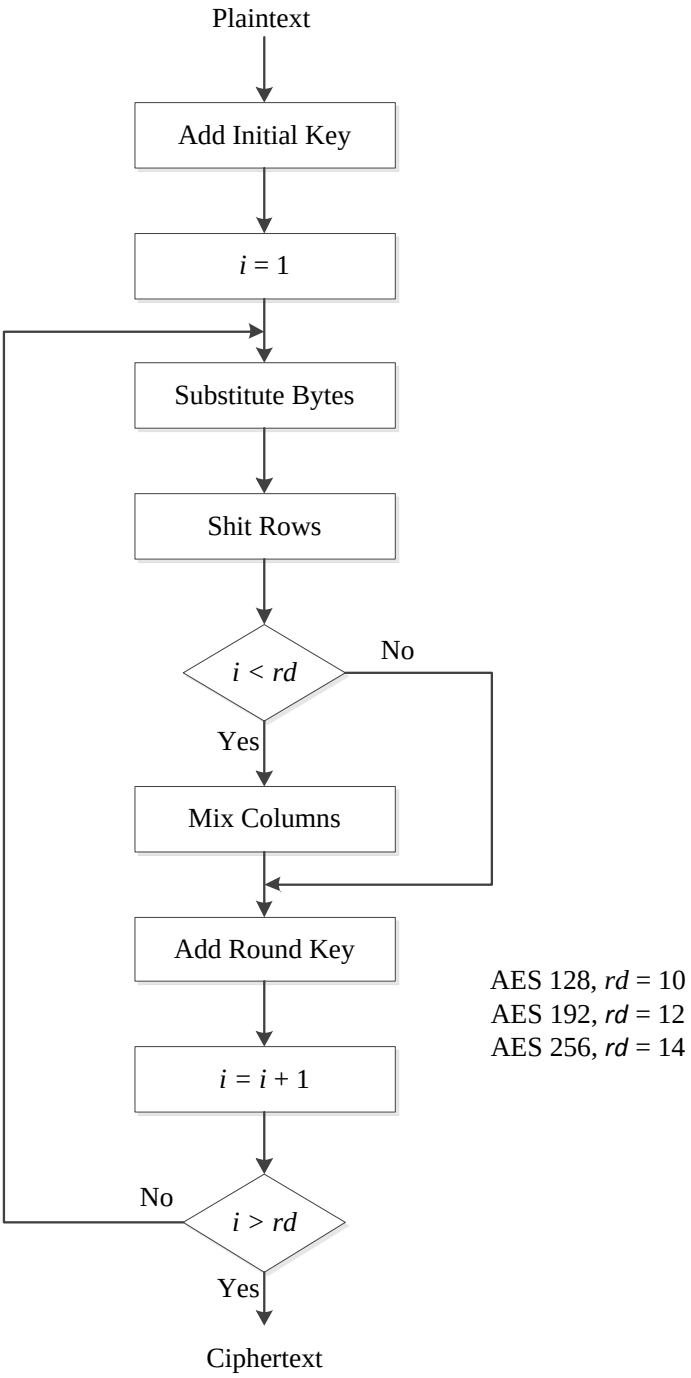


Figure 2.10: AES Encryption Flow diagram

These are applied to the data block in the encryption procedure and in reverse order with inverse transformations in the decryption procedure. The round transformation in the last round of encryption or first round of decryption is applied without the Mix/Inverse Mix column transformation.

The SubBytes transformation is a non-linear byte substitution that operates independently on each byte of the state using an S-Box. The S-box was designed on the basis of the following criteria [3]: the probability of the maximum difference

propagation has to be as small as possible; the amplitude of the IOC_{max} has to be as small as possible; and the algebraic expression of the S-Box has to be complex. Detail descriptions of the S-box and an analysis of its non-linear properties can be found in Chapter 4.

The forward and inverse ShiftRows transformation is a byte transposition achieved by cyclically shifting the rows of the state with different levels of offset. The offset depends on the length of the key. The inverse ShiftRows applies the same amount of offset but in the opposite direction. The main criterion in the design is that the offsets should be different for each row.

The MixColumns is a linear transformation used to mix the bytes of each column, thereby providing local diffusion, where the columns of the state are considered as a polynomial over Galois field GF(2^8) and the mix columns operation is undertaken by multiplying the columns modulo (x^4+1) by a fixed polynomial $c(x)$. For inverse mix columns, the alternative fixed polynomial $d(x)$ is used. The transform has an optimal diffusion power; its branch number of 5 means that for the modification of any n input bytes, at least 5-n output bytes will be modified. The $c(x)$ and $d(x)$ are given in hexadecimal values below [3]:

$$c(x) = 03 \cdot x^3 + 01 \cdot x^2 + 01 \cdot x + 02 \tag{2.5}$$

$$d(x) = 0B \cdot x^3 + 0D \cdot x^2 + 09 \cdot x + 0E \tag{2.6}$$

The round keys are derived from the cipher key by means of the key schedule algorithm, where the total number of round keys equals the number of rounds plus one, and the length of each round key is the same as the block length. In this step the value of the state is modified by bitwise XORing its value with the relevant round key.

The key schedule algorithm consists of two phases: key expansion and round key selection. In the key expansion phase the key expanded array is initially filled with a cipher key, and then the array is expanded to cover all the required round keys. Four operations are used in the expansion, which are, in sequence, Rotward, Subword, bitwise XOR with Rcon, and bitwise XOR with a word in the location [$i - N_k$], where i is the location of the processed word and N_k is the number of words in the key (key length over 32). The function of Rotward is to rotate the 4-bytes of the word to the left by one byte. The rotated word is then non-linearly mapped through the S-box. The

results are then bitwise XORed with the round constant (Rcon). Finally, the word is bitwise XORed with the word in the location $[i - N_k]$. These operations are applied in the generation process for every N_k word. The generation of the other words is achieved by merely bitwise XORing the previous word and the word in the location $[i - N_k]$. In the case that $N_k > 6$, an additional Subword operation is applied to the intermediate words before the last bitwise XOR operation for every $(i - 4)$ a multiple of N_k. The round keys are chosen in sequence in the second phase (round key selection), such that the words of round i are in the locations $W[4 \times i]$ to $W[4 \times (i + 1)] - 1$.

Security of the AES

Since the AES is a standard algorithm worldwide it has attracted much interest from cryptanalysts. The reduced version of AES has been broken by different kinds of attacks, the most efficient of which does not exceed 7-rounds for AES-128 [48, 49] and 8-rounds for AES-192 and AES-256 [50].

2.5 Block Cipher Modes of Operation

Block ciphers may employ different modes of operation, which essentially involve ways of encrypting messages of arbitrary length. The NIST defines four modes of operation for the DES algorithm [51]: the electronic codebook (ECB), cipher block chaining (CBC), cipher feedback (CFB) and output feedback (OFB). The NIST subsequently extend [52] the use of the four modes of operation to any approved FIPS symmetric key block ciphers. In addition, a new confidentiality mode of operation, the counter mode (CTR) was added. These modes of operation can support the effectiveness of the cipher; moreover they add extra properties in their implementation. For instance, in certain modes of operation a block cipher can be transformed into a stream cipher. The five modes of operation mentioned above are briefly described below.

2.5.1 Electronic Codebook Mode

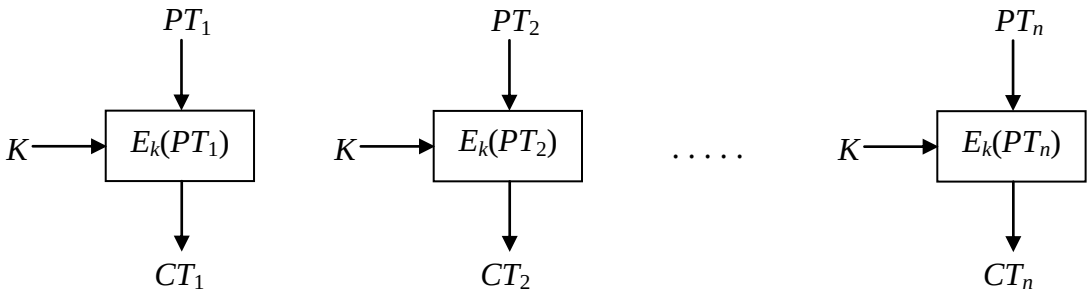
The ECB mode is the simplest mode of operation, and it allows the cipher to encrypt/decrypt each block of a message independently, so that multiple blocks can be processed in parallel. The message length must be a multiple of the block size; therefore last block must be padding accordingly if necessary. The drawback of the ECB mode is that, for a given key, encrypting the same plaintext blocks always results in the same ciphertext blocks. Therefore the ECB is recommended for use in processing messages

of short length. Furthermore, processing a block with a flipping-bit error only affects the block where the error occurs, whereas the influence of a slip-bit error is propagated to other blocks. A block diagram for the ECB mode of operation is shown in Figure 2.11.

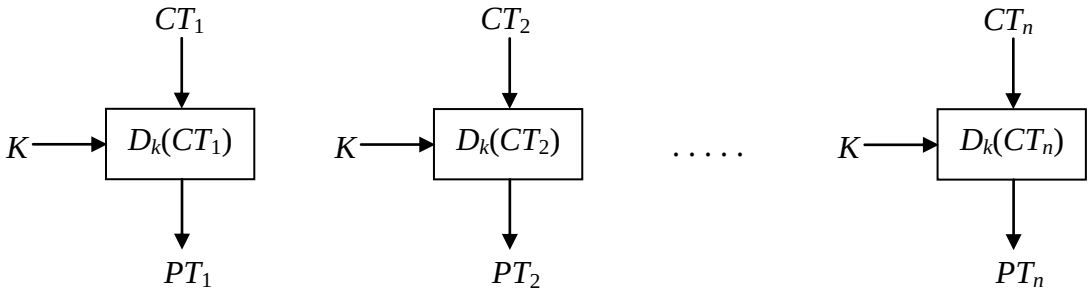
2.5.2 Cipher Block Chaining Mode

The CBC mode forces the input blocks into chains, it combines them using a bitwise XOR operation with the previously processed block before encryption or with the successive block after decryption. The first block is combined with an initial value (*IV*) which should be unpredictable, although it is not necessary for it to be secret [52]. The length of the message, as with the ECB mode, should be a multiple of the block size; accordingly padding could be expected in the last block.

The CBC mode solves the problem that arises with the ECB mode regarding of obtaining similar output blocks resulting from the processing of similar input blocks using the same key. However, due to the structure of the CBC mode which is sequential in nature, the parallel processing of multiple blocks cannot be applied in the encryption phase, whereas this it is possible in the reverse phase. Processing a block with a flipping-bit error can affect both the block where it occurs and the following block, while the influence of a slip-bit error propagates to further blocks. A block diagram of the CBC mode is illustrated in Figure 2.12.

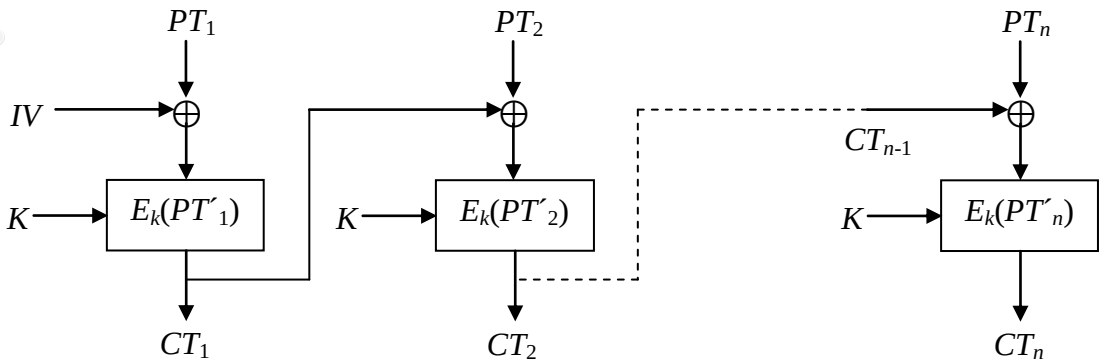


a. Encryption

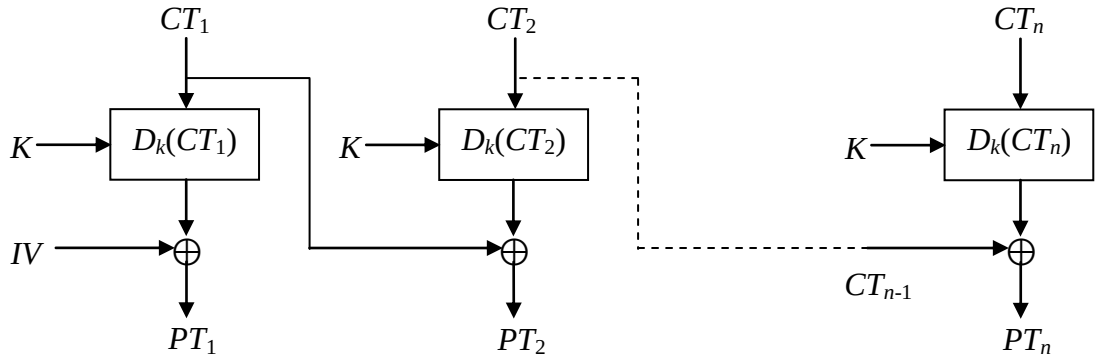


b. Decryption

Figure 2.11: Electronic Codebook Mode (ECB)



a. Encryption



b. Decryption

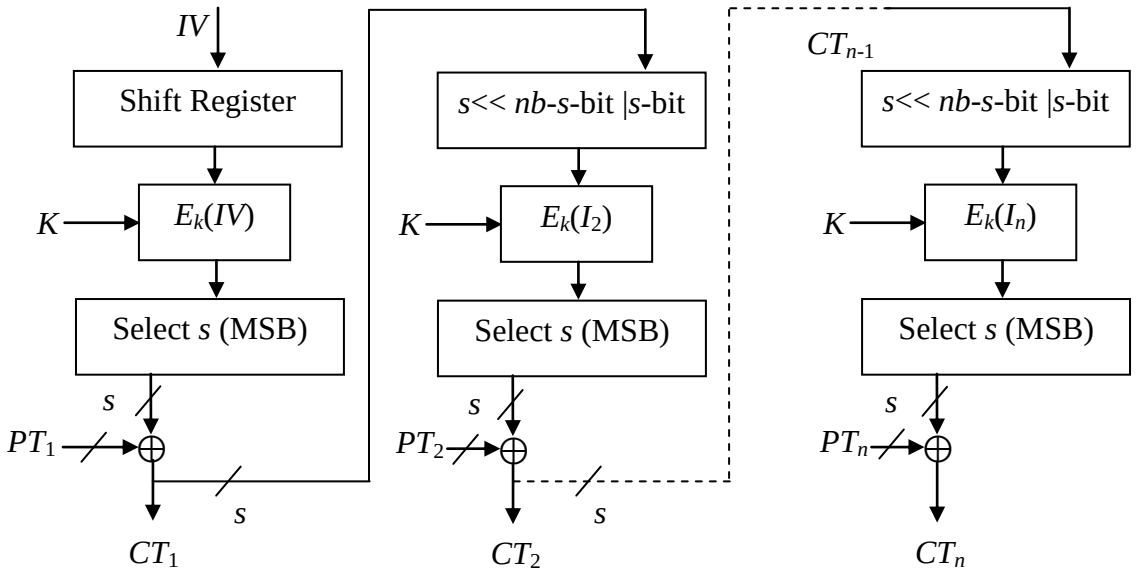
Figure 2.12: Cipher Block Chaining Mode (CBC)

2.5.3 Cipher Feedback Mode

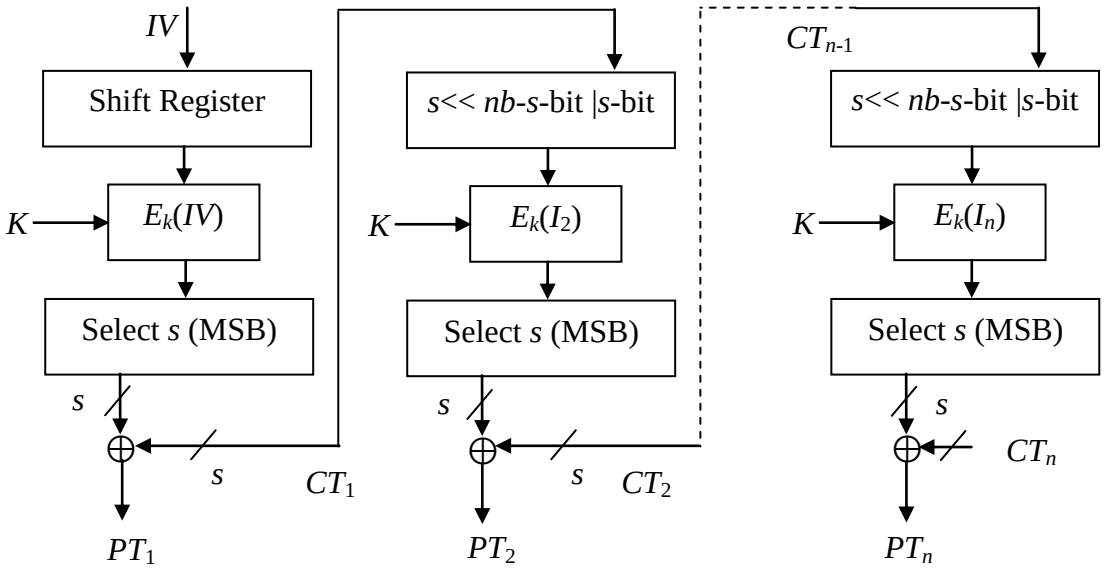
The CFB mode is a mechanism used to convert a block cipher into a stream cipher, by utilising the cipher to generate a key-stream which is later bitwise XORed with a plaintext to produce a ciphertext. At each point one bit or more can be processed depending on an integer value of *s*, which should be decided in advance. The segment *s* can be any length of 1-bit, 1-byte or a maximum value which is the length of the block (*nb*). However, the plaintext should be a multiple of *s*. This mode can sometimes be identified from the length of the segment which precedes the name of the mode; for example, a 1-bit CFB mode.

The first block is processed by accepting an *IV* as an input to the cipher. Again this value is not necessarily secret but should be unpredictable. The *s* most significant bits of the result is XORed with a plaintext segment of length *s* to produce the first ciphertext segment. The processing of the following blocks is the same, by each time the initial value is left-shifted by *s*-bit and the *s* least significant bit is filled with the previously generated ciphertext segment. Decryption is the same, swapping the locations of the

plaintext and ciphertext, as illustrated in Figure 2.13. Like the CBC mode, the CFB mode is not capable of the parallel encryption of multiple blocks due to its structure, since the processing of the following blocks requires the results of the previous step. Processing a block with a single bit error causes the error to propagate from the current block to the following *nb*/*s*-1 blocks [5]. The CFB and CBC modes can also be used for authentication purposes by producing a message authentication code (MAC), since any change in the input block will result in all subsequent output blocks also changing [53].



a. Encryption



b. Decryption

Figure 2.13: Cipher Feedback Mode (CFB)

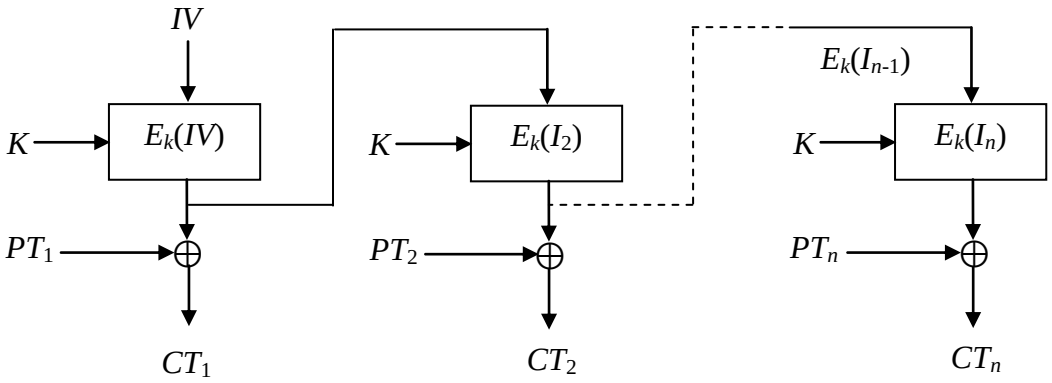
2.5.4 Output Feedback Mode

The OFB mode as shown in Figure 2.14 has a similar structure to the CFB mode. The only difference is the input to the encryption, the feedback. The feedback in OFB mode is the encryption output, whereas in CFB mode it is the ciphertext segment. The segment in OFB mode has a length equal to that of the block.

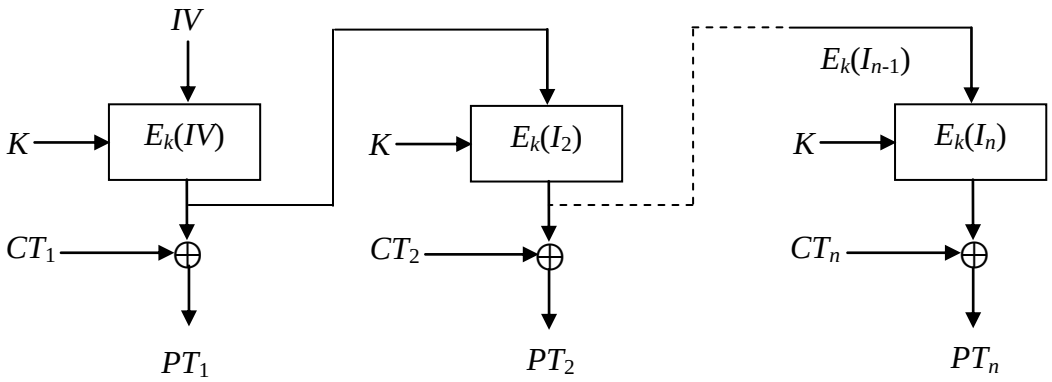
The *IV* in this mode is nonce; that is, for a given key, different values of *IV* should be used when processing different messages.

One advantage of the OFB mode is that a flipping-bit error does not propagate, and only the recovered value is affected. On the other hand, slipping-bit error causes a loss of synchronisation resulting from an incorrect message recovered from that point onward.

The parallel processing of multiple blocks again cannot be achieved due to the structure of this mode; however the key-stream can be generated in advance as the values of *IV* and the key are known. Later encryption/decryption can be processed by bitwise XORing the plaintext/ciphertext with the key-stream already generated, providing fast implementation.



a. Encryption



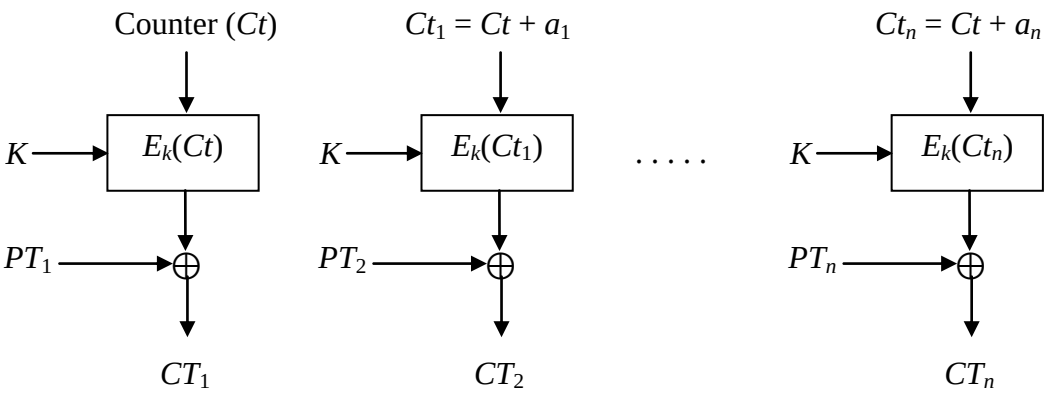
b. Decryption

Figure 2.14: Output Feedback Mode (OFB)

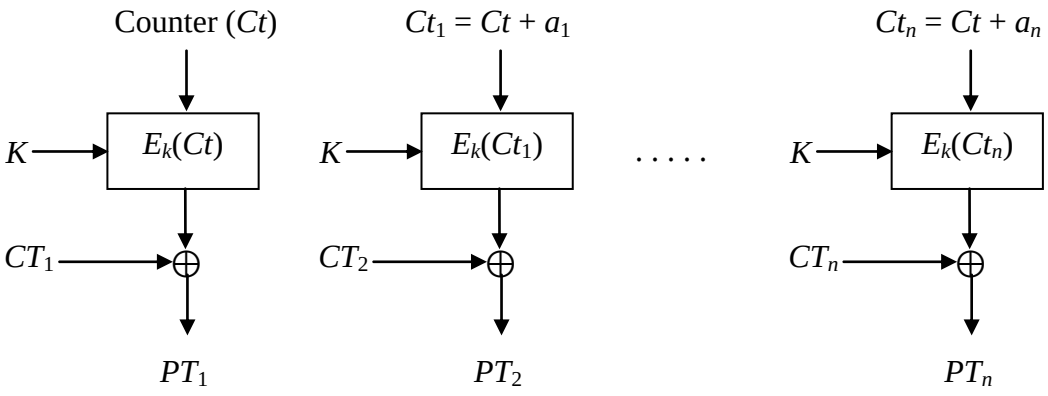
2.5.5 Counter Mode

In CTR mode, as shown in Figure 2.15, the input to the encryption is a counter, and the output from the encryption is bitwise XORed with the plaintext to produce the ciphertext and vice versa. The counter is given an initial value using random-sequence generator, then for following blocks the counter value is incremented by 1 or a certain constant. The only restriction here is that the counter value should not be used in more than one block for a given key across all messages.

The CTR mode is very fast, and multiple blocks can be implemented in parallel. In addition, all processing can be prepared in advance, and accordingly a sequence of XOR operations is only applied when the plaintext/ciphertext becomes available. Errors resulting from flip or slip-bit have exactly the same impacts as those in the OFB mode.



a. Encryption



b. Decryption

Figure 2.15: Counter Mode (CTR)

2.6 Types of Cryptanalytic Attacks

Cryptanalytic attacks are classified depending on the amount of information available to the cryptanalyst. Kerckhoff [54], has stated that the system should be secure even if all the details are exposed except the secret key; that is do not rely on "security through obscurity". Accordingly, for all types of cryptanalytic attacks listed below, it is assumed that the cryptanalyst has full details of the algorithm. The mission of the cryptanalyst is to recover the plaintext or, furthermore, to deduce the key, in order to recover all messages encrypted with that key. Knudsen [55], classified the breaking of algorithm into four categories: a total break, i.e., recovering the key; global deduction, allowing cryptanalyst to decrypt messages with an alternative algorithm without accessing the key; instance deduction, recovering the plaintext; and finally information deduction, gaining information about part of the plaintext or the key bits. The different types of cryptanalytic attacks are listed below [5, 12]:

1. Ciphertext-only Attack

   In this kind of attack an adversary has the least amount of information, perhaps only the ciphertext of a few messages. Any algorithm subject to a successful attack of this kind is considered to be weak and insecure [5, 6].

2. Known-plaintext Attack

   In a known-plaintext attack further information is at hand in addition to the ciphertext of several messages, for example their sources are also known. An intruder's goal is to deduce the key, and a typical example of such an attack is linear cryptanalysis [56].

3. Chosen-plaintext Attack

   A well known chosen-plaintext attack is differential cryptanalysis [43], in which the adversary not only has access to plaintext-ciphertext pairs, but also the capability to choose plaintexts, encrypt them and derive their outputs. His job then is to deduce the key.

4. Adaptive Chosen-plaintext Attack

   The adaptive chosen-plaintext attack is a type of chosen-plaintext attack with extra facilities, where an adversary can update his choice according to the results obtained from the previous encryption.

5. Chosen-ciphertext Attack

   This attack allows an adversary to decrypt a chosen ciphertext and gain its corresponding plaintext. The objective again is to recover the key.

6. Chosen-text Attack

   This is a more powerful attack combining the chosen-plaintext and chosen-ciphertext attacks, allowing the adversary to encrypt or decrypt the amount of text required to recover the key.

It is worth noting that an algorithm is considered to be unconditionally secure if it impossible to recover a plaintext for a given ciphertext without knowledge of the key, even if time and resources are unlimited. A one-time pad [4] is the only type of algorithm which falls into this category, however this is considered impractical as it requires the length of a key to be equal to that of a message. Therefore, to be secure, algorithms not in the latter category should meet the following criteria [12]:

- The cost of recovering the data is more expensive than the actual value of the information.
- The time required to recover the data exceeds the lifespan of the information's utility.

2.7 Field Programmable Gate Array

FPGA technology was developed in 1984 by Xilinx. It is an integrated circuitry that contains a large number of configurable logic blocks (CLBs) along with configurable interconnections between these blocks, including configurable general purpose input/output (I/O) interfaces. By configuring a number of these blocks with corresponding links, a variety of applications can be performed. At the beginning most designers used FPGAs for limited data processing tasks such as glue logic implementations for connecting and interfacing between different large blocks [57, 58]. A few years later, with the evolution of silicon technology, the FPGA was migrated from a simple technology providing programmable connectivity between different system components into a complete programmable system component in its own right. Nowadays, FPGAs contain millions of gates, large amounts of memory, embedded multipliers, adders, DSP functions, microprocessor cores and high speed programmable I/O interfaces, in addition to parallelism facilities, resulting in powerful devices that can be used in a wide range of applications. Moreover the development cost of these devices is low and application prototypes can be produced in a short time [57, 59, 60].

Three different epochs of development of the FPGA technology are pointed out in [58]: invention, expansion, and accumulation. These eras were classified depending on

the ability of the FPGA related to the application problem size, which was very limited in the first age and became far outweigh in the latter age.

A CLB consists of a number of slices, which may be 2 or 4 depending on the device. Each slice comprises generally of 2 core building blocks. The logic cells (LCs) consist of look-up tables (LUTs), multiplexers (MUXs), registers and special high-speed carry logics for the purposes of arithmetic operations, which establish dedicated interconnection between the cells, a diagram of an internal structure of the FPGA is illustrated in Figure 2.16, which is depicted from [57]. Each LUT can also be used as a RAM (distributed RAM) or as a shift register. For instance, a 4-bit LUT can be used as a 16×1 RAM or a 16-bit shift register. Two different types of RAM are available: distributed RAMs and block RAMs. Their control signals can also be configured; for instance the clock signal can be configured to be either rising or falling-edge, enabling a signal which is used to control the read and write operations and set/reset signals that force the data output into a predefined value. The device consists of a large number of embedded Block RAMs, which are placed in columns or around the device. Block RAMs can be used individually or some of them can be combined to form single large blocks. Depending on the application, these RAMs can be used as single-port RAM, dual-port RAM or first-in first-out (FIFO).

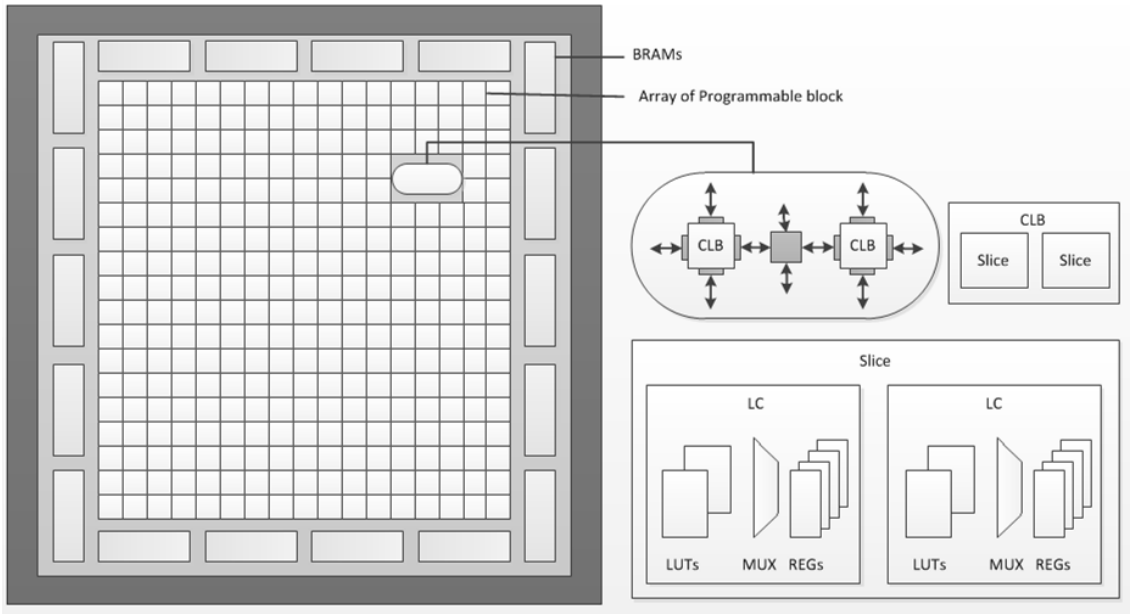


Figure 2.16: FPGA Internal Structure

2.7.1 Programmable Connectivity Technologies

The programmable connectivity between blocks is based on three technologies: antifuse, E^2PROM and SRAM [57]. Antifuse is a one-time programmable (OTP) link, and once a fuse is blown there is no way to remove it. The fuse acts as an open circuit due to its high resistance, and programming such a fuse is achieved by applying pulses with relatively high voltage and current using a special device programmer. Antifuse-based FPGAs are non-volatile, and hence there is no need to upload a configuration file when a system is powered up, thus saving the time and memory it would require to carry such a file. In addition, the device is immune to the effects of radiation due to its internal structure, which is known as 'rad hard', qualifying it to be used for special applications if accompanied with flip-flop radiation protection, such as in the use of a triple redundancy design to makes all environment radiation intensive.

The second technique is based on electrically erasable programmable read-only memory (E^2PROM). The configuration of E^2PROM-based FPGA devices can either be programmed off-line using a device programmer or may be in-system programmable (ISP); however the programming time is around three times that of devices based on SRAM. The data contained in this device, as with the previous one, is non-volatile, and therefore when the device is powered up the configuration is already available, which is known as "instant on". The static power consumption of E^2PROM-based FPGA is, however, considered to be high due to the large number of internal pull-up resisters [57].

Finally, programming may be based on static random-access memory (SRAM). This type is considered to be the state-of-the-art, and most current FPGA devices are configured based on the use of SRAM cells. The main advantage of these devices is that they can be reconfigured over and over again. Although, SRAM-based devices are volatile; meaning that they need to be reconfigured every time they are powered up, the configuration process runs very rapidly, and in addition further configurations can be run before the main task. The configuration file that is used to program a device is usually stored in an external memory, and therefore to maintain the confidentiality of this file a bitstream encryption facility is used.

The choice of the programming connectivity technology depends on the application and the computation environment in which the FPGA device is used [61].

2.7.2 FPGA Design Flow

The Xilinx Integrated Software Environment (ISE) package contains all the necessary tools needed by a designer to build up and process an FPGA design prototype. The package includes a graphical integrated development environment (IDE), design entry tools, a simulator for design verification, a synthesiser, implementation tools and finally tools for producing a bitstream for FPGA device configuration. The FPGA design flow is summarised in Figure 2.17 and described briefly below:

- Specification

  The designer should have a detailed knowledge of target device specifications.

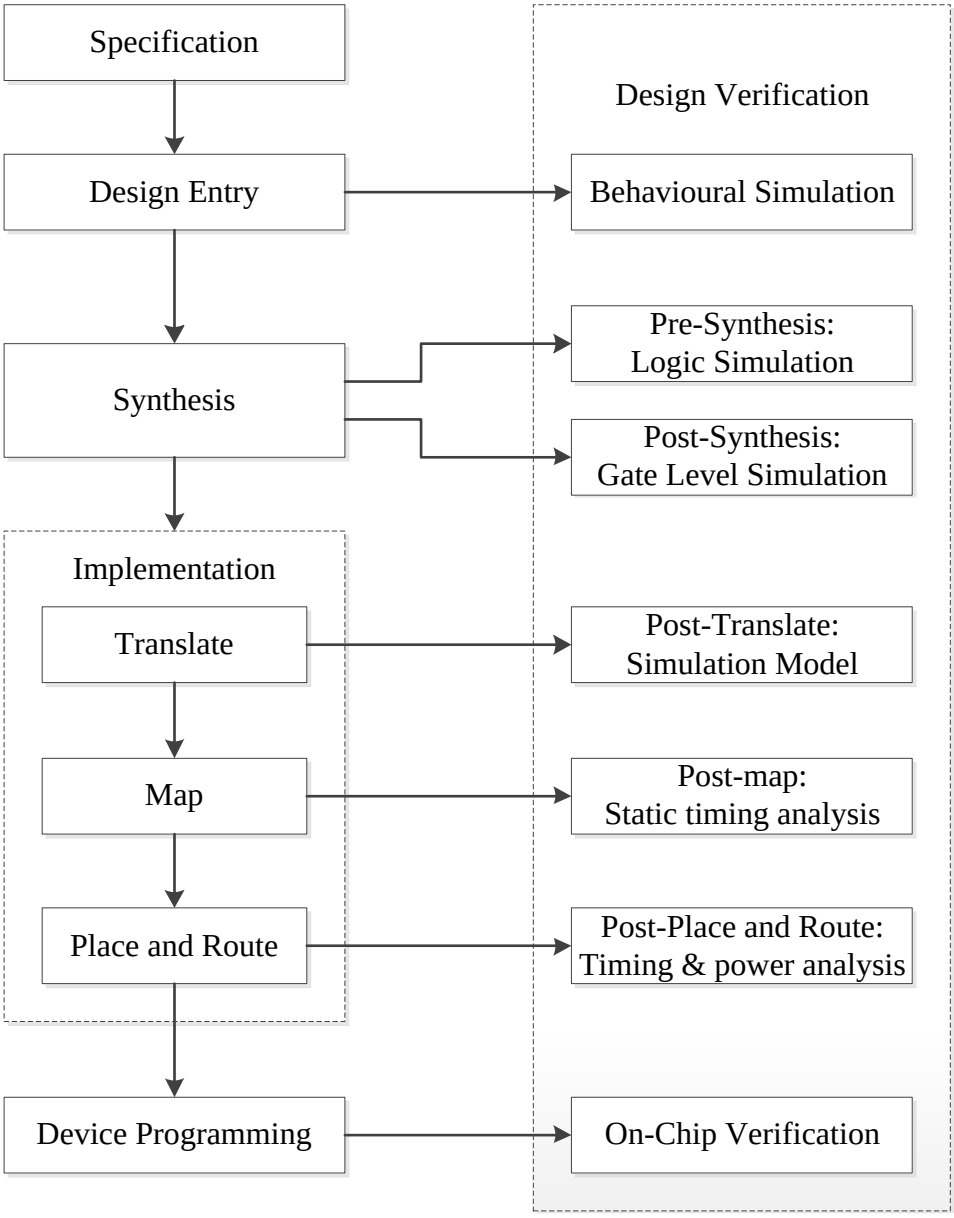


Figure 2.17: FPGA Design Flow

- Design Entry

  There are a variety of choices available for representing the design, due to the flexibility in the Xilinx ISE in accepting designs in different formats. Generally, the Register Transfer Level (RTL) is the most common level for FPGA design entry, where the circuit design can be represented in behavioural or structural models. The former are further classified into algorithmic and architectural models, while the latter is categorised as RTL, gate or switch level [57]. RTL design can be created using hardware description language (HDL) or a schematic form. The possible choices for representing the design are listed below:

  - Hardware description language (HDL), such as:
    - VHDL, which is a standard language defined in IEEE-STD-1076
    - Verilog, also a standard language defined in IEEE-STD-1364
  - High level languages:
    - SystemC, Handel-C, Catapult C, Impulse C
    - Matlab M-Code
    - Accel DSP
  - High level graphical design entry:
    - Matlab Simulink (Xilinx System Generator or Altera DSP Builder tools are used to convert the design developed in the Matlab platform into an FPGA implementation)
    - Labview
  - Configurable IP
    - Xilinx Platform Studio/Embedded Development Kit (EDK)
    - Altera SOPC Builder

- Synthesis

  Xilinx ISE has the built-in synthesiser tool called Xilinx Synthesis Technology (XST). Synthesis is used to convert the RTL code into a gate level netlist (NGC), which contains both design logic and constraints. Constraints are determined through a user constraints file (UCF) that includes timing, I/O and placement constraints. A number of properties for synthesis options have to be set prior to synthesis in order to obtain optimum performance, such as area,

speed and power reduction optimisation. Device utilisation summary can be viewed in the resulted synthesiser reports.

- Implementation

  The Xilinx ISE implementation processes the synthesised netlist (NGC) in three steps: translate, map, and place and route. Translate is used to prepare synthesised netlists for the place and route step, merging multiple netlists and constraints, adding I/O pads to top level ports, and generating the Xilinx Native Generic Database (NGD) file which can be mapped to the target device. The following step, map, is basically used to map the design into the FPGA resources available (LUTs, registers, I/Os, RAMs, DSP and others), producing a Native Circuit Description (NCD) to be used in the place and route step. The final step in the implementation is place and route. As the name suggests, it places and routes the mapped designed in the device, choosing the precise physical components to be used and establishing real connections between them.

- Device Programming

  The final step in the design flow is to generate a configuration bitstream file from the placed and routed design (NCD file), which has to be loaded in the FPGA device in order to program it. The configuration bitstream file is generated using the Xilinx ISE BitGen program, and the target FPGA device is configured using the iMPACT software. The final bitstream file can be compressed, and in addition can be encrypted before it is saved in the external memory in order to retain its confidentiality.

- Design Verification

  Design verification is carried out using the Xilinx ISE ModelSim Simulator at various points along the design flow in order to ensure functionality; following the maxim that the earlier a bug is found, the easier it is to correct.

2.7.3 Categories of Architecture Design

In general the hardware architecture of the design can fall into one of the following categories. The decision concerning which architecture the designer should choose depends on the application itself and the availability of resources. The functionality of some applications can be processed in parallel other in sequential or accept pipelining.

The differences between various types of architecture can be seen as tradeoffs between area and throughput. The possible architecture types are described below.

- Basic architecture

  Also known as iterative looping architecture, it is suitable for applications that require the usage of minimum area, and thus the focus is on area rather than throughput, in situations where resources are limited. Only one round transformation is designed in this architecture, followed by a register, and implementation is achieved by processing the data throughout this circuit *rd* times depending on the number of rounds. In general each round is completed in one clock cycle, and therefore the total number of clock cycles required to implement one block of data depends on the number of rounds. This architecture is suitable for a design with homogeneous rounds.

- Loop unrolling architecture

  Unlike the basic architecture, loop unrolling architecture focuses on throughput rather than area, duplicating the hardware required to implement each round. Hence the expansion in area depends on the number of copies, where the maximum would be the number of rounds. One block of data is processed at a time, and the optimum time is one clock cycle.

- External (outer) pipelining architecture

  This is similar to loop unrolling architecture, and in placing registers in between the unrolled rounds forms what is called the stages of a pipeline. If registers are placed between each round, the result is called full pipeline architecture. This type of architecture further increases processing speed by decreasing critical delay and processing multiple blocks of data simultaneously.

- Internal pipelining architecture

  An internal, inner or sub-pipeline architecture is achieved by placing registers inside the round in between the layers. This type of architecture is useful for complex round functions in order to reduce critical delay. Full sub-pipeline architecture refers to cases where registers are added in between all the steps within a round. Adding registers in between combinational logics reduces the delay; however, at the same time it increases the number of clock cycles required to process the algorithm.

- Hybrid pipelining architecture

  This type of architecture combines internal and external pipelining architectures and implements them simultaneously.

- Parallel architecture

  Parallel architecture is suitable for algorithms that have the ability to process their functionalities in parallel.

2.8 Literature Survey

Large contributions in the field of cryptography can be found in the literature due to the importance of this topic in maintaining the confidentiality of stored information or that transmitted over insecure channels. This section reviews prominent studies of the hardware implementation of block cipher algorithms meeting previous and current standards, as well as describing other AES candidate algorithms. In addition, a number of algorithms whose core elements are based on the NTTs are also outlined, in the following sections.

2.8.1 Data Encryption Standard

The previous standard block cipher has been described in detail in section 2.4.1, and several important published hardware implementations are discussed below.

Hardware for the DES algorithm was designed and implemented using Xilinx XC4020E [62]. An iterative structure was suggested and the S-boxes generated based on a logic design. A 26.7Mbits/sec encryption speed was achieved using 438 CLBs.

A fully unrolled and pipelined architecture for the DES algorithm has also been reported [63]. The design was implemented using Virtex V150-6 FPGA, with a throughput of 10.7 Gbits/sec using 1584 slices and a power consumption of 3.2 watts. The logics related to entering the cipher key and generating the round keys were eliminated by counting these values in software.

Full and partial pipeline architectures have been designed and implemented on the FPGA Altera platform [64]. The maximum throughput achieved was 1054.24 Mbits/sec for the 16-stage full pipeline architecture implemented on an EP1K100FC484-3 device. The throughput was 665.28 Mbits/sec for an 8-stage pipeline implemented with the same device. For the forth, second and first stage pipelines, the throughputs were 254.36

Mbits/sec, 138.88 Mbits/sec and 69.56 Mbits/sec, respectively, which were implemented with a EPF10K30(50)BC356-3 Altera device.

A design for full pipeline architecture has been proposed for the DES algorithm [65]. This design was implemented with a Xilinx Virtex XCV1000-4 FPGA device, and achieved a throughput of 3.87Gbits/sec utilising 6446 CLBs slices.

Non- and fully-pipelined architectures have also been proposed as a trade-off between area and throughput [66]. Xilinx Virtex 6 xc6vlx240t-3ff1156 FPGA technology was used in the implementation, resulting in a throughput of 4.8 Gbits/sec for the non-pipelined design, and 18.82 Gbits/sec for the pipeline architecture.

2.8.2 Advanced Encryption Standard

Different architectures and implementations for the current standard AES algorithm can be found in the literature with different achievements depending on the availability of resources and the application concerned. Some of the important published studies are described below.

In 2001, McLoone and McCanny [67] proposed an AES architectures based on utilising look-up tables to implement the entire Rijndael round functions. The design was able to achieve a throughput of 12 Gbits/sec at 93.9 MHz using 244 BRAMs and 2000 CLB slices on a Xilinx Virtex XCV812E device.

Two different architectures as tradeoffs between throughput and area for the AES algorithm have also been designed and implemented [68]. The first was based on a feedback logic that reaches a throughput of 259 Mbits/sec at 22MHz using 2358 slices on a Xilinx Virtex XCV300BG432 device, which is suitable for applications with limited resources. The second architecture uses a pipeline technique to attain high speed performance, and throughput was here improved to 3.65 Gbits/sec at 28.5 MHz using 17314 slices when implemented with a Xilinx Virtex XCV1000BG560 device.

In 2004, Hodjat and Verbauwhede [69] presented a loop unrolling and inner-round and outer-round pipelining architecture for an AES encryption processor. A maximum throughput of 21.54 Gbits/sec at 168.3 MHz was achieved using 12450 slices on a Virtex XC2VP30 FPGA device.

In 2006, Iyer et al. proposed a fully sub-pipelined architecture for the AES-128 core with both inner and outer round pipelining [70]. The architecture was implemented

using Virtex XC2VP30 device and a throughput of 26.47 Gbits/sec at 206.84 MHz was achieved using 11720 CLB slices.

A three-stage pipeline based architecture has been proposed for the AES algorithm [71] which processes three blocks of data simultaneously. The design includes both encryption and decryption, and in addition the keys are generated on-the-fly. It operates in a CBC mode and can works with the three possible AES key lengths. The architecture was simulated in Verilog HDL and implemented on a Xilinx FPGA Virtex XC2V2000-5bf957 device. A throughput of 1.315 Gbits/sec at 102.8 MHz for a key 128-bit long was achieved using 3223 slices.

Two further architectures were designed and implemented for the AES-128 algorithm on a XC2VP7X FPGA device with a LUT S-box [72]. The first architecture used a basic iterative, which implement the same hardware for all rounds and a throughput of 3.85 Gbits/sec at 300 MHz with 2599 CLB slices was obtained. In the second architecture, a one stage sub-pipelined element was added as well as one stage of outer pipelining, and throughput was improved to 6.2 Gbits/sec at 481 MHz with 3119 CLB slices.

2.8.3 Serpent (5-finalist AES candidate algorithm)

The Serpent is a block cipher based on the SPN [27]. It processes a block of 128-bit in size under the control of a key 256-bit in length, and the key should be padded to a 256-bit in case it is supplied at a lesser length. It is faster than the DES and more secure than the triple-DES. The cipher starts with an initial permutation followed by 32 identical rounds (apart from in the last round, where a linear transformation is replaced by an additional key mixing operation), and ending with a final permutation which is the inverse of the initial permutation. The initial and final permutations are included in the design for the purposes of improving both the optimisation of the implementation and computational efficiency, rather than adding to its cryptographic significance [27]. The round functions consist of a key mixing operation, S-boxes, and a linear transformation. The key mixing operation is a simple bitwise XOR operation between the intermediate result and the round key. Eight different 4×4 S-boxes are used in the algorithm, and this layer is applied in parallel by duplicating 32 similar S-boxes at each round, so that each S-box is used four times, i.e., every eight rounds. The structure of these S-boxes is an improvement over the structure of the DES S-boxes, guaranteeing higher resilience in the face of possible attacks. In the linear transformation layer the block is divided into

four sub-blocks with a word length of 32-bit each, a number of bitwise XOR operations, circular rotation and shift operations with different offsets which are applied to the 32-bit four words in order to maximise the diffusion and avalanche effects.

The decryption algorithm is different, however, in applying the encryption algorithm in reverse order, and inverse S-boxes as well as an inverse linear transformation are used instead. In addition, the round keys are applied in reverse order.

2.8.4 Twofish (5-finalist AES candidate algorithm)

The Twofish [73] is a 128-bit iterative block cipher with a key of variable length up to 256 bits. It is a Feistel based structure with additional key bitwise XORing at the input and output. The block is internally divided into four 32-bit words; the cipher runs through 16 identical rounds, and at each round the first left word is processed through the *g* function that consists of four key-dependent 8×8 S-boxes, followed by a linear transformation of a 4×4 maximum distance separable (MDS) matrix over GF(2^8). The second left word is first left-rotated by 8-bit and then processed through the same *g* function, and this can be conducted in parallel with the processing of the first word. The outputs from these two layers are fed to the following layer, the Pseudo-Hadamard Transform (PHT), which mixes its two inputs x_0 and x_1 according to equations 2.7 and 2.8. The outputs from the PHT layer are XORed with another two sub-key words, completing the *F* function. Next, the values of the two words in the right are modified by bitwise XORing their values with the two processed words. The first right word is 1-bit left-rotated before modifying its value, and the second right word is 1-bit right-rotated after modifying its value. The two halves are then interchanged and fed to the next round.

$$x_0' = x_0 + x_1 \; mod \; 2^{32} \tag{2.7}$$

$$x_1' = x_0 + 2x_1 \; mod \; 2^{32} \tag{2.8}$$

2.8.5 RC6 (5-finalist AES candidate algorithm)

The RC6 [25] is a fully parameterised block cipher, and is a further developed version of the RC5 [24] algorithm with better performance and security. The block size is 128-bit; however it can also support smaller sizes of 64 or 32-bit. The key is variable and can be any length up to 2040 bits. The number of rounds is recommended to be 20 for

the AES, although it can be any number up to 255. These parameters were selected to be suitable for the levels of security and performance efficiency required. The non-linear part of the algorithm is not based on an S-box, but instead on data-dependent rotations. The security of the RC6 relies on the extensive use of data-dependent rotation accompanied by modular integer multiplication and addition as well as bitwise XOR operations. The multiplication is based on a quadratic function given in equation 2.9. The offset for data-dependent rotation is determined by the least significant $(log_2 w)$ bits of the relevant word, where w is the word length in bits. The structure of the algorithm is a type-2 generalised Feistel network with the addition of an extra key. Decryption is performed by replacing each modular addition with subtraction, reversing the direction of rotation and using the round keys in reverse order.

$$f(x) = x\,(2x + 1) \tag{2.9}$$

2.8.6 MARS (5-finalist AES candidate algorithm)

The MARS [23] is a symmetric 128-bit block cipher with variable key length ranging from 128 to 400-bit. The algorithm is based on a type-3 generalised Feistel network, where one data word is used to modify the other three words. The MARS was designed using a mixed structure such that the middle rounds (known as the cryptographic core, or second phase) are treated differently from the outer rounds (the wrapper layers, or first and third phases) providing more resistance against attack. The cryptographic core consists of 8 rounds of keyed forward transformation and 8 rounds of keyed backward transformation. The wrapper layers are used to provide rapid mixing and key avalanche, where the first phase starts with a key addition followed by a layer of 8 rounds of unkeyed forward mixing. The third phase in the cipher involves a mixing layer of 8 rounds of unkeyed backwards operations followed by a key subtraction, which is the inverse of the first phase.

This cipher is word-oriented, in that all operations are performed at a level of 32-bit words, and a round combines a variety of operations to provide a very strong cipher. These operations are XORs, additions, subtractions, S-boxes, multiplications, and both fixed and data-dependent rotation. Decryption is achieved by processing the cipher and the key in reverse order.

2.8.7 Other Algorithms Based on NTTs

In addition to the above mentioned algorithms, some other algorithms have been designed using one of the families of NTTs.

A new algorithm was developed in [74] based on the transitional (T) transform. This is a 128-bit iterated block cipher with 5 rounds, each of which consists of four modules: non-linear key-controlled substitution; modified T transform; key-controlled multiplication; and a shuffle module. Non-linear key-controlled substitution consists of 16 operations of additions and XOR between the plaintext and the round keys, and the final result is obtained by inverting the elements processed in the transform domain. In the modified T transform, the input is diffused and shuffled through multiple stages of different T transform lengths. The output is then confused via key-controlled multiplication, which is a byte-by-byte multiplication between the key and the intermediate results. The final shuffle module first reorders the elements such that elements in even positions are located first and then the bytes are cyclically shifted depending on the value of a round key.

The same author then developed another block cipher [75] based on the NMNT which utilises a cascade of five of such transforms with different transform lengths so as to ensure high diffusion rates throughout the processing. In addition, each NMNT is preceded by a key-dependent S-box, and the different S-boxes are generated at each stage depending on the value of the stage key.

2.9 Conclusions

This chapter has introduced and briefly described the basic concepts of cryptography as well as the FPGA. Special attention has been given to block ciphers, since most of the relevant algorithms fall within this category. Different well-known block ciphers are described in detail in order to explain the methods and principles used in designing secure block ciphers. The possible architectures and design options for the FPGAs are discussed, which mainly involve a tradeoff between area and throughput. Finally a number of the most important studies in the field are outlined.

Chapter 3

Diffusion Analysis of the NTTs

In this chapter, parameter-based transforms, the NMNT and the FNT are analysed and their diffusion power are evaluated. The NMNT or FNT is used in the proposed cipher mainly to enhance system's diffusion.

Consider that diffusion power of the algorithm in the design is very important, since the number of rounds for any iterated block cipher is inversely proportional to this value. Additionally, achieving higher levels of diffusion is likely to result in a more secure system with a lower number of rounds, which improves system performance regarding speed and complexity.

The chapter is organised as follows: section 3.1 provides a brief introduction. Sections 3.2 and 3.3 explain the NMNT and FNT, respectively. The analyses are carried out in section 3.4 and the results are described in section 3.5. A discussion is given in section 3.6 and finally the conclusions of the chapter are summarised in section 3.7.

3.1 Introduction

As mentioned in the previous chapter, Shannon [15] introduced two main principles for designing secure cryptographic systems: confusion and diffusion. Substitution is one of the processes used to achieve confusion, in which the elements of the plaintext are

mapped onto other elements so as to complicate the statistical correlation between the plaintext and the corresponding ciphertext, and the strength of confusion depends on the strength of the non-linear properties of the S-box applied, as explained in detail in the next chapter. Diffusion is then a process of rearranging the plaintext into the ciphertext. Accordingly, how influential the diffusion process is can be measured by how widely the plaintext is redistributed across the ciphertext, where a small change in the variables influencing the diffusion process, such as the key or the plaintext itself, should have a significant impact on the resulting ciphertext. This effect is called the avalanche effect, and a system is considered to have good avalanche characteristics if roughly half of the output bits change for a single change in input bits [30, 76]. An extension to this criterion is called the strict avalanche criterion (SAC) [77], which states that each output bit changes with a probability of 0.5 for a single input bit change. In other words, all output bits are equally likely to change for a single input change. The same author [77] also proposed another criterion, the bit independence criterion (BIC), which states that any two output bits should change independently for any single input bit change. Moreover another concepts, called completeness, can be applied such that a system is considered complete if every output bit depends on all input bits [8, 78]. To quantify this it is necessary to verify that the system is resilient to statistical attacks and to ensure that the ciphertext remains incoherent.

Once differential [43] and linear [56] cryptanalysis had developed, designing the diffusion part of algorithms by relying only on the transposition of elements or permutation was no longer secure and such algorithms became subject to successful attack. Hence more sophisticated techniques have been used to improve and strengthen diffusion, such as the application of transforms. For instance, in the Twofish algorithm [29], a fixed transform, a 4×4 MDS matrix over GF(2^8), is utilised. Here an input vector of four bytes in length is multiplied by the MDS over GF(2^8) in each round. A MDS matrix in hexadecimal form is given below [29]:

$$MDS = \begin{pmatrix} 01 & EF & 5B & 5B \\ 5B & EF & EF & 01 \\ EF & 5B & 01 & EF \\ EF & 01 & EF & 5B \end{pmatrix} \tag{3.1}$$

In the current state-of-the-art, the AES algorithm [3], a transform called 'mix columns' is used for diffusion purposes to mix the bytes in each column to ensure local diffusion. This transform is explained in section 2.4.3.

These transforms are powerful in diffusing data, but their lengths are fixed for these dedicated algorithms. The disadvantage of this is that there is a need for an alternative algorithm in case the block size or key length becomes insufficient to meet security requirements. This might happen due to future increases in processor power and parallel processing technologies as was the case with the previous standard, the DES algorithm [2]. Accordingly, a practical solution is the use of a parameter-based transform such that the block size or key length can be changed by changing the transform size. This could then achieve the desired level of security, and would represent an algorithm which would not require revision, ensuring practical usage for the proposed lifespan.

The suggested parameter-based transforms are the NMNT and FNT, both of which belong to the NTT family. NTTs use modular arithmetic operations on a field or ring of integers, without the rounding and/or truncation errors inherent to normal floating-point operations such as those found in the discrete Fourier transform (DFT), for example. NTTs have found wide application in different areas, including digital signal processing [79], digital filtering [80, 81], image processing [82], decoding [83], cryptography [74, 75, 84] and the concealment of digital image information [85].

3.2 New Mersenne Number Transform

The NMNT is defined modulo of the Mersenne numbers (M_p) [86, 87]. This transform can be used in the forms of one or multiple dimensions [88, 89]. Fast algorithms such as the radix-2 [90, 91], radix-4 [92, 93] and split-radix [94] can be adapted to speed up its processing, where decimation can be carried out either in the time or frequency domains. The forward and inverse transforms have a similar appearance, with a scale factor $1/N$ being the only difference. The forward 1-D NMNT $X(k)$ of an integer sequence $x(n)$ with a transform length $N = 2^m$ for $m = 1,2, ..., P$ and its inverse can be defined as follows:

$$X(k) = \left\langle \sum_{n=o}^{N-1} x(n)\beta(nk) \right\rangle_{M_p} \quad k = 0, 1, 2 \ldots N\text{-}1 \tag{3.2}$$

$$x(n) = \left\langle \frac{1}{N} \sum_{k=o}^{N-1} X(k)\beta(nk) \right\rangle_{M_p} \quad n = 0, 1, 2 \ldots N\text{-}1 \tag{3.3}$$

Where:

$$\beta(nk) = \beta_1(nk) + \beta_2(nk) \tag{3.4}$$

$$\beta_1(nk) = \left\langle \mathrm{Re}\left(\alpha_1 + j\alpha_2\right)^{nk} \right\rangle_{M_P} \tag{3.5}$$

$$\beta_2(nk) = \left\langle \mathrm{Im}\left(\alpha_1 + j\alpha_2\right)^{nk} \right\rangle_{M_P} \tag{3.6}$$

$$M_p = 2^P - 1 \tag{3.7}$$

$$\alpha_1 = \pm\left\langle 2^q \right\rangle_{M_P} \tag{3.8}$$

$$\alpha_2 = \pm\left\langle -3^q \right\rangle_{M_P} \tag{3.9}$$

$$q = 2^{p-2} \tag{3.10}$$

The above kernels $\beta_1(nk)$ and $\beta_2(nk)$ are calculated for a maximum transform length of 2^{P+1}. For transform lengths less than this, the values can be calculated using the following equations:

$$\beta_1(nk) = \left\langle \mathrm{Re}\left(\left(\alpha_1 + j\alpha_2\right)^d\right)^{nk} \right\rangle_{M_P} \tag{3.11}$$

$$\beta_2(nk) = \left\langle \mathrm{Im}\left(\left(\alpha_1 + j\alpha_2\right)^d\right)^{nk} \right\rangle_{M_P} \tag{3.12}$$

where $Re(\cdot)$ and $Im(\cdot)$ stand for real and imaginary parts of the enclosed term, respectively, $\langle\cdot\rangle_{M_P}$ denotes modulo the M_p and $d = 2^{P+1}/N$, which is an integer power of two.

3.3 Fermat Number Transform

The FNT is defined modulo of the Fermat number (F_t) [81, 95]. This transform can also be used in either single or multi-dimensional forms [96]. The forward 1-D FNT and its inverse are defined as follows:

$$X(k) = \left\langle \sum_{n=o}^{N-1} x(n)\alpha^{nk} \right\rangle_{F_t} \quad k = 0,1,2,\ldots N-1 \tag{3.13}$$

$$x(n) = \left\langle \frac{1}{N} \sum_{k=o}^{N-1} X(k)\alpha^{-nk} \right\rangle_{F_t} \quad n = 0,1,2,\ldots N-1 \tag{3.14}$$

$$F_t = 2^{2^t} + 1 \tag{3.15}$$

where: t is a positive integer, such that Fermat numbers are primes for $0 \leq t \leq 4$, and composite after that, $\langle\cdot\rangle_{F_t}$ denotes the modulo F_t and α is a root of unity of order N, where N is the least positive integer such that:

$$\alpha^N = \langle 1 \rangle_{F_t}, \text{ and } \alpha^i \neq \langle 1 \rangle_{F_t}, \text{ for } 0 < i < N \tag{3.16}$$

Table 3.1 outlines the maximum transform length N_{max} and other parameters for different values of t, where the maximum transform length for Fermat primes depends on the kernel value [81] as follows:

$$N_{\max} = 2^{t+1},\ 2^{t+2}, \text{ or } 2^{2^t} \text{ for } \alpha = 2, \sqrt{2}, \text{ or } 3, \text{ respectively} \tag{3.17}$$

Table 3.1: Maximum FNT length for t = 1, 2, 3, 4

t	F_t	α	N	N_{max}
1	5	$\sqrt{2}$	8	8
		2	4	
2	17	3, $\sqrt{2}$	16	16
		2	8	
		4	4	
3	257	3	256	256
		9	128	
		81	64	
		136, $\sqrt{2}$	32	
		2	16	
		4	8	
		16	4	
4	65537	3	65536	65536
		9	32768	
		81	16384	
		6561	8192	
		54449	4096	
		61869	2048	
		19139	1024	
		15028	512	
		282	256	
		13987	128	
		8224,$\sqrt{2}$	64	
		2	32	
		4	16	
		16	8	
		256	4	

3.4 Analysis

To reflect the sensitivity of the transforms for any changes in the input or output elements, a simple example can be considered which illustrates the effect of modifying a single output (transformed) element to the input elements. The text and ASCII representations for both the input elements to the transforms and the corresponding output elements are illustrated in Figures 3.1 and 3.2 for the NMNT and FNT, respectively. The recovered plaintexts result after modifying one of the transformed elements (shadowed) are shown to be completely different, confirming the high sensitivity of the transforms regarding any changes in the input or output elements. In other words, the transforms possess good avalanche characteristics.

The calculations are achieved by applying equations 3.2 and 3.3 for the NMNT, respectively. Where $N = 8$ (plaintext length is eight strings), $P = 7$, $M_p = 2^P\text{-}1 = 127$, $\alpha_1 = \alpha_2 = 119$, $\beta(n) = 1\ \ 111\ \ 1\ \ 0\ \ 126\ \ 16\ \ 126\ \ 0$, and *PT* and *CT* in Figures 3.1 and 3.2 stand for the ASCII representations for both the plaintext and ciphertext, respectively. In addition, to implement equations 3.13 and 3.14 for the FNT, $F_t = 257$.

Plaintext:	A	n	a	l	y	s	i	s
PT :	65	110	97	108	121	115	105	115
CT :	74	16	113	64	67	110	109	94
Ciphertext:	J		q	@	C	n	m	^
CT´ :	74	16	113	64	63	110	109	94
PT´ :	1	47	33	45	57	52	41	52
Plaintext´:		/	!	-	9	4	)	4

Figure 3.1: 1-D NMNT output modification

Plaintext:	A	n	a	l	y	s	i	s
PT :	65	110	97	108	121	115	105	115
CT :	65	119	16	238	197	27	209	163
Ciphertext:	A	w		ε	┼		╤	ú
CT´ :	65	119	16	238	193	27	209	163
PT´ :	193	239	225	237	249	244	233	244
Plaintext´:	┴	∩	ß	Ø	.	⌠	Θ	⌠

Figure 3.2: 1-D FNT output modification

Two different techniques are used in scrutinising and verifying the diffusion power of the transforms. The first technique involves the calculation of the branch number of the transforms, in order to characterise the diffusion power of a linear transformation. The branch number (*BN*) of a transformation (*F*) is calculated based on equation 3.18 [3]:

$$BN(F) = \min_{a \neq 0} \{W(a) + W(F(a))\} \tag{3.18}$$

where: $W(a)$ is the bundle (element) weight (number of non-zero elements, also known number of active elements) and *F* is the linear transformation.

The branch number determines the worst case diffusion of a transform; therefore it is a lower bound for the number of active S-boxes in two consecutive rounds of a linear or differential characteristic. The branch number of a transform with maximum diffusion power is $N+1$, and by considering an input weight $W(a) = 1$, the output weight is a maximum of *N*. Therefore the branch number of a transform *F* is [3]:

$$BN(F) \leq N + 1 \tag{3.19}$$

The second technique is based on calculating the diffusion power as a range of probabilities for different cases. These cases are determined according to the kernel matrix analysis [97, 98] explained in sections 3.4.1-2. These cases differ depending on the number of modified elements and their locations. The type of element modification depends on the modified values, which may be the: same value, different values with a total sum equal to the modulus for each modified pair/elements, and different values with a total sum not equal to the modulus for each modified pair/elements. The range of probabilities for each case is calculated by counting the distance (number of differences) between the elements of the modified and unmodified versions, where diffusion power represents the results of the process over-all elements (*N*) by 100%. The results are verified by recalculating the above cases and modifying the elements in three different tests. The first test is performed by transforming the input elements and producing the initially diffused elements. Next, the input is modified and transformed and the output is compared to the transformed output of the unmodified input. The second test is performed by modifying the transformed output elements and recalculating the input elements by applying the inverse transform and comparing the original input to the inversely transformed input. The final test involves modifying the mathematical equation according to the relevant cases as shown in the explanation below, for modifying: a single element (equations 3.20 and 3.26); a single paired elements

(equations 3.21 and 3.27) and unpaired elements (equations 3.22 and 3.28), all-odd elements (equations 3.23 and 3.29), all-even elements (equations 3.24 and 3.30), and all elements (equations 3.25 and 3.31) for both the NMNT and FNT, respectively:

$$X(k) = \left\langle \sum_{n=0}^{N-1} x(n)\beta(nk) + a\beta(ik) \right\rangle_{M_p} \quad k = 0,1,2,...N-1 \tag{3.20}$$

$$X(k) = \left\langle \sum_{n=0}^{N-1} x(n)\beta(nk) + a_1\beta(ik) + a_2\beta(\left(i + N/2^x\right)k) \right\rangle_{M_p} \tag{3.21}$$

$$X(k) = \left\langle \sum_{n=0}^{N-1} x(n)\beta(nk) + a_1\beta(i_1k) + a_2\beta(i_2k) \right\rangle_{M_p} \tag{3.22}$$

$$X(k) = \left\langle \sum_{n=0}^{N-1} x(n)\beta(nk) + \sum_{n=0}^{N/2-1} a_{2n}\beta\left(2nk\right) \right\rangle_{M_p} \tag{3.23}$$

$$X(k) = \left\langle \sum_{n=0}^{N-1} x(n)\beta(nk) + \sum_{n=0}^{N/2-1} a_{2n+1}\beta\left(\left(2n+1\right)k\right) \right\rangle_{M_p} \tag{3.24}$$

$$X(k) = \left\langle \sum_{n=0}^{N-1} x(n)\beta(nk) + \sum_{n=0}^{N-1} a_n\beta\left(nk\right) \right\rangle_{M_p} \tag{3.25}$$

$$X(k) = \left\langle \sum_{n=o}^{N-1} x(n)\alpha^{nk} + a\alpha^{ik} \right\rangle_{F_t} \tag{3.26}$$

$$X(k) = \left\langle \sum_{n=o}^{N-1} x(n)\alpha^{nk} + a_1\alpha^{ik} + a_2\alpha^{(i+N/2^x)k} \right\rangle_{F_t} \tag{3.27}$$

$$X(k) = \left\langle \sum_{n=o}^{N-1} x(n)\alpha^{nk} + a_1\alpha^{i_1k} + a_2\alpha^{i_2k} \right\rangle_{F_t} \tag{3.28}$$

$$X(k) = \left\langle \sum_{n=0}^{N-1} x(n)\alpha^{nk} + \sum_{n=0}^{N/2-1} a_{2n}\alpha^{2nk} \right\rangle_{F_t} \tag{3.29}$$

$$X(k) = \left\langle \sum_{n=0}^{N-1} x(n)\beta(nk) + \sum_{n=0}^{N/2-1} a_{2n+1}\alpha^{(2n+1)k} \right\rangle_{F_t} \tag{3.30}$$

$$X(k) = \left\langle \sum_{n=0}^{N-1} x(n)\beta(nk) + \sum_{n=0}^{N-1} a_n\alpha^{nk} \right\rangle_{F_t} \tag{3.31}$$

where i is the location of the modified element ($0 \leq i \leq N$-1) and a is the modification value that is added to the initial value.

Considering all these cases is very important that, apart from determining the diffusion power, the cases that provide maximum or minimum diffusion percentages are determined. These can then be exploited or avoided in the design.

The elements of an input vector are modified at the following locations:

1. Initially, all of the single elements at even and odd locations are modified.
2. Next, all of the even/odd numbers of paired elements and up to $N/2$-1 pairs are modified at their corresponding even/odd/mix locations. This is shown in Figure 3.3 using the formula $\left(i, i+N/2^{x}\right)$, where ($1 \leq x \leq log_2 N$-1).
3. The following case is the modification of even/odd groups of unpaired elements that are situated in the even/odd/mix locations.
4. A combination is performed that requires the modification of both the paired elements at even/odd/mix locations $\left(i, i+N/2^{x}\right)$ using predetermined values and modifying the remaining unpaired elements by replacing with random values.
5. The elements are modified that reside in all-even positions, followed by the elements that reside in all-odd locations.
6. Finally, all of the elements are modified for the last time, completing this particular process within the implementation.

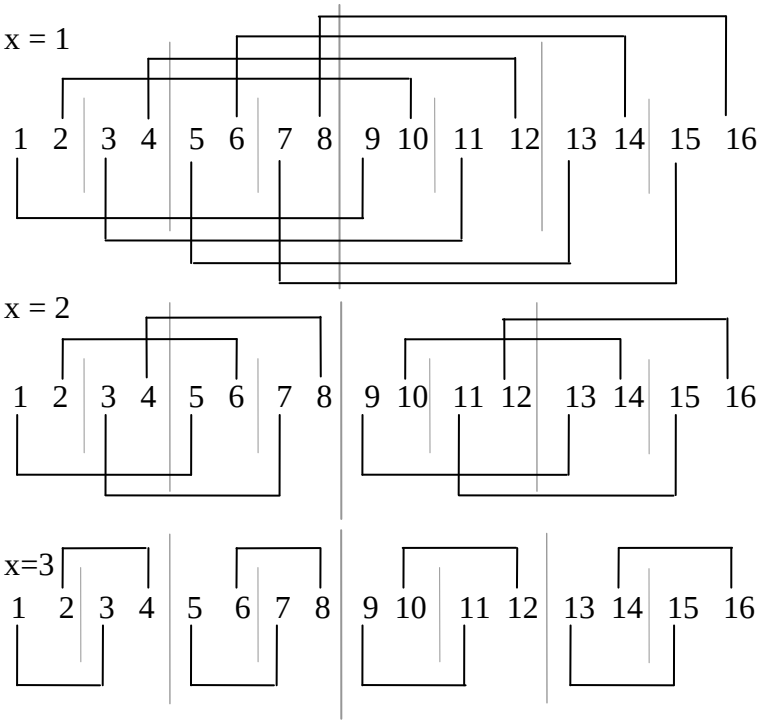


Figure 3.3: Pair distributions

3.4.1 Analysis of the NMNT Kernel Matrix

The NMNT is analysed for diffusion by first analysing the structure of the kernel in a matrix form. The following parameters are taken for illustration purposes: P = 7 and N, transform length = 16.

Then: $M_p = 2^P-1 = 127$, $\alpha_1 = 106$, $\alpha_2 = 103$

$\beta(n) = 1 \quad 82 \quad 111 \quad 82 \quad 1 \quad 3 \quad 0 \quad 124 \quad -1 \quad 45 \quad 16 \quad 45 \quad -1 \quad 124 \quad 0 \quad 3$

for $n = 0, 1, 2 \ldots N-1$

The distribution of $\beta(nk)$ in a matrix form is represented in Table 3.2, and its analysis which is suitable for any transform length is listed below:

1. The first column always consists of elements with +1 value. Accordingly, the first output element result from multiplying an input vector with the transform represents the sum of all vector elements, and its value changes with any change in the input elements, except when the sum of the total changes equals zero or the modulus (M_p).

Table 3.2: $\beta(nk)$ Matrix distribution for P = 7, M_p = 127, N = 16

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	82	111	82	1	3	0	124	-1	45	16	45	-1	124	0	3
1	111	1	0	-1	16	-1	0	1	111	1	0	-1	16	-1	0
1	82	0	45	-1	3	111	3	-1	45	0	82	1	124	16	124
1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1
1	3	16	3	1	45	0	82	-1	124	111	124	-1	82	0	45
1	0	-1	111	-1	0	1	16	1	0	-1	111	-1	0	1	16
1	124	0	3	-1	82	16	82	-1	3	0	124	1	45	111	45
1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1
1	45	111	45	1	124	0	3	-1	82	16	82	-1	3	0	124
1	16	1	0	-1	111	-1	0	1	16	1	0	-1	111	-1	0
1	45	0	82	-1	124	111	124	-1	82	0	45	1	3	16	3
1	-1	-1	1	1	-1	-1	1	1	-1	-1	1	1	-1	-1	1
1	124	16	124	1	82	0	45	-1	3	111	3	-1	45	0	82
1	0	-1	16	-1	0	1	111	1	0	-1	16	-1	0	1	111
1	3	0	124	-1	45	16	45	-1	124	0	3	1	82	111	82

2. The first row always consists of elements with +1 value. As a result, any change in the value of the first element of an input vector will be reflected equally in all output elements.
3. The sum of all elements of each column/row (except the first column/row) modulo M_p equal zero.
4. The matrix is symmetric, i.e., the matrix and its transpose are the same.
5. The even columns/rows always contain only two elements with zero value (except for N which equals 4), with $N/2$ locations between each one at odd positions (even for $N = 8$).
6. The four odd columns/rows always contain no zero elements, only +1 or ±1 at locations 1, $(N/4)+1$, $(N/2)+1$, and $(3N/4)+1$.
7. The four odd columns/rows (except for $N = 4$) always contain a maximum number of zero elements $N/4$ at locations $(N/8)+1$, $(3N/8)+1$, $(5N/8)+1$, and $(7N/8)+1$.
8. The number of elements with zero value in the odd columns/rows range from zero elements in four columns/rows (point 6 above) to the maximum $N/4$ elements in four columns/rows (point 7 above) and the rest in between $N/8$, $N/16$ etc.
9. The total number of elements with zero value in the matrix is $N(\log_2 N-2)$.
10. Element values in the second half of each even column/row are the inverse of the values of the corresponding elements in the first half (their sum modulo M_p is equal to zero).
11. Element values in the second half of each odd column/row are the same as those of the corresponding elements in the first half. Each half may further consist of another two identical halves. However, the final values of the elements of the second half are the inverse of those of the elements of the first half (except in the first column/row).
12. Any modification in the elements of an input vector has the same effect as modifying its transformed elements, as the same equation is used with a scale factor $1/N$.

3.4.2 Analysis of the FNT Kernel Matrix

The FNT kernel matrix has a structure similar to that of the previous transform (NMNT) regarding points 1 - 4, 10 and 11. The main difference is that there are no elements with a zero value in the matrix, and this has a positive effect on diffusion power.

3.5 Results

The calculations of the branch number for the NMNT and FNT according to equation 3.18 for transforms of length (N) 4 and 8 as well as for the AES are shown in Tables 3.3 and 3.4. As mentioned earlier, for maximum diffusion power the branch number is N+1. To illustrate this, consider the case for an input weight equal to 1. For AES in Table 3.3, the output weight is 4, in total giving 5, which represents N+1, and for an input weight of 2 the minimum output weight is 3, in total 5. The same results are obtained for any input weight, indicating that the transform has a branch number equal to 5, signifying that the transform has maximum diffusion power. For the NMNT and FNT it is clear from Table 3.3 that most cases provide maximum diffusion power. However for an input weight equal to 2, the output weight is a minimum of 2, in total giving 4, which means that for this case the transforms have a value lower than N+1, providing less than maximum diffusion. This is especially the case when modifying an even number of active elements and up to N/2 from the total elements N. As the branch number is computed relative to the worst cases, the NMNT and FNT for transforms of length 4 have a branch number equal to 4. The details of all cases, including those that provide low diffusion, are explained in detail in the second method presented in the following section.

Table 3.3: Minimum active bundle for transform length 4

Bundle weight	AES	NMNT	FNT
1	5	5	5
2	5	4	4
3	5	6	5
4	5	5	5

Table 3.4: Minimum active bundle for transform length 8

Bundle weight	NMNT	FNT
1	7	9
2	4	6
3	7	7
4	6	6
5	8	9
6	7	8
7	9	9
8	9	9

3.5.1 Results of the NMNT Analysis

The NMNT has been extensively analysed with different values of modulus (for P = 7, 13, 17, and 19) and transform lengths (N = 4, 8, 16, 32…1024). Tables 3.5 and 3.6 give samples of the results for P = 7, N = 32 and P = 17, N = 256, respectively, while all other results are listed in Appendix A. Note that all calculations are based on bundle level, that is, P-bit. The results of the analysis are outlined below:

1. Modifying single elements at odd locations offers $((N-C)/N)\times 100\%$ diffusion, where C is the number of zero elements in that row in the kernel matrix corresponding to the location of the modified element. Minimum level of diffusion is 75% which achieved for rows that contain a maximum number of zero elements that is N/4, and 100% in rows that contain no zero elements (always 100% for N = 4 or 8).
2. Modifying single elements at even locations offers $((N-2)/N)\times 100\%$ diffusion (100% for N = 4). Accordingly the percentage diffusion improves with larger transform lengths (N). For instance, 87.5% for N = 16, and 98.44% for N = 128, as shown in Figure 3.4.

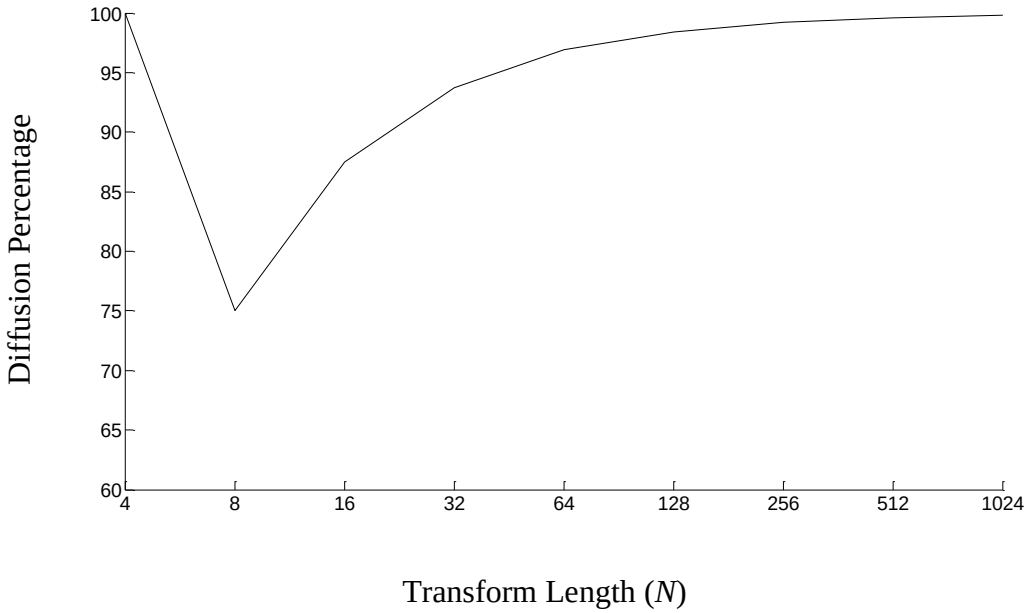


Figure 3.4: Single elements modifications at even locations

Table 3.5: NMNT diffusion for $P = 7$, $M_p = 127$, $N = 32$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $M_p = 0$	Different Modifi. modulo $M_p \neq 0$
Paired elements *i*, *i*+16 *i*-even (x = 1)	Odd	43.8-50%	25-50%	68.8-100%
	Even	12.5- 50%	25-50%	56.3-100%
	All	6.25%	37.5-50%	87.5-100%
Paired elements *i*, *i*+16 *i*-odd (x = 1)	Odd	37.5-50%	25-50%	62.5-100%
	Even	12.5-43.8%	25-50%	50-100%
	All	6.25%	31.3-50%	81.3-100%
Paired elements *i*, *i*+16 *i*-mix (x = 1)	Odd	37.5-50%	34.4-50%	71.9-100%
	Even	18.8-50%	34.4-50%	81.3-100%
	All	3.125%	43.8-50%	90.6-100%
Paired elements *i*, *i*+8 *i*-even (x = 2)	Odd	68.75%	56.3-75%	81.3-100%
	Even	62.5-68.8%	62.5-75%	68.8-100%
	All	56.25%	62.5-75%	87.5-100%
Paired elements *i*, *i*+8 *i*-odd (x = 2)	Odd	50-75%	50-75%	62.5-100%
	Even	62.5%	62.5-75%	75-100%
	All	56.25%	62.5-75%	87.5-100%
Paired elements *i*, *i*+8 *i*-mix (x = 2)	Odd	68.8-75%	62.5-75%	84.4-100%
	Even	56.3-71.9%	62.5-75%	84.4-100%
	All	53.125%	65.6-75%	90.6-100%
Unpaired elem. Even	Single	93.8%	93.75%	93.75%
	Odd	93.8%	68.8-93.8%	68.8-100%
	Even	56.3-100%	56.3-93.8%	68.8-100%
Unpaired elem. Odd	Single	75-100%	75-100%	75-100%
	Odd	75-100%	68.8-93.8%	75-100%
	Even	50-93.8%	50-93.8%	50-100%
Unpaired elem. Mix	Odd	71.9-100%	68.8-96.9%	62.5-100%
	Even	56.3-100%	50-96.9%	68.8-100%
i, *i*+16, random for others	Odd	84.4-100%	87.5-100%	87.5-100%
	Even	84.4-100%	84.4-100%	87.5-100%

Table 3.6: NMNT diffusion for $P = 17$, $M_p = 131071$, $N = 256$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $M_p = 0$	Different Modifi. modulo $M_p \neq 0$
Paired elements *i*, *i*+128 *i*-even (x = 1)	Odd	49.2-50%	48.4-50%	97.7-100%
	Even	21.1-50%	48.4-50%	97.7-100%
	All	0.781%	50%	100%
Paired elements *i*, *i*+128 *i*-odd (x = 1)	Odd	37.5-50%	25-50%	75-100%
	Even	18.8-50%	43.8-50%	87.5-100%
	All	0.781%	50%	100%
Paired elements *i*, *i*+128 *i*-mix (x = 1)	Odd	37.5-50%	49.2-50%	99.2-100%
	Even	24.2-50%	49.2-50%	99.2-100%
	All	0.391%	50%	100%
Paired elements *i*, *i*+64 *i*-even (x = 2)	Odd	73.4-75%	73.4-75%	97.7-100%
	Even	55.5-75%	73.4-75%	97.7-100%
	All	50.78%	75%	100%
Paired elements *i*, *i*+64 *i*-odd (x = 2)	Odd	50-75%	50-75%	87.5-100%
	Even	54.7-75%	73.4-75%	93.8-100%
	All	50.78%	75%	100%
Paired elements *i*, *i*+64 *i*-mix (x = 2)	Odd	68.8-75%	74.2-75%	99.2-100%
	Even	61.7-75%	74.2-75%	78.8-100%
	All	50.39%	74.6-75%	100%
Unpaired elem. Even	Single	99.2%	99.2%	99.2%
	Odd	98.4-100%	87.5-99.2%	97.7-100%
	Even	71.1-100%	74.2-99.2%	97.7-100%
Unpaired elem. Odd	Single	75-100%	75-100%	75-100%
	Odd	75-100%	87.5-99.2%	95.3-100%
	Even	50-100%	50-99.2%	87.5-100%
Unpaired elem. Mix	Odd	87.5-100%	87.5-99.6%	98.4-100%
	Even	86.7-100%	98.4-99.6%	99.2-100%
i, *i*+128, random for others	Odd	99.2-100%	99.2-100%	99.2-100%
	Even	99.2-100%	99.2-100%	99.2-100%

3. Modifying odd numbers of paired elements with:
   a. the same values for each pair, at any location results in 37.5-50% diffusion for x = 1. In general diffusion percentages vary between $\left(\left(2^{x-1}-1\right)/2^{x-1}\right)\times 100\%$ and $\left(\left(2^{x}-1\right)/2^{x}\right)\times 100\%$ for x > 1. The lower bounds approach the upper bounds when modifying pairs at even locations with larger transform length or values of *P*, as shown in Figure 3.5.
   b. different values, with a total sum equal to the modulus for each pair, at any location achieves 25-50% diffusion for x = 1. In general diffusion percentages vary between $\left(\left(2^{x-1}-1\right)/2^{x-1}\right)\times 100\%$ and $\left(\left(2^{x}-1\right)/2^{x}\right)\times 100\%$ for x > 1. The lower bounds approach the upper bounds when modifying pairs at even locations with larger values of *P*.
   c. different values, with a total sum not equal to the modulus for each pair, at any location gives levels of diffusion between 62-100%. The lower bounds approach the upper bounds when modifying pairs at even locations with larger transform lengths or values of *P*.
4. Modifying even numbers of paired elements with:
   a. the same values for each pair, at any location produces diffusion levels between $\left(\left(2^{x+2}-7.25\right)/2^{x+2}\right)\times 100\%$ and $\left(\left(2^{x}-1\right)/2^{x}\right)\times 100\%$. Levels of 9.4-50% for x = 1, increase to 54.7-75% for x = 2, so that larger values of x are better.

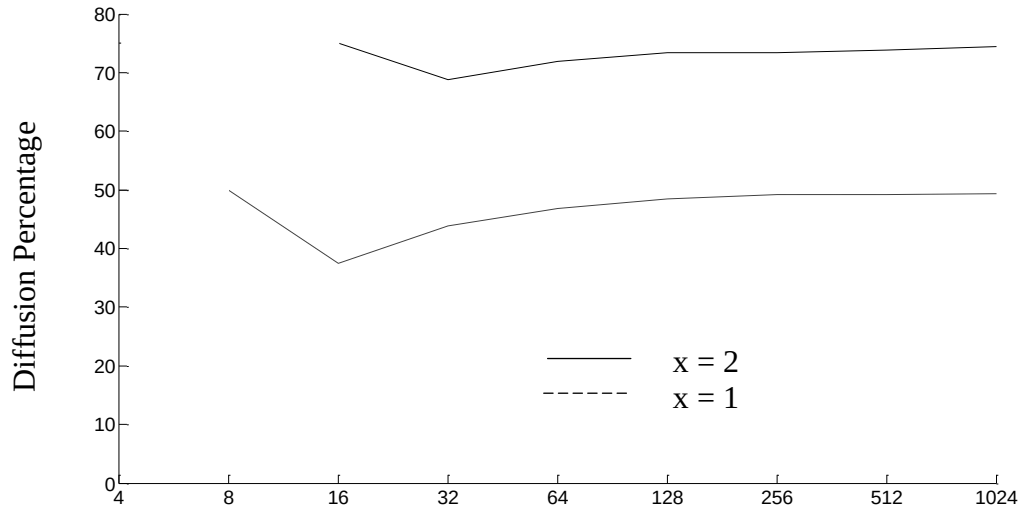


Figure 3.5: Lower bounds for modifying odd number of paired elements at even locations with the same values

b. different values, with a total sum equal to the modulus for each pair, at any location results in $\left(\left(2^x - 1\right)/2^x\right)\times 100\%$ diffusion at upper bounds, and the lower bound varies from half of the upper bound up to the upper bound's values with larger values of P or transform lengths.

c. different values, with a total sum not equal to the modulus for each pair, at any location gives diffusion levels between 50-100%. The lower bound approaches 100% when modifying pairs at even locations with larger transform lengths or values of P.

5. Modifying unpaired elements with the same values for:
   a. odd numbers of elements at even locations produces diffusion levels between $((N - C)/N)\times 100\%$ and 100%, where C equals 2 and can be up to $\log_2 N$.
   b. even numbers of elements at even locations leads diffusion to vary between 56.25-100%; the lower bound improves with larger transform lengths.
   c. odd numbers of elements at odd locations results in diffusion levels between 75-100%, as illustrated in Figure 3.6.
   d. even numbers of elements at odd locations gives diffusion between 50-100%.
6. Modifying unpaired elements with different values, with a total sum equal to the modulus for:
   a. odd numbers of elements at any location produces diffusion between 68% and $((N - 2)/N)\times 100\%$, and the lower bound improves with larger values of P.

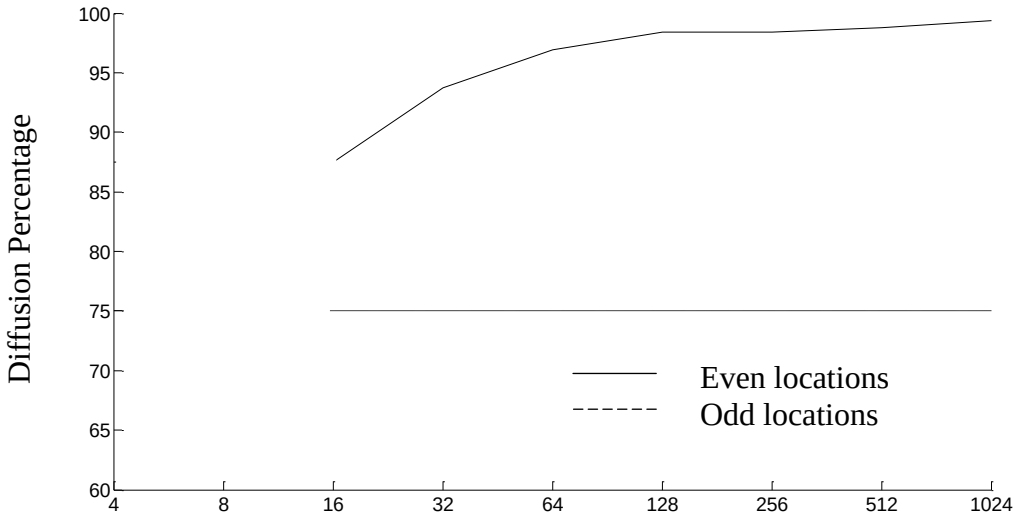


Figure 3.6: Lower bounds for modifying odd numbers of unpaired elements with the same values

b. even numbers of elements at even locations gives diffusion between 56% and ((N - 2)/N)×100%, the lower bound improving with larger transform lengths.
c. even numbers of elements at odd locations leads diffusion to vary between 50% and ((N - 2)/N)×100%, as illustrated in Figure 3.7.

7. Modifying unpaired elements with different values, with a total sum not equal to the modulus for:
   a. odd numbers of elements at any location results in diffusion between 75-100%, the lower bound improves with larger values of P and transform length.
   b. even numbers of elements at any location gives diffusion between 50-100%; the lower bound improves with larger values of P and transform length.
8. Modifying any number of paired elements with any values at any location and with all other elements modified randomly, produces in general levels of diffusion over 75%. On average these are between ((N - 2)/N)×100% and 100% for larger values of P and transform length, as shown in Figure 3.8.
9. Modifying all input elements with the same value, which is equivalent to adding a DC value, gives a diffusion percentage $N^{-1}\times 100\%$. As a result, the ratio decreases as N increases. The best ratio is 25% for N = 4, and decreases as N increases (see Figure 3.9).

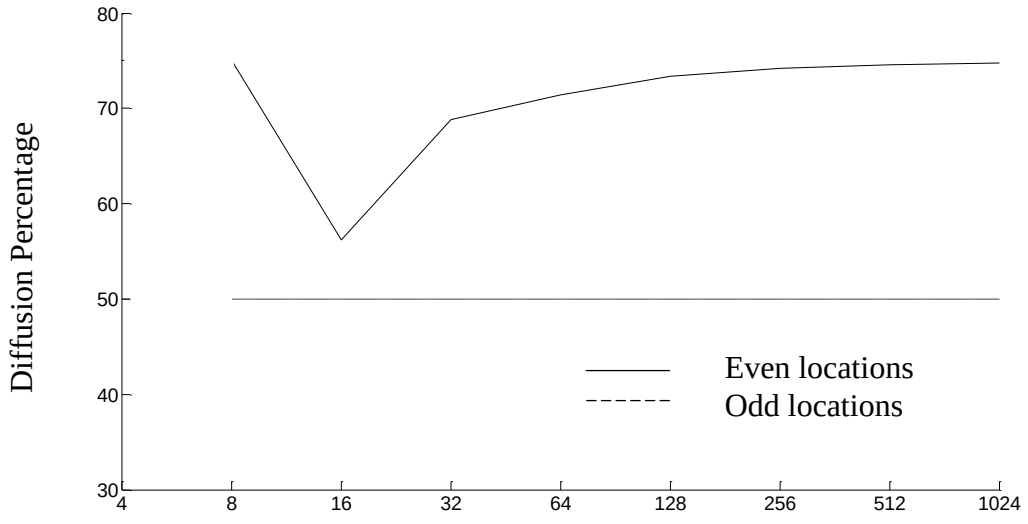


Figure 3.7: Lower bounds for modifying even numbers of unpaired elements with different values their sum equal M_p

10. Modifying all even input elements with the same value, or all odd input elements with the same value or different values, with a total sum equal to the modulus for each pair, the diffusion percentage becomes $2 \times N^{-1} \times 100\%$, which is double that in the previous case, as shown in Figure 3.9.

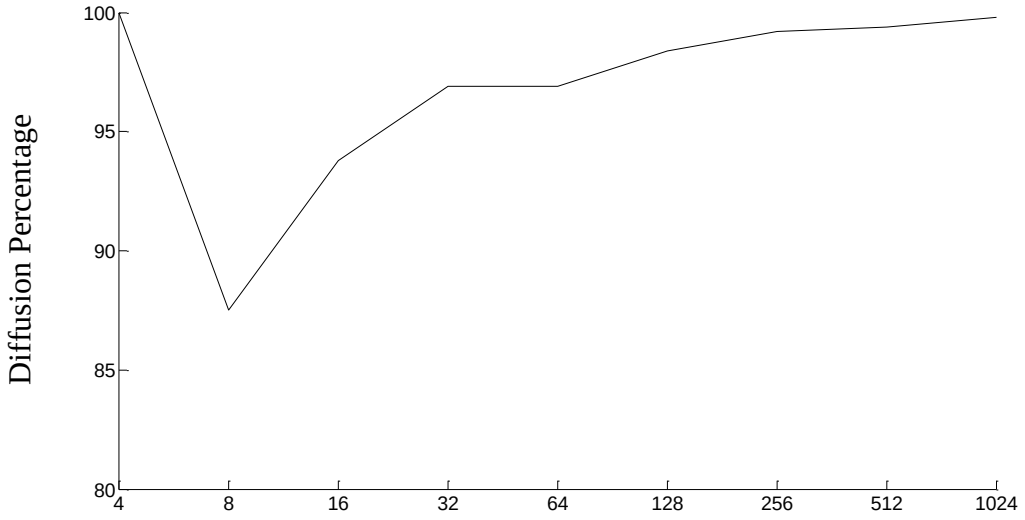


Figure 3.8: Lower bounds for modifying any number of paired elements with any value and location and the remaining elements modified randomly

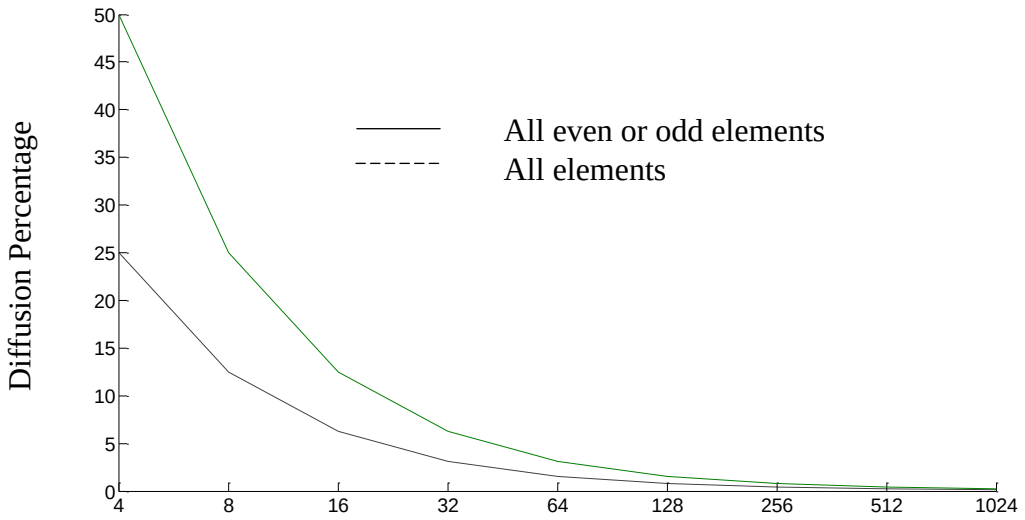


Figure 3.9: All elements and all even/odd elements modification with the same value

From the above results, most cases provide levels of diffusion over than or equal to 50%, except three which can be avoided either by increasing the values of the transform parameters, *P* and/or *N*, or in the design by inserting additional layers to reduce the effect of transform symmetry. The following discusses these cases.

1. Modifying a number of paired elements $\left(i, i+N/2^{x}\right)$up to *N*/2-1 pairs for x = 1 and leaving all other elements unchanged; at any location with the same value for each pair (Figure 3.5), or different values but with a total sum equal to the modulus for each pair (points 3.a, 3.b, 4.a and 4.b above). This would give in the best cases a diffusion level of 50%. The lower bounds approach the upper bounds at 50% when modifying pairs at even locations with a larger modulus and/or transform length. The probability of this case arising can be calculated by assuming that all nonzero changes of all possible values of the bundles belonging to the set M_p-1 are independent and equally likely to occur. The same assumption also applies in the following cases as well as for the FNT. The probability of a single pair arising is $N \times M_p^{-N} \times \left(M_p - 1\right)$, where $N \geq 8$, for example 1.49×10^{-14} for $P = 7$ and $N = 8$, decreasing to 1.83×10^{-131} for $P = 7$ and $N = 64$. The probability increases by increasing the number of modified pairs, a maximum probability is attaining when modifying *N*/4 pairs.
2. Modifying all input elements with the same value (as in point nine above), gives a diffusion level of $N^{-1} \times 100\%$ (Figure 3.9). The probability of this case occurring is $M_p^{-N} \times \left(M_p - 1\right)$. For $P = 7$, $M_p = 127$ and $N = 128$ the probability is thus 6.51×10^{-268}.
3. Modifying all even input elements with the same value; or all odd input elements with the same value (Figure 3.9); or different values, with a total sum equal to the modulus for each pair, with the same values for all modified pairs at odd locations (as in point ten above), gives a diffusion level of $2 \times N^{-1} \times 100\%$. The probability of this case arising is $2M_p^{-N} \times \left(M_p - 1\right) + M_p^{-N} \times \left(M_p - 1\right)^{-N/2+2} \times \left(M_p - 2\right)^{N/4}$. For $P = 7$, $M_p = 127$ and $N = 8$ the probability is 3.74×10^{-15}, decreasing to 1.4×10^{-1185} for $P = 31$ and $N = 128$.

One of the main reasons for the diffusion percentage improving as the transform length increases is that the percentage of the number of zero elements relative to the total (*Zp*) is inversely proportional to *N*, as shown in equation 3.32. Figure 3.10,

graphically represents the percentage of the number of zero elements against the total for different transform lengths.

$$Zp = \left(\frac{\log_2 N - 2}{N} \right) \times 100\% \tag{3.32}$$

Table 3.7 illustrates some of these results using the example of $P = 7$, $M_p = 127$ and $N = 16$. The first two rows represent the initial input to the NMNT and its corresponding transformed output elements. The third row shows the initial input but modifying only one element in the odd position (shadow); its output (row 4) is completely different from that of the initial output (row 2), giving 100% diffusion. The output elements are completely different because the modified input element is located at position $(N/4)+1=5$, where there are no zero elements in the corresponding row in the $\beta(nk)$ matrix. While for row number five, a single element is modified also in the odd position, but the resulting output (row 6) has four elements unchanged; this is because the modified input element is located at position $(N/8)+1=3$, which contains the maximum number of zero elements $N/4=4$ in the corresponding row in the $\beta(nk)$ matrix, giving 75% diffusion. Finally, in row seven a single element is modified in the even location, and the resulting output (row 8) modifies all elements except two. This is because of the two zero elements in the even rows at the $\beta(nk)$ matrix, giving a diffusion level of $((N-2)/N)\times 100\% = 87.5\%$, which is better with larger values of N.

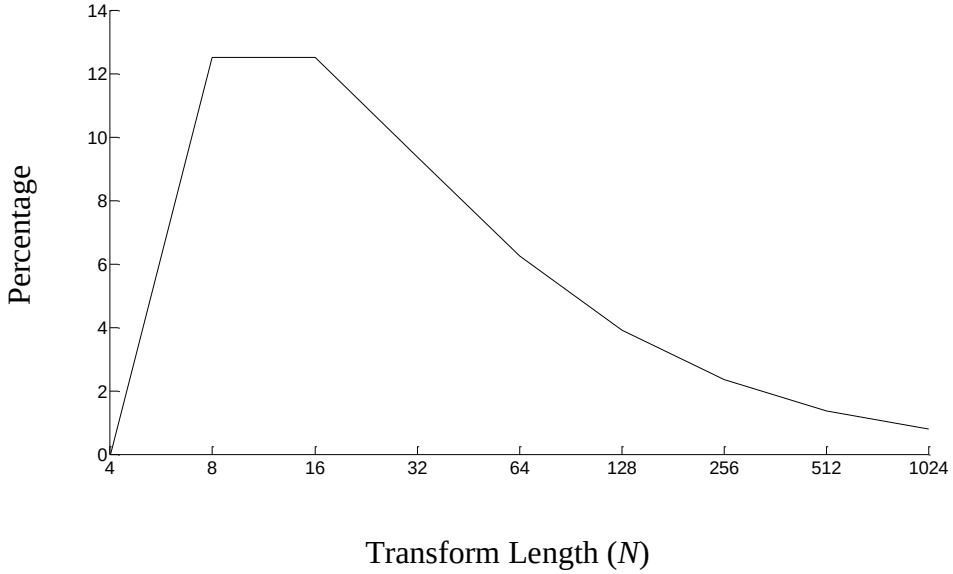


Figure 3.10: Percentages of the number of zero elements relative to the total

Table 3.7: NMNT modification comparisons for $P = 7$, $M_p = 127$, $N = 16$

1	I/P	81	43	121	17	44	119	111	69	75	3	26	38	51	29	107	33
2	O/P	78	72	98	122	50	69	122	66	11	61	9	30	103	48	15	88
3	I/P	81	43	121	17	64	119	111	69	75	3	26	38	51	29	107	33
4	O/P	98	92	78	102	70	89	102	46	31	81	116	10	123	68	122	68
5	I/P	81	43	129	17	44	119	111	69	75	3	26	38	51	29	107	33
6	O/P	86	71	106	122	42	70	114	66	19	60	17	30	95	49	7	88
7	I/P	81	53	121	17	44	119	111	69	75	3	26	38	51	29	107	33
8	O/P	88	3	65	53	60	99	122	36	1	3	42	99	93	18	15	118

3.5.2 Results of the FNT Analysis

The FNT has been also extensively analysed with different values of modulus and transform length. Tables 3.8 and 3.9 give sample results for $t = 3$, $N = 16$ and $t = 4$, $N = 512$, respectively, while all other results are presented in Appendix B. All calculations are based on bundle level. The results are listed below:

1. Modifying single elements at any location gives 100% diffusion.
2. Modifying odd numbers of paired elements at any location with:
   a. the same values for each pair produces nearly $\left(\left(2^x - 1\right)/ 2^x\right)\times 100\%$ diffusion. It is nearly 50% for x = 1, increasing to 75% for x = 2, i.e., improving for higher values of x.
   b. different values, with a total sum equal to the modulus (F_t) for each pair, results in 37.5-50% diffusion for x = 1, increasing to 62.5-75% for x = 2, and improving with higher x. The upper bounds are limited to $\left(\left(2^x - 1\right)/ 2^x\right)\times 100\%$, while the lower bounds vary from 75% up to 100% of the upper bound values with larger t.
   c. different values, with a total sum not equal to the modulus for each pair, leads diffusion to vary between 75-100%. The lower bounds improve with larger values of t and N.
3. Modifying even numbers of pairs of elements at any location with:
   a. the same values for each pair, leads diffusion to vary between $\left(\left(2^{x+2} - 7\right)/ 2^{x+2}\right)\times 100\%$ and $\left(\left(2^x - 1\right)/ 2^x\right)\times 100\%$. For example, it is 12.5-50% for x = 1, increasing to 56.25-75% for x = 2, i.e. better with larger values of x.

Table 3.8: FNT diffusion for $t = 3$, $F_t = 257$, $N = 16$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $F_t = 0$	Different Modifi. modulo $F_t \neq 0$
Paired elements i, i+8 i-even (x = 1)	Odd	50	37.5-50	87.5-100
	Even	25- 37.5	25-50	75-100
	All	12.5	25-50	87.5-100
Paired elements i, i+8 i-odd (x = 1)	Odd	50	37.5-50	75-100
	Even	25-37.5	25-50	75-100
	All	12.5	25-50	87.5-100
Paired elements i, i+8 i-mix (x = 1)	Odd	50	37.5-50	87.5-100
	Even	18.8-50	25-50	81.3-100
	All	6.25	43.8-50	81.3-100
Paired elements i, i+4 i-even (x = 2)	Odd	75	75	75-100
	Even	-	-	-
	All	62.5	62.5-75	75-100
Paired elements i, i+4 i-odd (x = 2)	Odd	75	75	75-100
	Even	-	-	-
	All	62.5	62.5-75	75-100
Paired elements i, i+4 i-mix (x = 2)	Odd	75	68.8-75	87.5-100
	Even	68.75	68.8-75	87.5-100
	All	56.25	62.5-75	87.5-100
Unpaired elem. Even	Single	100	100	100
	Odd	100	75-87.5	75-100
	Even	62.5-87.5	75-87.5	75-100
Unpaired elem. Odd	Single	100	100	100
	Odd	100	75-87.5	75-100
	Even	62.5-87.5	75-87.5	75-100
Unpaired elem. Mix	Odd	100	75-93.8	75-100
	Even	62.5-100	75-93.8	81.3-100
i, i+8, random for others	Odd	87.5-100	81.3-100	87.5-100
	Even	81.3-100	81.3-100	87.5-100

Table 3.9: FNT diffusion for $t = 4$, $F_t = 65537$, $N = 512$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $F_t = 0$	Different Modifi. modulo $F_t \neq 0$
Paired elements *i*, *i*+256 *i*-even (x = 1)	Odd	49.2-50	46.9-50	93.8-100
	Even	24.6-50	49.2-50	99.2-100
	All	0.391	50	100
Paired elements *i*, *i*+256 *i*-odd (x = 1)	Odd	49.2-50	48.4-50	96.9-100
	Even	24.6-50	49.2-50	98.8-100
	All	0.391	50	99.6-100
Paired elements *i*, *i*+256 *i*-mix (x = 1)	Odd	49.6-50	49.6-50	99.6-100
	Even	24.8-50	49.6-50	99.4-100
	All	0.195	50	99.8-100
Paired elements *i*, *i*+128 *i*-even (x = 2)	Odd	74.2-75	73.4-75	98.4-100
	Even	60.9-75	73.4-75	98.4-100
	All	50.391	75	100
Paired elements *i*, *i*+128 *i*-odd (x = 2)	Odd	74.2-75	71.9-75	96.9-100
	Even	60.9-75	73.4-75	98.4-100
	All	50.391	75	99.6-100
Paired elements *i*, *i*+128 *i*-mix (x = 2)	Odd	74.6-75	74.6-75	99.6-100
	Even	62.3-75	74.6-75	99.6-100
	All	50.195	74.8-75	99.8-100
Unpaired elem. Even	Single	100	100	100
	Odd	98.4-100	93.8-99.6	98.4-100
	Even	75-100	75-99.6	99.2-100
Unpaired elem. Odd	Single	100	100	100
	Odd	98.4-100	93.8-99.6	98.4-100
	Even	75-100	75-99.6	99.2-100
Unpaired elem. Mix	Odd	98.4-100	96.9-99.8	98.4-100
	Even	98.2-100	99.2-99.8	99.6-100
i, *i*+256, random for others	Odd	99.4-100	99.6-100	99.4-100
	Even	99.6-100	99.4-100	99.6-100

b. different values, with a total sum equal to the modulus for each pair, gives levels of diffusion varying between $\left(\left(2^{x+2}-6\right)/2^{x+2}\right)\times 100\%$ and $\left(\left(2^{x}-1\right)/2^{x}\right)\times 100\%$. For example, it is 25-50% for x = 1, increasing to 62.5-75% for x = 2. The result is better with larger values of x and the lower bounds increase with larger *t*.

c. different values, with a total sum not equal to F_t for each pair, leads to maximising diffusion levels to between 75-100%. The lower bounds improve with larger values of *t* and transform length.

4. Modifying odd numbers of unpaired elements at any location with:
   a. the same values give diffusion levels of nearly 100%.
   b. different values, with a total sum equal to the modulus, leads to diffusion between 75% and $\left((N-2)/N\right)\times 100\%$. The lower bounds improve with larger values of *t*.
   c. different values, with a total sum not equal to the modulus, results in levels of diffusion between 75-100%. The lower bounds increase with larger values of *t* and transform length.
5. Modifying even numbers of unpaired elements at any location with:
   a. the same values give levels of diffusion between 62.5-100%, while the lower bounds increase up to 75% for larger transform lengths.
   b. different values, with a total sum equal to the modulus, results in diffusion between 75% and $\left((N-2)/N\right)\times 100\%$.
   c. different values, with a total sum not equal to the modulus, increases diffusion to between 75-100%. The lower bounds again improve with larger values of *t*.
6. Modifying any number of paired elements with any value at any location and all other elements modified randomly produces, in general, diffusion levels over 75%, on average lying between $\left((N-2)/N\right)\times 100\%$ and 100% for larger values of *t* and transform length, as shown in Figure 3.11.
7. Modifying all input elements with the same value, which is equivalent to adding a DC value, gives a diffusion percentage of $N^{-1}\times 100\%$ (see Figure 3.9).
8. Modifying all even or all odd input elements with the same value, the diffusion level achieved is $2\times N^{-1}\times 100\%$, which is double the percentage in the preceding case (see Figure 3.9).

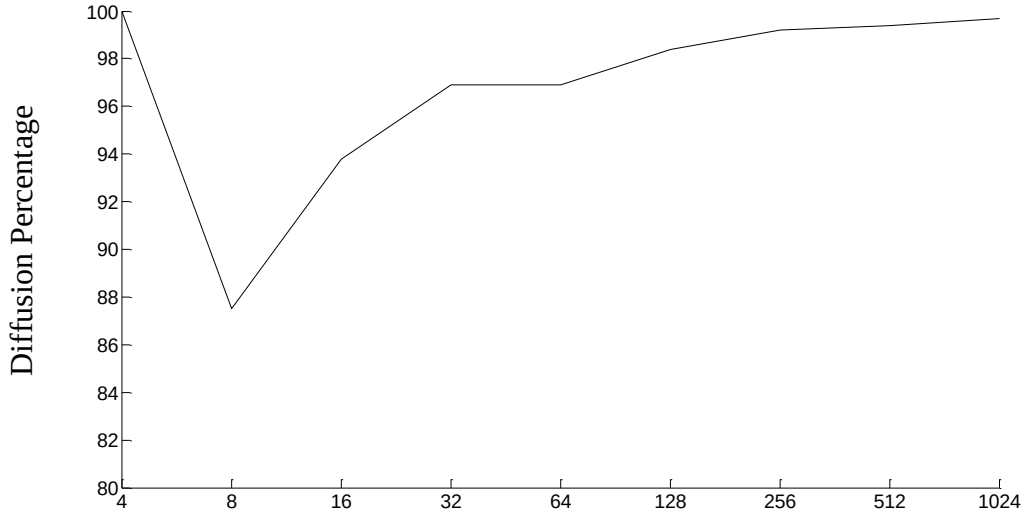


Figure 3.11: Lower bounds for modifying any number of paired elements with any value and location and the remaining elements modified randomly (FNT)

From the above results, most cases have good levels of diffusion at a minimum 50%. The probability of those levels below 50% occurring can be reduced by increasing either values of *t* and/or those of transform length. The cases that provide low levels of diffusion are shown below:

1. When modifying even or odd numbers of paired elements $\left(i, i+N/2^{x}\right)$ for x = 1 with the same values or different values, with their sum equal to the modulus for each pair, and leaving all other elements unchanged (points 2.b, 3.a and 3.b above). The probability of this case arising under the same assumptions as in the NMNT can be determined by considering that all nonzero changes of all possible values of the bundles belonging to the set $F_t - 1$ are independent and equally likely. The probability of a single pair arising is thus $N \times F_t^{-N} \times \left(F_t - 1\right)$. Where $N \geq 8$, for example the probability is 1.08×10^{-16} for $t = 3$ and $N = 8$, decreasing to 9.52×10^{-151} for $t = 3$ and $N = 64$. The probability increases by increasing the number of modified pairs, a maximum probability is reached when modifying $N/4$ pairs.
2. When modifying all input elements with the same value (point 7 above), the diffusion level achieved is $N^{-1} \times 100\%$. The probability of this case occurring is $F_t^{-N} \times \left(F_t - 1\right)$. For example, for $t = 3$ and $N = 128$ the probability is 8.65×10^{-307}.

3. When modifying all even or all odd input elements with the same value (point 8 above), the level of diffusion becomes $2 \times N^{-1} \times 100\%$. The probability of this case arising is $2F_t^{-N} \times (F_t - 1)$. For t = 3 and N = 8 it is 2.69×10^{-17}, decreasing to 1.58×10^{-614} for t = 4 and N = 128.

Table 3.10 explains some of these results using the example of t = 3, F_t = 257 and N = 16. The first two rows stand for the initial input to the FNT and its corresponding output. The third row represents the initial input but modifying only one element in the even location (shadowed); its output (row 4) is completely different from that of the initial output (row 2), giving 100% diffusion. The same level of diffusion is gained when modifying a single element in any location whether even or odd. In the fifth row the initial input is modified with an odd number of unpaired elements at mix locations (shadowed). Its output (row 6) is completely different from that of the initial output, giving 100% diffusion. Finally, rows 7-10 explain the case of modifying singles and couples of pairs $(i, i + N/2)$ of elements. The first example modifies the paired elements with different values whose sum is equal to F_t, while the second modifies each pair of elements with the same value different from that of the other. The resulting levels of diffusion are 50%.

Table 3.10: FNT modification comparisons for t = 3, F_t = 257, N = 16

1	I/P	77	16	65	112	84	37	19	53	2	107	41	72	35	22	96	87
2	O/P	154	3	61	44	18	184	7	83	170	18	85	223	193	147	201	155
3	I/P	77	16	65	108	84	37	19	53	2	107	41	72	35	22	96	87
4	O/P	150	228	62	52	82	182	248	212	174	50	84	215	129	149	217	26
5	I/P	78	116	45	114	84	37	19	53	4	107	41	74	35	22	96	87
6	O/P	235	124	142	79	97	124	46	29	51	253	137	200	156	112	33	201
7	I/P	77	216	65	112	84	37	19	53	2	164	41	72	35	22	96	87
8	O/P	154	32	61	160	18	134	7	140	170	246	85	107	193	197	201	98
9	I/P	79	16	65	117	84	37	19	53	4	107	41	77	35	22	96	87
10	O/P	168	3	191	44	119	184	51	83	164	18	220	223	100	147	165	155

3.5.3 Summary of results for NMNT and FNT

The first class of results includes cases that provide good diffusion power of a minimum of 50%, and the second class lists cases that exhibit low diffusion power up to 50%. As mentioned above, all calculations are based on a bundle level:

1. Cases that provide good diffusion power
   a. Modifying single elements

      NMNT: Modifying single elements at odd locations, gives levels of diffusion between 75-100%, depending on the number of elements with zero value at that row in the kernel matrix corresponding to the position of the modifying element. Modifying single elements at even locations produces $((N-2)/N)\times 100\%$ diffusion (100% for N = 4), and the diffusion level increasing with larger transform length, for instance, 87.5% for N = 16, and 98.44% for N = 128.

      FNT: Modifying single elements at any locations achieves 100% diffusion.

   b. Modifying paired elements

      Modifying any numbers of paired elements $\left(i, i+N/2^{x}\right)$ at any location with any value and x > 1, results in a minimum diffusion level of 50%.

      NMNT: The lower bounds increase with larger values of x and modulus, as well as improving for larger transform lengths for elements modified at even locations.

      FNT: The lower bound increases with larger values of x, modulus and transform length.

   c. Modifying unpaired elements

      NMNT: Modifying even numbers of unpaired elements at any location with any value gives diffusion levels between 50-100%, increasing to 68-100% when modifying odd numbers of unpaired elements.

      FNT: Modifying even numbers of unpaired elements at any location with any value leads diffusion to vary between 62-100%, increasing to 75-100% when modifying odd numbers of unpaired elements.

      For both cases the lower bounds increase in most cases with larger values of modulus and/or transform length.

   d. Modifying all elements

      NMNT and FNT: Modifying any number of paired elements with any value at any location and all other elements modified randomly offers levels of diffusion between 50-100%. In general levels exceeding 75% increase with larger values of modulus and transform length.

2. Cases that produce low diffusion power
   a. Modifying paired elements

Modifying any number of paired elements $(i, i+N/2^x)$ for just x = 1 and leaving all other elements unchanged at any location with the same values for each pair, or different values with a total sum equal to the modulus for each pair, gives diffusion levels in the best cases of 50%. The probability of a single pair arising is $N \times Modulus^{-N} \times (Modulus - 1)$, it is increased when the number of modified pairs is increasing, maximum is achieved when modifying $N/4$ pairs.

NMNT: The lower bounds approach the upper bounds at 50% when modifying pairs at even locations with larger values of modulus and/or transform length.

FNT: The lower bounds increase with larger values of modulus and/or transform length.

For both transforms, the diffusion power improves when x increase to greater than 1.

b. Modifying all input elements

NMNT and FNT: modifying all input elements with the same value gives a diffusion level of $N^{-1} \times 100\%$. The probability of this case occurring is $Modulus^{-N} \times (Modulus - 1)$.

c. Modifying all even input elements or all odd input elements

NMNT: modifying all even input elements with the same value, or all odd input elements with the same value, or different values with their sum equal to the modulus, with the same values for all modified pairs, the diffusion percentage achieved is $2 \times N^{-1} \times 100\%$. The probability of this case appearing is $2M_p{}^{-N} \times (M_P - 1) + M_p{}^{-N} \times (M_p - 1)^{-N/2+2} \times (M_p - 2)^{N/4}$.

FNT: modifying these elements with the same value, the level of diffusion resulting is $2 \times N^{-1} \times 100\%$. The probability of this case is $2F_t^{-N} \times (F_t - 1)$.

The probabilities of occurrence of the last two cases (points b and c) can be reduced by increasing the values of modulus and/or the transform length.

3.6 Discussion

The diffusion power of both the NMNT and FNT have been considered in this chapter using two different techniques in order to evaluate their suitability for secure applications.

The branch number of the transforms which is discussed on the first technique indicates that the transforms can provide maximum diffusion power in most cases, with the exception of mostly for even input weight and up to the half of the transform length. However, the analysis from the second technique explains deeply this case which obviously arises with very low probability when modifying pairs of elements $\left(i, i+N/2^{x}\right)$ (just for x = 1) with only the same value or different values with their sum equal to the modulus.

The results achieved with the second technique are classified into two classes. In the cases providing good diffusion power over 50%, the lower bounds are further increased with increasing values of modulus and/or transform length. However, cases where the diffusion power provides is less than 50% can be avoided by involving other layers in the design, or alternatively the probability of those cases arising can be reduced by increasing the values of the modulus and/or transform length. In general, increasing the modulus and/or transform length is beneficial as it either improves the diffusion power or reduces the probability of such cases arising. It is important to ensure that diffusion power improves with larger block sizes or key lengths, which may be achieved by increasing the modulus and/or the transform length. This will facilitate the design by providing the possibility of changing the block size or key length so as to achieve the required level of security without the need to alter the algorithm itself. At the same time the number of rounds required can be fixed for different sizes, which would support the compatibility of the algorithm on different platforms.

3.7 Conclusions

The diffusion power of the NMNT and FNT is evaluated in this chapter. Although the results demonstrate that in certain cases the transforms provide lower diffusion than the maximum possible, due to matrix symmetry which could be avoided in the design, it can be concluded that the transforms have many features making them suitable in the design of a secure cryptosystem. Advantages include: parameterisation, providing flexibility to change the block size and key length to meet the required level of security; and sensitivity, the diffusion power has been proven in general it is good [97-99]; having a long transform length (of the power of two); these operations are performed without the errors that normally arise through using floating-point operations; finally, they are suitable for real time implementation as fast algorithms can be adapted to them to speed up processing.

Chapter 4

Substitution Box Analysis

The previous chapter has explained the details of the transforms responsible for providing diffusion in the design. This chapter turns to the function of confusion, which is achieved in the design through the use of the S-box. The methods of S-box construction are briefly described, along with the mechanism used to analyse its non-linear properties. The chapter is organised as follows: section 4.2 presents different ways of building an S-box. The methods used to analyse the non-linear properties of an S-box are explained in section 4.3. The non-linear properties of S-boxes employed in the previous and current standard algorithms are then analysed in sections 4.4 and 4.5, respectively. In the subsequent two sections, new S-boxes are generated based on the AES S-box. Finally, the conclusions of the chapter are outlined in section 4.8.

4.1 Introduction

The security of most block cipher cryptosystems relies heavily on the strength of the S-boxes involved. Therefore an S-box with strong non-linear properties is essential to ensure high resilience against attacks, especially those deploying differential and linear cryptanalysis and their variants. The S-box is usually preceded with $n \times m$ dimensions, where n is the number of input bits to the S-box and m is the number of output bits, and

n and *m* may be equal or unequal. For instance, the DES algorithm [2] has eight 6×4 S-boxes, the AES [16] uses one 8×8 S-box and Blowfish [17] has four 8×32 S-boxes. A larger S-box is more resistant to attacks, but at the same time a larger look-up table is needed and the size of the look-up table increases exponentially with larger values of *n*. Hence, for practical purposes in implementation, the value of *n* is limited to between 8 and 10 [12].

4.2 Methods of S-box Construction

S-boxes can be constructed using several different techniques which may be summarised as follows:

1. Random / Random and key-dependent

   The elements of such an S-box are chosen completely at random. Smaller S-boxes can be insecure and vulnerable to attacks. One of the methods used to make the appearance of the S-box random is the key-dependent scheme, which alters the contents using a key [12]. The Blowfish [17] is one of the prominent algorithms using this approach to generate S-boxes.
2. Random followed by testing

   In this scheme, the S-box entries are also chosen randomly. However, the elements are tested and only accepted if they pass certain criteria. This scheme has been used in many algorithms, such as CAST [19, 20], DES [2], Mars [23], Serpent [27], and Twofish [29].
3. Algebraic constructions

   S-boxes generated based on algebraic constructions offer more resistance to differential and linear cryptanalysis, because good non-linear properties can be achieved. Various techniques have been proposed for algebraic construction; for example, a combination of an inverse function in $GF(2^n)$ and an affine transformation, as with the AES [3], Camellia [18], Shark [37] and Square [38] algorithms.

   Although, S-boxes carefully generated using algebraic construction are secure against differential and linear cryptanalysis they are, to some extent, subject to algebraic attacks if weak functions are used.

To build a reliable secure cryptosystem, it is important to be certain that all of the S-boxes involved in the design have good non-linear properties, meaning that it should not be possible to derive an S-box output as a linear function of the input. In addition, the designed S-box should actively responses to the SAC.

4.3 Analysis of Non-linear Properties of an S-box

The S-box plays a crucial role in the overall security of an algorithm, and hence it is important to evaluate the strength of all of the S-boxes involved in the design by analysing and enumerating their non-linear properties. In order to achieve this, it is necessary to calculate the DPP_{max} and the IOC_{max} as well as the robustness of the S-box. These calculations are needed to verify the resistance of the algorithm to differential and linear cryptanalysis and other related attacks, which are explained in detail in chapter 6. To illustrate the procedures used to calculate the required values, it is useful to start by considering a small S-box for the purposes of simplicity. For instance, the 3×3 S-box shown in Table 4.1 maps 3 input bits to 3 output bits, the procedures used with it are explained in the following sections [100].

Table 4.1: 3×3 S-box representation

I/P		O/P	
000	0	011	3
001	1	010	2
010	2	111	7
011	3	101	5
100	4	001	1
101	5	110	6
110	6	000	0
111	7	100	4

4.3.1 Maximum Difference Propagation Probability

The calculation of the DPP_{max} is achieved by first building the XOR distribution table. This is not a straightforward process, and passes through a number of stages as follows:

1. Calculate the input XOR difference, by XORing all possible input pairs ($2^3 \times 2^3$), as shown in Tables 4.2 and 4.3 for binary and decimal representation, respectively.
2. Separate input pairs into groups according to their input XOR values (columns 1 and 2, Table 4.4).
3. Map all input pairs through the S-box to derive their corresponding output pairs (column 3).
4. Calculate the output XOR difference, by XORing all output pairs (column 4).

Table 4.2: Input XOR difference (binary)

I/P XOR Difference	000	001	010	011	100	101	110	111
000	000	001	010	011	100	101	110	111
001	001	000	011	010	101	100	111	110
010	010	011	000	001	110	111	100	101
011	011	010	001	000	111	110	101	100
100	100	101	110	111	000	001	010	011
101	101	100	111	110	001	000	011	010
110	110	111	100	101	010	011	000	001
111	111	110	101	100	011	010	001	000

Table 4.3: Input XOR difference (decimal)

I/P XOR Difference	0 $\bar{x}_2\bar{x}_1\bar{x}_0$	1 $\bar{x}_2\bar{x}_1x_0$	2 $\bar{x}_2x_1\bar{x}_0$	3 $\bar{x}_2x_1x_0$	4 $x_2\bar{x}_1\bar{x}_0$	5 $x_2\bar{x}_1x_0$	6 $x_2x_1\bar{x}_0$	7 $x_2x_1x_0$
0	0	1	2	3	4	5	6	7
1	1	0	3	2	5	4	7	6
2	2	3	0	1	6	7	4	5
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	7	6	1	0	3	2
6	6	7	4	5	2	3	0	1
7	7	6	5	4	3	2	1	0

Table 4.4: Output XOR difference

I/P XOR	I/P Pairs	O/P Pairs	O/P XOR
0	0,0	3,3	0
	1,1	2,2	0
	2,2	7,7	0
	3,3	5,5	0
	4,4	1,1	0
	5,5	6,6	0
	6,6	0,0	0
	7,7	4,4	0
1	(0,1) , (1,0)	(3,2) , (2,3)	1
	(2,3) , (3,2)	(7,5) , (5,7)	2
	(4,5) , (5,4)	(1,6) , (6,1)	7
	(6,7) , (7,6)	(0,4) , (4,0)	4
2	(0,2) , (2,0)	(3,7) , (7,3)	4
	(1,3) , (3,1)	(2,5) , (5,2)	7
	(4,6) , (6,4)	(1,0) , (0,1)	1
	(5,7) , (7,5)	(6,4) , (4,6)	2
3	(0,3) , (3,0)	(3,5) , (5,3)	6
	(1,2) , (2,1)	(2,7) , (7,2)	5
	(4,7) , (7,4)	(1,4) , (4,1)	5
	(5,6) , (6,5)	(6,0) , (0,6)	6
4	(0,4) , (4,0)	(3,1) , (1,3)	2
	(1,5) , (5,1)	(2,6) , (6,2)	4
	(2,6) , (6,2)	(7,0) , (0,7)	7
	(3,7) , (7,3)	(5,4) , (4,5)	1
5	(0,5) , (5,0)	(3,6) , (6,3)	5
	(1,4) , (4,1)	(2,1) , (1,2)	3
	(2,7) , (7,2)	(7,4) , (4,7)	3
	(3,6) , (6,3)	(5,0) , (0,5)	5
6	(0,6) , (6,0)	(3,0) , (0,3)	3
	(1,7) , (7,1)	(2,4) , (4,2)	6
	(2,4) , (4,2)	(7,1) , (1,7)	6
	(3,5) , (5,3)	(5,6) , (6,5)	3
7	(0,7) , (7,0)	(3,4) , (4,3)	7
	(1,6) , (6,1)	(2,0) , (0,2)	2
	(2,5) , (5,2)	(7,6) , (6,7)	1
	(3,4) , (4,3)	(5,1) , (1,5)	4

5. Count the number of pairs that give different output XORs for corresponding input XORs. The complete table is shown in Table 4.5, which represents the input-output XOR distribution table or, in short the XOR distribution table, which is also referred to as the difference distribution table.
6. Compute the DPP_{max} by dividing the maximum value in the XOR distribution table generated (ignoring the value in the top left corner) by the total elements (2^n), where *n* is the element length in bits. Hence, the DPP_{max} for this S-box is $4/8 = 2^{-1}$.

Table 4.5: XOR distribution table

I/P-O/P XOR Difference	0 $\bar{y}_2\bar{y}_1\bar{y}_0$	1 $\bar{y}_2\bar{y}_1y_0$	2 $\bar{y}_2y_1\bar{y}_0$	3 $\bar{y}_2y_1y_0$	4 $y_2\bar{y}_1\bar{y}_0$	5 $y_2\bar{y}_1y_0$	6 $y_2y_1\bar{y}_0$	7 $y_2y_1y_0$
0	8	0	0	0	0	0	0	0
1	0	2	2	0	2	0	0	2
2	0	2	2	0	2	0	0	2
3	0	0	0	0	0	4	4	0
4	0	2	2	0	2	0	0	2
5	0	0	0	4	0	4	0	0
6	0	0	0	4	0	0	4	0
7	0	2	2	0	2	0	0	2

From Table 4.5 the probability of the occurrence of any particular input-output difference (XOR) can be determined. For example, an input difference of 2 may result in an output difference 4 with a probability of 2/8. While, input difference 2 may NOT cause output difference 3 as its probability is 0. For an ideally randomising S-box, the probability of any particular output difference occurring with any particular input difference is $1/2^n$ [100], which is mathematically not possible. For differential cryptanalysis to be possible, the cryptanalyst must search for a particular input difference that gives a particular output difference with a high probability.

All XOR distribution tables share the following properties. Firstly, the sum of elements in every row is equal to the total input elements (for S-boxes that have the same number of input and output bits, the sum in every column is equal to the total as well). Secondly, the values of all elements are even, providing that a pair of (x, x´) element values has the same difference as a pair of (x´, x) element values. Finally, for an input difference of zero, the resulting output difference can only be zero.

It is worth mentioning that if the elements of an S-box or just their order are changed, the DPP_{max} and overall XOR distribution table will be changed accordingly. For example, if the first two output elements of the S-box in Table 4.1 are swapped, i.e. 2 become the output for an input of 0 and 3 the output for an input of 1, the XOR distribution table for this modified S-box is shown in Table 4.6. The maximum value in the table is 2, giving a DPP_{max} of $2/8 = 2^{-2}$, which is less than the maximum value for the original S-box and therefore has improved the S-box. Thus caution should be exercised when changing, for example, the appearance of the elements of an S-box or their order since this could degrade its non-linear properties.

Table 4.6: XOR distribution table (modified S-box)

I/P-O/P XOR Difference	0	1	2	3	4	5	6	7
0	8	0	0	0	0	0	0	0
1	0	2	2	0	2	0	0	2
2	0	2	2	0	0	2	2	0
3	0	0	0	0	2	2	2	2
4	0	2	0	2	0	2	0	2
5	0	0	2	2	2	2	0	0
6	0	0	2	2	0	0	2	2
7	0	2	0	2	2	0	2	0

4.3.2 Maximum Input-Output Correlation

To examine the linear vulnerability of an S-box, the probability bias or input-output correlation for each linear approximation represented in equation 4.1 needs to be computed, where X_i represents the i^{th} bit of the input X to the S-box and Y_j is the j^{th} bit of the output Y from the S-box, respectively, and $\oplus$ denotes bitwise XOR operation.

$$X_{i1} \oplus X_{i2} \oplus \ldots \oplus X_{in} \oplus Y_{j1} \oplus Y_{j2} \oplus \ldots \oplus Y_{jm} = 0 \qquad (4.1)$$

All possible combinations of input and output bits in the S-box shown in Table 4.1 are represented in Tables 4.7 and 4.8, respectively. The probability bias is calculated by counting the number of matches between input and output combinations over all 2^n minus 0.5 for each linear expression, n is the element length in bits. For instance,

considering all values for the linear expression $X_0 \oplus Y_0 = 0$, which is equivalent to $X_0 = Y_0$, the number of matches between the input and output combinations can be counted. There are precisely 2 cases out of 8 that hold the above expression true, and so the probability bias for this expression is $\frac{2}{8} - \frac{1}{2} = -\frac{1}{4}$. Similarly, for the linear expression $X_0 = Y_1 \oplus Y_0$, the probability bias is $\frac{6}{8} - \frac{1}{2} = \frac{1}{4}$, whereas for the linear expression $X_0 = Y_2 \oplus Y_0$ the probability bias is $\frac{4}{8} - \frac{1}{2} = 0$.

Table 4.7: Possible S-box input combinations

Initial I/P			All possible I/P combinations						
x_2	x_1	x_0	x_0	x_1	$x_1 \oplus x_0$	x_2	$x_2 \oplus x_0$	$x_2 \oplus x_1$	$x_2 \oplus x_1 \oplus x_0$
0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1
0	1	1	1	1	0	0	1	1	0
1	0	0	0	0	0	1	1	1	1
1	0	1	1	0	1	1	0	1	0
1	1	0	0	1	1	1	1	0	0
1	1	1	1	1	0	1	0	0	1

Table 4.8: Possible S-box output combinations

Initial O/P			All possible O/P combinations						
y_2	y_1	y_0	y_0	y_1	$y_1 \oplus y_0$	y_2	$y_2 \oplus y_0$	$y_2 \oplus y_1$	$y_2 \oplus y_1 \oplus y_0$
0	1	1	1	1	0	0	1	1	0
0	1	0	0	1	1	0	0	1	1
1	1	1	1	1	0	1	0	0	1
1	0	1	1	0	1	1	0	1	0
0	0	1	1	0	1	0	1	0	1
1	1	0	0	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	1	1	1

A complete input/output linear approximation is shown in Table 4.9, where the probability bias of any particular linear combination of input and output bits represents the value in the table over 2^n (total number of elements $2^3 = 8$). Each value represents the number of matches in the linear expression between the input and output combinations minus 2^{n-1} (where minus $2^{n-1} = 4$ followed by dividing over all is equivalent to subtraction of 0.5). The maximum value in Table 4.9 (by ignoring the value in the top left corner) is 4, indicating that the maximum bias probability is $4/8=2^{-1}$. For linear cryptanalysis to be possible, the cryptanalyst searches for a linear expression that holds a linear probability further away from 0.5, which is the probability for random values. On the other hand, the correlation between input and output combinations is equal to their inner products divided by their norms [3]. Where the norm is equal to the square root of the domain size, i.e. $2^{n/2}$. This can be computed by summing the products of all inner values over the total for each particular input-output combination, replacing each 0 with 1 and each 1 with -1. The input-output correlation is exactly twice the bias.

It should be noted that, in the linear approximation table, the sum of elements in every column is equal to the absolute value of the half of the total input elements, i.e., 2^{n-1} (for S-boxes that have the same number of input and output bits, the sum in every row is equal to the total as well) and the top left value is always 2^{n-1} providing a probability bias of 0.5, which represents the case of a linear combination with no input-output bits.

Table 4.9: Linear approximation table

		Output Sum							
		0	1	2	3	4	5	6	7
Input Sum	0	4	0	0	0	0	0	0	0
	1	0	-2	0	2	2	0	2	0
	2	0	0	-2	-2	2	-2	0	0
	3	0	-2	2	0	0	-2	-2	0
	4	0	-2	-2	0	0	2	-2	0
	5	0	0	-2	2	-2	-2	0	0
	6	0	2	0	2	2	0	-2	0
	7	0	0	0	0	0	0	0	4

As in the previous case, changing the values of elements in an S-box, or just their order, will generally change the maximum bias probability, IOC_{max} and overall linear approximation table. For example, if the first two output elements of the S-box in Table 4.1 are swapped, so that 2 become the output for input 0 and 3 is the output for input 1, the linear approximation table for this modified S-box is shown in Table 4.10. The maximum value is |2|, giving a maximum bias probability of $2/8 = 2^{-2}$, which is less than the maximum value for the original S-box. Thus, again, caution should be exercised when changing the order or values of elements in an S-box as this could degrade its non-linear properties.

Table 4.10: Linear approximation table for modified S-box

		Output Sum							
		0	1	2	3	4	5	6	7
Input Sum	0	4	0	0	0	0	0	0	0
	1	0	0	0	0	2	2	2	-2
	2	0	0	-2	-2	2	-2	0	0
	3	0	0	2	-2	0	0	-2	-2
	4	0	-2	-2	0	0	2	-2	0
	5	0	2	-2	0	-2	0	0	-2
	6	0	2	0	2	2	0	-2	0
	7	0	2	0	-2	0	2	0	2

4.3.3 Robustness of the S-box

This is a measure of the resistance of an S-box to differential cryptanalysis. The robustness, R_s, is defined in equation 4.2 [101], and is based on the two variables *nNZ* and D_{max} which are derived from the XOR distribution table: *nNZ* is the number of non-zero elements in the first column (excluding the first element), and D_{max} is the maximum value in the table. The value of *nNZ* is very important, as it represents the number of cases that give no output change for an input change. An S-box with a higher value of R_s is more resistant to differential cryptanalysis [101].

$$R_s = \left(1 - \frac{nNZ}{2^n}\right)\left(1 - \frac{D_{max}}{2^n}\right) \tag{4.2}$$

where *n* is the number input bits to the S-box

4.4 Analysis of the DES S-boxes

The DES algorithm consists of a set of eight different 6×4 S-boxes. Mapping 6 input bits and produces 4 output bits, each S-box is a two dimensional array of 64 elements with 4-bit each, distributed in 4 rows and 16 columns. The mapping is achieved by combining the first and last input bits to identify the row number and then the inner input bits are used to select the column number. These S-boxes are shown in Table 4.11. For example, a decimal input of 48, which is equivalent to 110000 in binary form, is mapped to the decimal 15 through the first S-box (S1), where the row number is 10 in base 2 from the first and last input bits and the column number is 1000 in base 2 from the middle input bits.

4.4.1 Analysis of the Non-linear Properties of DES S-boxes

The non-linear properties of all DES S-boxes can be examined by following the same procedures described in section 4.3. Tables 4.12 and 4.13 are samples for the XOR distribution and linear approximation tables for the first S-box (S1), respectively.

The maximum values found in the XOR distribution and linear approximation tables for all DES S-boxes are summarised in Table 4.14. The DPP_{max} is the same for all of the S-boxes, at $16/64 = 2^{-2}$, and the maximum bias probability of -20/64 is found in the fifth S-box (S5).

4.4.2 Robustness of DES S-boxes

The values of robustness for all of the DES S-boxes are calculated by applying equation 4.2 and are shown in Table 4.15. The results indicate that all of the S-boxes are approximately have the same robustness, and that S4 can be more resistance to differential cryptanalysis.

Table 4.11: DES S-boxes

		Column Number															
	R	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S-box 1	0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S-box 2	0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S-box 3	0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S-box 4	0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S-box 5	0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S-box 6	0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S-box 7	0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S-box 8	0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Table 4.12: DES - XOR distribution table (S1)

Input Difference \ Output Difference	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	6	0	2	4	4	0	10	12	4	10	6	2	4
2	0	0	0	8	0	4	4	4	0	6	8	6	12	6	4	2
3	14	4	2	2	10	6	4	2	6	4	4	0	2	2	2	0
4	0	0	0	6	0	10	10	6	0	4	6	4	2	8	6	2
5	4	8	6	2	2	4	4	2	0	4	4	0	12	2	4	6
6	0	4	2	4	8	2	6	2	8	4	4	2	4	2	0	12
7	2	4	10	4	0	4	8	4	2	4	8	2	2	2	4	4
8	0	0	0	12	0	8	8	4	0	6	2	8	8	2	2	4
9	10	2	4	0	2	4	6	0	2	2	8	0	10	0	2	12
10	0	8	6	2	2	8	6	0	6	4	6	0	4	0	2	10
11	2	4	0	10	2	2	4	0	2	6	2	6	6	4	2	12
12	0	0	0	8	0	6	6	0	0	6	6	4	6	6	14	2
13	6	6	4	8	4	8	2	6	0	6	4	6	0	2	0	2
14	0	4	8	8	6	6	4	0	6	6	4	0	0	4	0	8
15	2	0	2	4	4	6	4	2	4	8	2	2	2	6	8	8
16	0	0	0	0	0	0	2	14	0	6	6	12	4	6	8	6
17	6	8	2	4	6	4	8	6	4	0	6	6	0	4	0	0
18	0	8	4	2	6	6	4	6	6	4	2	6	6	0	4	0
19	2	4	4	6	2	0	4	6	2	0	6	8	4	6	4	6
20	0	8	8	0	10	0	4	2	8	2	2	4	4	8	4	0
21	0	4	6	4	2	2	4	10	6	2	0	10	0	4	6	4
22	0	8	10	8	0	2	2	6	10	2	0	2	0	6	2	6
23	4	4	6	0	10	6	0	2	4	4	4	6	6	6	2	0
24	0	6	6	0	8	4	2	2	2	4	6	8	6	6	2	2
25	2	6	2	4	0	8	4	6	10	4	0	4	2	8	4	0
26	0	6	4	0	4	6	6	6	6	2	2	0	4	4	6	8
27	4	4	2	4	10	6	6	4	6	2	2	4	2	2	4	2
28	0	10	10	6	6	0	0	12	6	4	0	0	2	4	4	0
29	4	2	4	0	8	0	0	2	10	0	2	6	6	6	14	0
30	0	2	6	0	14	2	0	0	6	4	10	8	2	2	6	2
31	2	4	10	6	2	2	2	8	6	8	0	0	0	4	6	4
32	0	0	0	10	0	12	8	2	0	6	4	4	4	2	0	12
33	0	4	2	4	4	8	10	0	4	4	10	0	4	0	2	8
34	10	4	6	2	2	8	2	2	2	2	6	0	4	0	4	10
35	0	4	4	8	0	2	6	0	6	6	2	10	2	4	0	10
36	12	0	0	2	2	2	2	0	14	14	2	0	2	6	2	4
37	6	4	4	12	4	4	4	10	2	2	2	0	4	2	2	2
38	0	0	4	10	10	10	2	4	0	4	6	4	4	4	2	0
39	10	4	2	0	2	4	2	0	4	8	0	4	8	8	4	4
40	12	2	2	8	2	6	12	0	0	2	6	0	4	0	6	2
41	4	2	2	10	0	2	4	0	0	14	10	2	4	6	0	4
42	4	2	4	6	0	2	8	2	2	14	2	6	2	6	2	2
43	12	2	2	2	4	6	6	2	0	2	6	2	6	0	8	4

44	4	2	2	4	0	2	10	4	2	2	4	8	8	4	2	6
45	6	2	6	2	8	4	4	4	2	4	6	0	8	2	0	6
46	6	6	2	2	0	2	4	6	4	0	6	2	12	2	6	4
47	2	2	2	2	2	6	8	8	2	4	4	6	8	2	4	2
48	0	4	6	0	12	6	2	2	8	2	4	4	6	2	2	4
49	4	8	2	10	2	2	2	2	6	0	0	2	2	4	10	8
50	4	2	6	4	4	2	2	4	6	6	4	8	2	2	8	0
51	4	4	6	2	10	8	4	2	4	0	2	2	4	6	2	4
52	0	8	16	6	2	0	0	12	6	0	0	0	0	8	0	6
53	2	2	4	0	8	0	0	0	14	4	6	8	0	2	14	0
54	2	6	2	2	8	0	2	2	4	2	6	8	6	4	10	0
55	2	2	12	4	2	4	4	10	4	4	2	6	0	2	2	4
56	0	6	2	2	2	0	2	2	4	6	4	4	4	6	10	10
57	6	2	2	4	12	6	4	8	4	0	2	4	2	4	4	0
58	6	4	6	4	6	8	0	6	2	2	6	2	2	6	4	0
59	2	6	4	0	0	2	4	6	4	6	8	6	4	4	6	2
60	0	10	4	0	12	0	4	2	6	0	4	12	4	4	2	0
61	0	8	6	2	2	6	0	8	4	4	0	4	0	12	4	4
62	4	8	2	2	2	4	4	14	4	2	0	2	0	8	4	4
63	4	8	4	2	4	0	2	4	4	2	4	8	8	6	2	2

Table 4.13: DES - Linear approximation table (S1)

Input Sum \ Output Sum	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	-2	-2	-4	-2	0	-4	6	2	0	0	6	4	-2	-6	4
3	0	-2	-2	-4	-2	0	-4	6	2	8	0	-2	4	6	-6	-4
4	0	2	-2	-4	-2	0	-4	-6	-2	4	8	2	0	-2	-6	12
5	0	-2	-2	0	-2	-4	-4	-2	2	-4	-4	2	4	-10	-2	-4
6	0	0	0	4	0	4	0	0	0	-4	4	4	0	0	-4	-8
7	0	-4	0	8	0	0	0	4	4	-4	-8	-4	4	0	0	0
8	0	4	-2	6	-6	-6	0	-4	-4	-4	2	-2	2	-2	0	0
9	0	0	6	-6	-2	-6	4	-4	0	-4	-2	6	2	-6	0	-4
10	0	-2	0	2	0	6	8	2	-2	0	-2	4	-2	0	-2	4
11	0	2	-8	-2	-4	-10	4	2	-6	8	2	4	-2	-4	-2	0
12	0	-2	0	6	0	2	0	2	2	0	6	-4	2	-4	6	0
13	0	6	0	6	4	-2	-4	-2	2	0	6	4	-2	8	-6	-4
14	0	0	-2	-2	2	2	0	0	4	4	6	-2	2	2	-4	4
15	0	0	-2	6	-2	-2	4	-4	-4	-4	-2	-2	-2	-2	0	0
16	0	2	2	0	-2	0	4	-6	0	6	2	-4	6	-4	-4	-18
17	0	2	-2	-4	2	-4	-4	10	-4	2	2	-4	-2	-4	0	-6
18	0	4	0	0	-4	4	0	4	-6	2	2	6	2	6	6	-10

19	0	4	-4	-4	0	0	-8	-12	-2	-2	-6	6	2	6	2	2
20	0	4	0	4	-8	-4	4	0	2	6	-2	2	6	2	-2	2
21	0	0	4	-4	-4	4	4	-4	10	2	2	2	-6	2	6	-2
22	0	6	2	0	2	-4	0	2	4	2	2	0	-2	0	0	2
23	0	2	6	-8	6	4	0	-2	-12	-2	-2	0	-6	0	0	-2
24	0	2	8	2	0	6	4	2	4	-2	4	6	0	-2	-4	2
25	0	-2	4	-6	0	-6	0	2	4	-6	8	6	0	2	0	-6
26	0	0	-6	2	-2	-2	4	4	-2	-2	0	0	-4	4	2	2
27	0	4	6	2	-10	2	-8	4	-2	-6	4	0	4	0	-2	2
28	0	-4	2	2	2	-6	0	-4	-2	-2	4	0	0	4	2	2
29	0	4	-2	-2	2	-6	-4	0	2	2	-4	0	-12	0	-6	-6
30	0	2	0	-2	4	-2	0	-2	0	6	-4	-2	0	-2	0	2
31	0	2	-4	2	-4	-2	4	2	4	-6	4	-2	-4	2	0	2
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
34	0	2	-2	0	2	0	0	6	-2	0	-4	6	4	10	10	0
35	0	2	-2	0	2	0	0	6	6	0	4	-10	-4	-6	2	0
36	0	2	-6	-8	2	4	4	2	2	0	0	2	0	6	-10	0
37	0	-2	2	4	2	0	-4	-2	-2	0	4	2	-4	6	-6	0
38	0	4	4	-4	8	-8	4	0	0	-8	0	-4	0	4	0	0
39	0	0	-4	-8	-8	4	-4	-4	4	-8	-4	-4	4	4	-4	0
40	0	4	-2	-2	-2	-2	-4	0	4	4	2	6	-2	-6	12	4
41	0	0	-2	-6	2	-2	-8	8	0	-4	-2	-2	6	-2	-4	0
42	0	2	0	-2	0	2	0	-2	2	-8	-6	-4	2	0	2	-4
43	0	-10	0	2	-4	2	4	6	-2	0	-10	4	2	-4	-6	0
44	0	6	-4	2	8	2	4	6	-2	-4	-2	12	-2	-8	-2	0
45	0	-2	-4	2	-4	-2	0	2	-2	-4	-2	4	-6	4	2	-4
46	0	-4	2	6	6	-6	-8	4	-4	0	2	6	6	2	4	0
47	0	-4	2	-2	2	-10	12	0	-4	0	2	-2	10	6	0	4
48	0	-2	-2	0	-2	4	0	2	0	2	6	4	6	0	0	-2
49	0	-2	2	4	2	0	0	-6	-4	-2	-2	-4	-2	0	-4	2
50	0	-4	-4	-4	0	0	0	4	-2	-2	-6	-2	-6	6	2	2
51	0	-4	0	0	4	-4	0	-4	-6	2	2	-2	2	-2	-2	-2
52	0	8	-8	8	4	4	0	0	-2	-2	2	-6	6	6	-2	-2
53	0	4	-4	0	-8	-4	0	-4	-2	2	-2	2	2	-2	-2	2
54	0	6	2	-8	2	-4	8	2	4	-6	2	0	6	0	0	2
55	0	2	-10	0	6	4	8	-2	4	6	-2	0	2	0	0	-2
56	0	-10	4	2	-4	-2	4	-2	4	2	0	6	-4	6	-4	-2
57	0	2	0	-6	-4	2	0	-2	-4	6	-4	-2	4	2	8	-2
58	0	0	6	-2	6	6	0	0	2	2	0	0	8	0	2	2
59	0	4	2	-2	-2	10	4	0	-14	-2	4	0	0	-4	-2	2
60	0	0	10	-2	-6	-2	0	8	-6	6	0	-8	-4	4	-2	2
61	0	8	-2	2	-6	-2	4	4	-2	-6	0	0	0	0	-2	2
62	0	2	0	-2	-8	2	4	2	0	-2	4	-2	-4	2	4	-2
63	0	-14	-12	-6	0	2	0	-2	-4	-6	12	-2	0	-2	4	-2

Table 4.14: Summary of the non-linear properties of DES S-boxes

S-Box	D_{max} (XOR Dist. Table)	L_{max} (Linear Appr. Table)
S1	16	-18
S2	16	-16
S3	16	16
S4	16	±16
S5	16	-20
S6	16	±14
S7	16	-18
S8	16	±16

Table 4.15: Robustness of DES S-boxes

S-Box	*nNZ*	D_{max}	R_s
S1	37	16	0.316
S2	33	16	0.363
S3	37	16	0.316
S4	24	16	0.469
S5	31	16	0.387
S6	33	16	0.363
S7	35	16	0.340
S8	36	16	0.328

4.5 Analysis of the AES S-box

The AES S-box is an 8×8 non-linear invertible substitution box constructed mathematically by applying two transformations in order to ensure high security against attacks, especially those of differential and linear cryptanalysis. The second transformation is added to further strengthen the algorithm against algebraic attacks such as an interpolation attack [102]. The process of generating the S-box is as follows:

1. Taking the multiplicative inverse in the GF(2^8) where the elements are represented in a polynomial form with a degree less than 8 and the coefficients in the GF(2). The multiplication is done modulo the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$. The element $\{00\}_h$ is mapped to itself.
2. Applying the following affine transformation over GF(2):

$$y_i = x_i \oplus x_{(i+4) mod\ 8} \oplus x_{(i+5) mod\ 8} \oplus x_{(i+6) mod\ 8} \oplus x_{(i+7) mod\ 8} \oplus c_i \qquad (4.3)$$

for $0 \leq i < 8$, where x_i is the ith bit of the byte, and c_i is the ith bit of a byte c with the value $\{63\}_h$.

In matrix form, the affine transformation element of the S-box is expressed as follows:

$$\begin{bmatrix} y_7 \\ y_6 \\ y_5 \\ y_4 \\ y_3 \\ y_2 \\ y_1 \\ y_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \tag{4.4}$$

The AES S-box is shown in Table 4.16, and the elements of the table are represented in hexadecimal format such that the value $\{xy\}_h$ in hexadecimal form represents the decimal value ($16x + y$), and x and y stand for the table's row and column indices. For example, the input $\{81\}_h$ to the S-box is mapped to $\{0c\}_h$ by the intersection of the row index 8 and the column index 1.

Table 4.16: AES S-box

Row No.	Column No. 0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

The AES Inverse S-Box is obtained by applying the inverse of the affine transformation as specified in equation 4.5 followed by taking the multiplicative inverse in GF(2^8).

$$\begin{bmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} y_7 \\ y_6 \\ y_5 \\ y_4 \\ y_3 \\ y_2 \\ y_1 \\ y_0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \tag{4.5}$$

The AES inverse S-box is shown in Table 4.17.

Table 4.17: AES inverse S-box

Row No.	Column No. 0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

4.5.1 Analysis of the Non-linear Properties of the AES S-box

The non-linear properties of the AES S-box are examined by following the steps discussed earlier in section 4.3. The XOR distribution and linear approximation tables are first built, but their large sizes of 256×256 dimensions mean that they are too big to reproduce here in full. Therefore, only a sample of one row with all columns from each table is shown below. For instance, from the XOR distribution table, a randomly selected row number 100 with all columns is shown in Table 4.18. All other rows have the same values with different distributions. The maximum value in the table is 4, and thus the DPP_{max} is $4/2^n = 4/2^8 = 2^{-6}$.

Table 4.18: Part of XOR distribution table of the AES S-box

0	2	0	2	0	0	0	0	2	2	0	0	0	2	2	2
2	0	0	2	2	0	0	2	0	2	0	2	0	2	2	0
2	2	2	2	2	0	2	2	0	0	0	0	2	2	0	2
0	0	2	0	0	0	0	2	0	0	0	0	2	0	2	2
2	0	2	2	0	2	0	0	2	2	2	2	2	2	2	2
2	0	0	0	0	2	2	2	2	2	0	0	0	0	0	2
2	2	2	0	0	2	0	2	2	0	2	2	0	2	2	2
0	2	0	2	0	2	2	2	0	0	0	0	0	2	2	2
0	2	2	2	0	0	2	2	2	0	2	2	2	2	2	2
2	0	0	0	2	0	2	2	4	2	0	0	0	2	2	0
0	0	0	2	0	0	2	2	0	0	0	2	2	2	0	0
2	0	2	0	0	0	0	0	0	0	2	2	0	2	0	0
0	2	2	2	0	0	0	2	2	0	0	2	0	0	2	0
2	2	2	0	0	2	2	2	0	0	2	0	2	2	2	0
0	2	0	0	0	2	0	0	0	2	0	2	0	0	2	0
0	0	2	0	0	2	0	2	0	2	0	2	0	0	2	0

Part of the linear approximation table for the same row number is displayed in Table 4.19. The remaining rows of the table have the same values with different distributions, and the maximum value in the table is |16|, resulting in a maximum probability bias (PB_{max}) equal to $16/2^n = 16/2^8 = 2^{-4}$. The IOC_{max} is equal to 2^{-3}, since as mentioned earlier the input-output correlation is twice the bias.

Table 4.19: Part of linear distribution table of the AES S-box

0	0	6	6	-14	2	-8	16	14	-2	12	-12	12	-4	2	2
-14	-10	12	8	0	-12	2	6	-4	-16	-2	2	-2	2	-16	12
-4	0	-2	2	6	2	8	12	10	-2	-4	-8	0	-4	10	6
6	-2	-4	-4	-4	12	-6	10	-8	0	14	6	2	2	8	0
-4	-8	-2	2	-10	10	-8	-4	6	10	0	-12	12	8	-2	2
6	-2	-12	12	-4	12	2	10	-12	4	-6	-14	-2	-10	4	12
-12	4	2	10	6	-10	-12	4	6	14	-12	12	-12	-4	-6	10
-2	-6	8	4	12	8	14	2	-4	0	6	2	14	2	-8	12
-10	-6	-12	0	8	4	14	-6	8	4	-2	10	-2	-14	4	0
4	4	-2	-2	-6	-14	-12	4	10	2	-4	-4	12	12	-10	6
-10	6	8	-16	-16	0	10	-6	8	0	-6	2	6	-2	8	8
-4	-8	-6	-10	-6	-10	-8	12	-6	-2	-8	-12	4	-8	-6	14
14	-10	0	-8	0	8	-6	-6	12	4	6	6	2	10	12	-12
-4	-8	2	6	2	-2	-8	-12	6	2	4	0	-8	-12	-2	2
2	6	8	-4	-4	8	10	14	-8	12	-2	10	6	-14	-4	-8
0	8	-6	-6	-2	14	8	8	-6	2	4	12	-12	4	6	-2

4.5.2 Robustness of the AES S-box

The robustness of the AES S-box can be calculated by applying equation 4.2. The results reflect the high resistance of this S-box to differential cryptanalysis.

$R_s = \left(1 - \frac{nNZ}{2^n}\right)\left(1 - \frac{D_{max}}{2^n}\right) = 0.984$, where $nNZ = 0$ and $D_{max} = 4$.

4.6 Generation of the 8×8 S-box

Three different S-boxes can be derived from the initial AES S-box by changing the order of the output bytes, such that they are circularly shifted with different offsets. All possible offsets are considered and only three cases result in the same characteristics and high non-linear properties as those of the AES S-box. These offsets are 64, 128 and 192. Accordingly, four different S-boxes can be used in the algorithm, while only one look-up table is actually constructed, and this will be reflected in the efficiency of the system in terms of its security and complexity.

4.7 Generation of the 7×7 S-box

A 7×7 S-box is generated by following the same steps used in the construction of the AES S-box, replacing both the degree-8 irreducible polynomials $(x^8 + x^4 + x^3 + x + 1)$ to degree-7 ($x^7 + x + 1$) [103, 104] and the degree-8 reduction polynomials ($x^8 + 1$) to degree-7 $(x^7 + 1)$, respectively. The non-linear properties are calculated such that the maximum value in the XOR distribution table is 2, providing a $DPP_{max} = 2^{-6}$ and a maximum value in the linear approximation table of ±10, thus giving a $PB_{max} = 2^{-3.678}$ or an $IOC_{max} = 2^{-2.678}$. The forward and inverse S-boxes in hexadecimal format are shown in Tables 4.20 and 4.21, respectively. In addition, another three S-boxes are generated from this S-box which have the same non-linear properties of the original by changing the order of the output elements with different offsets of rotation of 32, 64 and 96.

4.7.1 Robustness of the 7×7 S-box

The robustness of the S-box generated can be computed by applying equation 4.2, and the result is exactly the same as for the robustness of the AES S-box, which is 0.984. This result reflects the robustness of this S-box to differential cryptanalysis.

Table 4.20: 7×7 S-box

Row No.	Column No.															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7C	33	3	54	43	53	35	27	2E	73	6B	64	8	48	16
1	5E	5A	5	38	74	24	67	50	20	25	9	58	29	71	6	70
2	22	7	3F	76	4F	52	0E	6D	37	65	0	4B	7E	60	3A	69
3	1D	44	5F	0B	56	40	3E	4C	46	72	75	4A	11	59	2A	21
4	1C	5D	4E	23	4D	78	36	0A	6A	2F	3B	6C	15	6E	7B	79
5	49	14	7F	1	0D	51	77	5B	32	13	3D	1B	0F	41	66	17
6	5C	1F	30	55	7D	7A	57	47	26	0C	2D	39	12	3C	34	19
7	31	1A	2B	6F	68	2	28	62	45	10	61	1E	18	4	42	2C

Table 4.21: 7×7 Inverse S-box

Row No.	Column No.															
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	2A	53	75	3	7D	12	1E	21	0D	1A	47	33	69	54	26	5C
1	79	3C	6C	59	51	4C	0F	5F	7C	6F	71	5B	40	30	7B	61
2	18	3F	20	43	15	19	68	8	76	1C	3E	72	7F	6A	9	49
3	62	70	58	2	6E	7	46	28	13	6B	2E	4A	6D	5A	36	22
4	35	5D	7E	5	31	78	38	67	0E	50	3B	2B	37	44	42	24
5	17	55	25	6	4	63	34	66	1B	3D	11	57	60	41	10	32
6	2D	7A	77	0	0C	29	5E	16	74	2F	48	0B	4B	27	4D	73
7	1F	1D	39	0A	14	3A	23	56	45	4F	65	4E	1	64	2C	52

4.8 Conclusions

In this chapter several objectives have been achieved as follows:

- The non-linear properties of both the DES and AES S-boxes are examined and determined by computing the DPP_{max} and PB_{max} or IOC_{max}, in addition to the robustness of the S-boxes. All of these calculations are conducted after building the XOR distribution and linear approximation tables.
- Three different S-boxes are derived from the AES S-box by changing the order of the output bytes. The S-boxes derived have the same non-linear properties as those of the original AES S-box.
- A new 7×7 S-box is generated which is similar to the AES S-box in structure, in order to meet the security requirements of the proposed cipher. The non-linear properties of this new S-box are good compared to the current state-of-the-art AES S-box, where the DPP_{max} is the same for both S-boxes which is 2^{-6} and the IOC_{max} is 2^{-2678}, whereas it is 2^{-3} for the AES. Moreover, another three S-boxes are generated from this S-box by changing the order of the output elements, and the same non-linear properties result.

Chapter 5

Design of the Proposed Algorithms

In this chapter a secure and efficient block cipher, based on the SPN and the NMNT, is designed according to Shannon's principles [15] and modern cryptographic theorems. The chapter is organised as follows. Section 5.1 gives a brief introduction to the topic and section 5.2 discusses the design of the proposed algorithms. Section 5.3 explains the generation of the keys, and the implementation of the algorithm and test vectors are presented in sections 5.4 and 5.5, respectively. The complexity of the algorithm is computed in section 5.6, and a discussion of the system is given in section 5.7. Another algorithm based on the FNT is then outlined in section 5.8, and finally the conclusions of the chapter are drawn in section 5.9.

5.1 Introduction

Fundamental principles when designing a secure block cipher are relate to confusion and diffusion [15], which remain the most widely accepted principles to design secure cryptosystems. In addition, the steps necessary to build efficient diffusion had been proposed [3], which can be constructed by combining two steps. The first step provides high local diffusion, by using a transform having a high branch number which works on a limited number of bundles. Examples here include the Mix column transform in AES [3], the MDS transform in Twofish [29], and the PHT in SAFER [26]. The second step provides high dispersion once the first step is implemented, the locations of elements

that are too close are changed. Confusion can be achieved by mapping the values of element through the use of a substitution box. The design of the proposed algorithm follows this sequence, as illustrated in the following sections.

5.2 Algorithm Design

The proposed algorithm is an iterated symmetric-key block cipher based on the SPN with variable block size, key length and word length. The block size and key length ranging from 16×*P*-bit up to 256×*P*-bit with a power of two, for a word length $P = 7$, 31, 61, or 127. The general block diagram for the proposed forward algorithm (encryption) is shown in Figure 5.1. It starts with an initial key addition, followed by *rd* identical rounds. The number of rounds is 10, however this can vary. The round functions consist mainly of the following five layers: element substitution; shifts transform; variable addition; NMNT; and, finally, round key addition. This order of operations is for encryption, and for decryption the algorithm and the round keys are run in reverse order, with inverse functions applied instead. High level descriptions of both the encryption and decryption are provided in Algorithms 5.1 and 5.2, respectively.

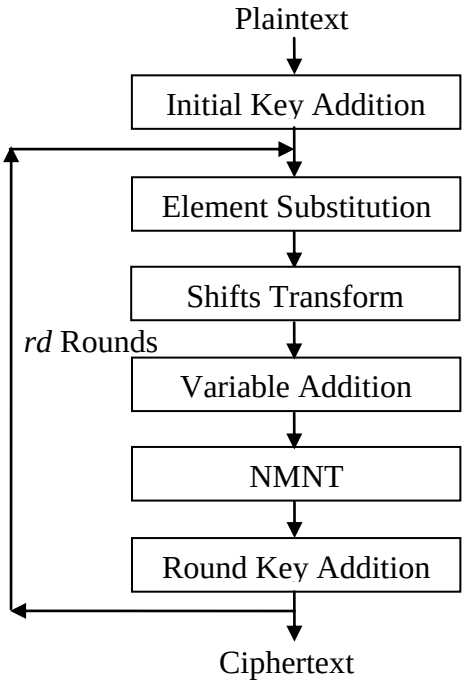


Figure 5.1: Block diagram of the proposed algorithm

/* Initial key addition */

a = (PT + $Kr(0)$) % M_p; /* PT: Plaintext block, Kr: round key */

/* Loop 10 times (10 : total number of rounds) */

for (i = 1; $i \leq 10$; i++)

{

/* Elements Substitution*/

a = S (a);

/* Shifts Transform*/

a = ST (a);

/* Variable addition */

a = VA (a) % M_p;

/* NMNT*/

a = NMNT (a);

/* Round keys addition */

a = (a + $Kr(i)$) % M_p;

}

CT = a;

/* CT: Output Ciphertext block */

Algorithm 5.1: Encryption Algorithm (High Level Description)

/* *CT*: Input Ciphertext block */

$a = CT$;

/* Loops 10 times in descending order (10 : total number of rounds) */

for ($i = 10$; $i > 0$; i--)

{

/* Round keys subtraction */

$a = (a - Kr(i)) \% M_p$;

/* Inverse NMNT*/

a = Inv_NMNT (a);

/* Variable Subtraction */

a = VS (a) % M_p;

/* Inverse Shifts Transform*/

a = Inv_ST (a);

/* Inverse Elements Substitution*/

a = SI (a);

}

/* Initial key Subtraction */

$PT = (a - Kr(0)) \% M_p$;

/* *PT*: Output Plaintext block */

Algorithm 5.2: Decryption Algorithm (High Level Description)

5.2.1 Block Size

Both the block size and key length of the proposed algorithm are variable. A block consists of a number of elements with a *P*-bit word length distributed in a two dimensional array, where the number of rows (*R*) can be 4, 8, or 16 and the number of columns is represented by the NMNT length (*N*), which can be 4, 8, or 16 as shown in Figure 5.2. Here, the number of rows is either the same as or half of the transform length. The block sizes are chosen carefully such that by processing the data the highest possible levels of diffusion are achieved.

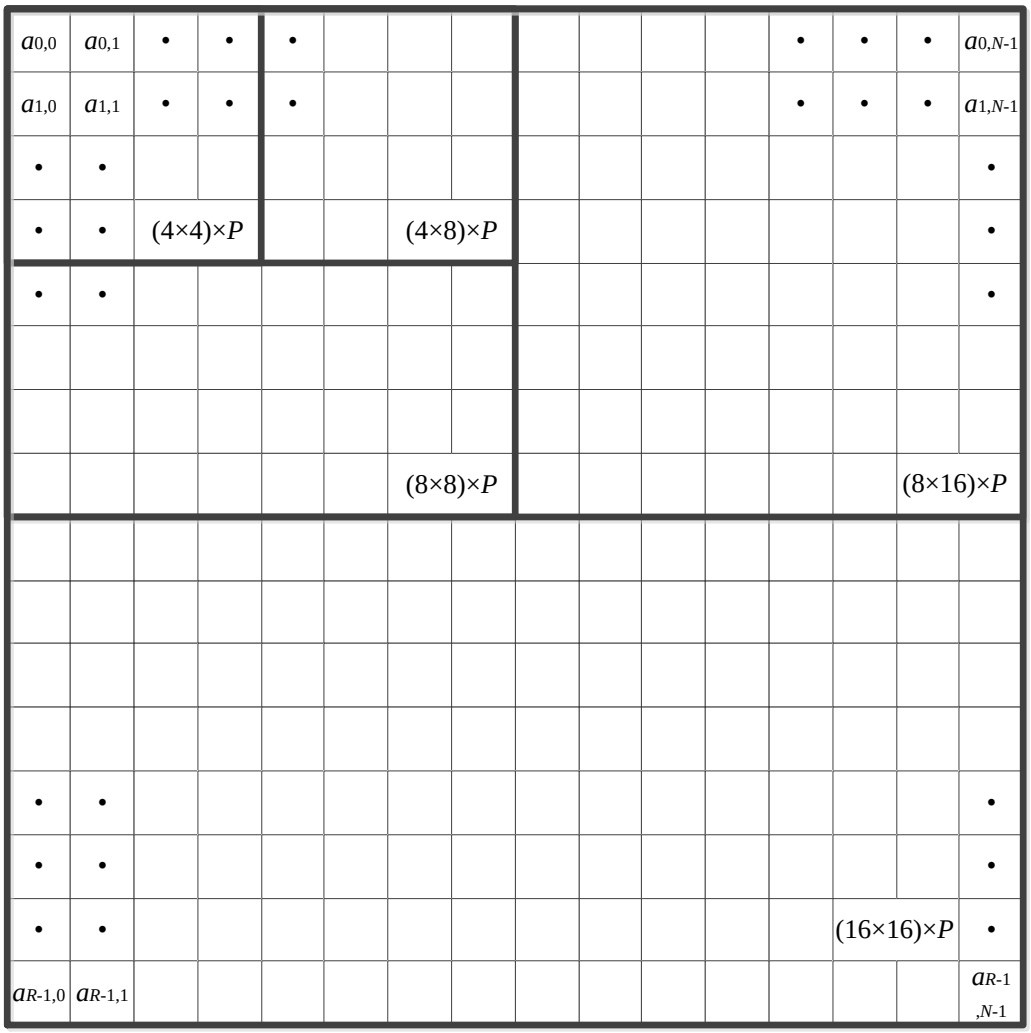


Figure 5.2: Proposed block sizes

The message is converted into a number of blocks, onto which the elements are vertically mapped starting from the top left corner (top to bottom and left to right), that is, in the order $a_{0,0}$, $a_{1,0}$ ... $a_{R-1,0}$, $a_{0,1}$, $a_{1,1}$ ... $a_{R-1,1}$, ... , $a_{0,N-1}$, $a_{1,N-1}$ ... $a_{R-1,N-1}$.

The selection of key length and block size as well as word length is usually agreed between the communicating parties before exchanging information. However, the system is built up in such a way that either these parameters are agreed beforehand or they are efficiently set by the system. The system is initially configured with default values of P = 7 for word length and 16×P for both key and block length. The system can select word length depending on the length of the processor registers in the machine that carries out the encryption. The block size is equivalent to the key length and is chosen depending on the message length such that the final block padding required is minimal, noting that a larger block is selected for the case of equal padding on different block sizes (the details are explained in Algorithm 5.3). This means that different plaintexts can have different block and key sizes depending on their length. So not only is block size variable, but its value is kept secret to increase the overall security of the system and to render the task of cryptanalysis as hard as possible. The values of block size and word length are encrypted and sent within the ciphertext at a random location depending on the key, in order to be used later in the decryption phase.

```
/* ml is the number of elements in the message (including 1 byte padding) */
ml = ceil ((nM + 8) / (P - 1));   /* nM is the message length in bits */
c = 256 - ml % 256;   /* modulo number of elements of larger block size */
/* nb is the block size in bits, P: word length */
nb = 256 × P;
for (i = 7; i > 3; i--) {
        if (pow (2 , i) - ml % pow (2 , i) < c) {
                c = pow (2 , i) - ml % pow (2 , i);
                nb = pow (2 , i) × P;
        }
}
```

Algorithm 5.3: Chosen a block size of the system

5.2.2 Components of the Design

The structure of the design as shown in Figure 5.1 starts with the initial key addition layer followed by round transformation which, as mentioned earlier, consists of five layers identical for all rounds. In addition, there are supplementary padding and conversion layers. The former is used to force the length of the plaintext to be a multiple of block size by padding at the end of the message with a number of bytes with zero value. The conversion layer is used to convert each word from one length to another and also to convert a vector to an array and vice versa. The elements are processed *rd* times throughout the round transformations until it is ensured that the ciphertext is resilient to attacks, taking into consideration predetermined safety margins. The components of the design are explained below.

1. Padding

   The padding layer starts with the insertion of a byte at the end of the plaintext with a value of 1. This indicates the end of the message, which is very useful when removing padded bytes. Next, a number of bytes with a value of 0 are padded at the end if necessary in order that the length of the plaintext is a multilpe of the block size. This step can be conducted after the size of the block and word length has been determined. The maximum number of padded bytes can be any value up to one block. The number of padded bytes (*nZ*) required to fill the gaps in the final block is calculated using equation 5.1. Noting that there is a 1-bit is reserved at each element for error avoidnace, as explained later in this chapter.

$$nZ = ib - \ remainder\ \left(\frac{nP+1}{ib}\right) \tag{5.1}$$

   where *nP* is the plaintext length in bytes and *ib* is derived in equation 5.2, which represents the number of required input bytes per block.

$$ib = \frac{R \times N}{8} \times (P - 1) \tag{5.2}$$

   where *R* and *N* stand for the block dimensions; *R* representing the number of rows, and *N* is column length (which is the transform length), and *P* is word length.

2. Conversion of Elements

In this layer the elements are converted from one length to another. In the encryption phase after the padding layer, the input bytes are converted from 8-bit to *P*-1-bit, and then converted into a number of blocks of predetermined size. Later, after the round transformation, the output is converted back from the transform domain (*P*-bit) into bytes (8-bit). Another conversion is needed inside the round transformation before and after the element substitution layer, in case the word length is greater than the default value. So for $P \neq 7$, each element is converted into a number of sub-elements with S-boxes input lengths which could be 7 or 8-bit or a combination of both, and then later the converted sub-elements are return back to the original length after substituting them. The conversion is achieved by converting the elements into a binary form, concatenating them all and then slicing them into the required length.

In the decryption phase, each element is converted from bytes to *P*-bit and later from *P*-bit to bytes after removing the zero bits added at the beginning.

3. Initial Key Addition

Starting and ending the cipher with a key addition, sometimes referred to as whitening [53], is important in the design, since any other layers before the first or after the last key additions could be peeled off without prior knowledge of the key, for instance in the initial and final permutations in the DES algorithm [3, 53]. In addition, this operation obstructs a cryptanalysis by hiding the required information in specific rounds. Initial key addition is applied in several designs, such as AES [3], Blowfish [17], IDEA [42], Khufu/Khafre [33], SAFER [26], and Twofish [29].

4. Element Substitution

This is the non-linear part of the proposed algorithm which plays a crucial role in the overall security of the system, where its strength is strongly influenced by that of the S-boxes involved. The strength of the S-box is evaluated according to the strength of its non-linear properties, as discussed in detail in chapter 4. Confusion is achieved in this layer by non-linearly mapping the elements through the destination S-boxes. The number of S-boxes used in this layer depends on the selection of word length. For example, for $P = 7$, one 7×7 S-box is used, while for word lengths greater than 7 a combination of 7×7 S-boxes and 8×8 S-boxes are utilised.

5. Shifts Transform

Elements are redistributed in this layer to satisfy the function of dispersion [3], where the columns of the array are cyclically shifted with different offsets such that the resultant elements in each row are a combination of elements from all rows.

6. Variable Addition

In this layer, a set of variable values are added to certain elements in the array and then the results are multiplied by 2 according to equation 5.3, where the addition and multiplication are modular operations. One element is modified from each row, and the locations of the modified elements are very important. All elements in the kernel matrix at row/column *N*/4+1 have no zero value, and consequently modifying elements at such locations ensure after processing the NMNT layer the effect of the elements modification is achieved. While modifying elements in other places except those mentioned in point 6, section 3.4.1, a maximum of *N*/4 elements in the worst cases could be unmodified in each row. The value of the variable is a function of the row number, the round number and the block number. Three purposes are achieved in this layer. Firstly, by adding the variables to the intermediate results ensure that the cases where the NMNT provides low diffusion are eliminated by eliminating symmetries in the correlated elements. Secondly, the addition of variables with different values to different elements results in a ciphertext consisting of different elements corresponding to a plaintext made up of the same elements. Thirdly, the involvement of the value of block number in the calculation of the variable results in different values for different blocks, which ensures that two different ciphertext blocks are obtained when encrypting two identical plaintext blocks using the same cipher key. Secure output is thus gained by implementing a simple mode of operation, which is then reflected in the efficiency of the system in terms of speed and complexity.

$$a'(i, N/4+1) = \langle (a(i, N/4+1) + (i^3 \times q) + bn) \times 2 \rangle_{M_P}$$
$$for\ i = 1,2 \dots R,\ \ q = 1,2 \dots rd \qquad (5.3)$$

where *a´* is the updated element value and *a* is the original element value, *N* is the transform length, *bn* is the block number, *R* is the maximum number of rows and *rd* is the maximum number of rounds

7. New Merssene Number Transform

   The NMNT is used in the design in order to ensure high diffusion. This transform was investigated in Chapter 3 and shown to exhibit good diffusion characteristics [97, 99]. Prior to the development of differential and linear cryptanalysis a simple permutation alone was sufficient to diffuse data. As new analytic systems evolved, more complex techniques were required, such as the use of a linear transformation which improves the avalanche characteristic of the cipher and increases its resistance to such attacks [105].

8. Round Key Addition

   In this layer, the round keys that are derived from the cipher key when applying the key schedule algorithm are added modulo the M_p to the intermediate result at each round, as explained in equation 5.4. The length of the round key is the same as the block length.

$$a'(i,j) = \langle a(i,j) + k_{rm}(i,j) \rangle_{M_P}$$
$$for\ i = 1,2 \dots R,\ j = 1,2 \dots N,\ m = 1,2 \dots rd \qquad (5.4)$$

   where *a´* is the updated element value, *a* is the original element value, *kr* is the round key, *R* is the maximum number of rows, *N* is the transform length and *rd* is the maximum number of rounds

5.2.3 Number of Rounds

The number of rounds is determined by investigating in which round the system can be successfully broken, and adding then a sufficient number of rounds to achieve the security margins required. Schneier has suggested that, if a system can be broken in *n* rounds, it should be designed with 2*n* or 3*n* rounds [5]. In addition, he recommended a minimum number of rounds for the current standard as follows: 16 rounds for AES-128; 20 rounds for AES-192; and 28 rounds for AES-256. The formula proposed for this is shown in equation 5.5 [55]. For instance, for AES-128, d = 1, nb = 128, and w = 8, resulting in $rd \geq 16$ as suggested by Schneier. In another example for Twofish [29], d = 2 (Feistel structure, half of the data is processed in the confusion stage at each round), nb = 128, and w = 8, producing $rd \geq 32$. Both values resulting from this formula have larger values than those proposed by the algorithm designers.

$$rd \geq (d \times nb)/w \qquad (5.5)$$

where *rd* is the number of rounds, *d* is the maximum number of rounds required to process all words of a block in the confusion stage, *nb* is the block size, and *w* is the minimum word size input to the confusion stage.

In the proposed algorithm, both differential and linear cryptanalysis have been investigated as explained in Chapter 6 and it has been proven that the system is secure against such attacks from round three regardless of block or key size. Accordingly, 10 rounds have been suggested, taking into consideration other possible types of attack and sufficient safety margins.

5.3 Generation of Cipher Key and Round Keys

This section deals with the generation of the necessary sub-keys. In the first part an algorithm is proposed for deriving the cipher key from the secret key, and the second part deals with the key schedule algorithm as the mechanism for deriving round keys from the cipher key. Two key schedule algorithms are proposed. Both combine high diffusion and non-linearity of properties in order to generate round keys with high resistance to attacks such as the related-key attack.

5.3.1 Cipher Key Generation

A cipher key (k_c) is generated from the secret key or password supplied by the user, which can be of any length, by padding the key if necessary to the block length by applying equation 5.6, which is derived from [3]. If the secret key provided has a length longer than required, then the cipher key is derived by truncating the key to the required length.

$$k_c(i) = \langle k_c(i-1) + k_c(i-nc) \rangle_{M_P} \qquad i = nc+1, nc+2, \ldots bl \qquad (5.6)$$

where *i* is the location of the processed element, *nc* is the number of elements in the provided secret key, *bl* is the block length (elements).

For example, if *bl* = 16 and the secret key = 1 2 3 4 5 6 7 8, then the padded cipher key is k_c = 1 2 3 4 5 6 7 8 9 11 14 18 23 29 36 44. If k_c = 1 2 3 4 5 6 7 8 9, then the cipher key generated is 1 2 3 4 5 6 7 8 9 10 12 15 19 24 30 37.

Padding the cipher key according to this method results in a more random appearance than just padding with zero elements, as applied in existing techniques.

5.3.2 Round Keys Generation

Round keys are derived from the cipher key by means of a key schedule algorithm. The lengths of the cipher key and round keys are the same as the block length. The total number of round keys is equal to the number of rounds. Two techniques are used in generating the round keys, as explained below.

1. Key schedule algorithm (first technique)

   The first key schedule algorithm applies the same round transformation of the cipher for easy analysis and implementation, where the input to the cipher is the cipher key and each round is used to generate one round key. This means generating all round keys equivalent to encrypting one block of data. The round transformation is modified in such a way that the initial key addition layer is eliminated and, the variable addition layer is a function of only row and round numbers. In addition, rather than adding a single round key at each round in the round key addition layer an accumulation of all previously generated round keys are added. Thus each round key generated is a function of all previously generated round keys, in addition to the cipher key and the round number [3]. For example, the key that is used in the round key addition layer to generate the first round key is the modular addition of the cipher key and the number 1 for round number one. The second round key is then the modular addition of the cipher key, the first generated round key and the number 2. For the final round, the key generated is the modular addition of the cipher key, and the first up to the ninth round keys and the number 10 for round number ten.

   For example, if P = 7, block size is 16×P, and the password or secret key is "Dell Laptop", then the ASCII representations for the secret key with 8-bit each is Ks = 68 101 108 108 32 76 97 112 116 111 112, where 32 is the ASCII code for the space character. Next, the cipher key will be derived from the secret key by applying equation 5.6 if necessary. The length of the cipher key is 16 P-bit or 16×P/8 = 14 bytes, i.e. Kc = 68 101 108 108 32 76 97 112 116 111 112 180 25 133. Then, the 14 bytes of Kc are converted into 16 7-bit elements to be compatible with the transform domain. The converted Kc = 34 25 45 70 97 1 24 97 56 29 13 119 5 80 51 5. The conversion is achieved by converting Kc elements into binary representation, concatenating all of the elements and then slicing them into 7-bit each. The detailed sequence of the generation process for the first three round keys is shown in Table 5.1. The final round keys generated are shown in Table 5.2, in addition to the cipher key, which is represented in the first row in the table.

Table 5.1: Three round keys generation using first technique

	Round 1				Round 2				Round 3			
Input Key	34	97	56	5	81	38	91	75	40	5	3	59
	25	1	29	80	121	62	77	4	51	102	70	10
	45	24	13	51	54	69	22	13	38	81	67	50
	70	97	119	5	45	52	97	104	76	107	56	98
Elements Sub.	63	31	70	67	20	14	27	108	55	67	3	74
	37	124	113	73	16	42	110	84	11	87	54	115
	96	32	8	11	62	120	103	8	14	20	35	95
	54	31	98	67	96	86	31	38	21	57	70	48
Shifts Transf.	54	32	113	67	96	120	110	108	21	20	54	74
	63	31	8	73	20	86	103	84	55	57	35	115
	37	31	98	11	16	14	31	8	11	67	70	95
	96	124	70	67	62	42	27	38	14	87	3	48
Variable Add.	54	66	113	67	96	117	110	108	21	46	54	74
	63	78	8	73	20	77	103	84	55	35	35	115
	37	116	98	11	16	9	31	8	11	42	70	95
	96	122	70	67	62	86	27	38	14	50	3	48
NMNT	46	67	34	69	50	122	108	104	68	66	82	122
	95	60	47	50	30	37	89	51	113	67	67	100
	8	44	8	88	64	113	30	111	91	15	71	121
	101	81	104	98	86	83	92	114	115	13	46	9
Accumul. Key	35	98	57	6	117	10	22	82	31	16	26	15
	26	2	30	81	21	65	108	86	73	41	52	97
	46	25	14	52	101	95	37	66	13	50	105	117
	71	98	120	6	117	24	91	111	67	5	21	83
Adding Key	81	38	91	75	40	5	3	59	99	82	108	10
	121	62	77	4	51	102	70	10	59	108	119	70
	54	69	22	13	38	81	67	50	104	65	49	111
	45	52	97	104	76	107	56	98	55	18	67	92

Table 5.2: Round keys generation using first technique

K_c	34	25	45	70	97	1	24	97	56	29	13	119	5	80	51	5
K_{r1}	81	121	54	45	38	62	69	52	91	77	22	97	75	4	13	104
K_{r2}	40	51	38	76	5	102	81	107	3	70	67	56	59	10	50	98
K_{r3}	99	59	104	55	82	108	65	18	108	119	49	67	10	70	111	92
K_{r4}	106	33	48	107	29	60	53	69	0	70	62	90	52	38	116	44
K_{r5}	65	80	73	16	57	12	0	64	23	123	67	16	30	31	27	111
K_{r6}	9	5	32	98	74	68	106	53	68	104	67	93	97	3	16	119
K_{r7}	27	70	107	82	91	33	47	106	53	56	23	16	18	17	118	93
K_{r8}	52	39	40	3	41	82	97	16	39	8	0	88	107	14	24	75
K_{r9}	66	65	32	41	2	22	123	18	101	65	72	102	22	3	65	72
K_{r10}	77	26	101	65	3	64	87	124	55	107	24	80	8	85	58	117

2. Key schedule algorithm (second technique)

The second proposed key schedule algorithm is key-dependent. With a structure similar to that of the first technique, it is achieved by replacing both the shifts transform layer and the variable addition layer with key-dependent permutation and key-dependent addition layers. These two layers are processed by first building up two arrays with the block size length at each round for each layer; these arrays represent the row and column indices, respectively. For the key-dependent addition layer, a key-dependent modular addition is applied to all elements, where the two arrays generated are used to determine for each element the location of the corresponding element that will be added to it. The first array, which is called the Addition row indices array (*Ar*), is generated by applying equation 5.7. It is achieved by modularising the key of the previous round (Kr_{d-1}) for modulo equal to the total number of rows after mapping it to the element substitution layer.

$$Ar_d(i,j) = \langle S\big(Kr_{d-1}(i,j)\big)\rangle_R$$
$$for\ d = 1,2 \ldots rd, i = 1,2 \ldots R, j = 1,2 \ldots N \qquad (5.7)$$

where $S(\cdot)$ for element substitution, R is the total number of rows, rd is the total number of rounds and N is the transform length.

The second array, known as the Addition column indices array (*Ac*), is generated according to equation 5.8, by modularising the accumulated key (k_a) at that round with modulo equal to the transform length. The accumulated key at each round represents the summation of the cipher key and all round keys generated before that round.

$$Ac_d(i,j) = \langle k_a(i,j) \rangle_N$$
$$for\ d = 1,2 \ldots rd, i = 1,2 \ldots R, j = 1,2 \ldots N \qquad (5.8)$$

For the key-dependent permutation layer, the locations of the elements are changed based on a key. Two arrays, *Pr* and *Pc*, are generated, which are used to determine the new location for each element. They are built by first filling up the arrays with an initial sequence of indices values, and then the sequence of the indices is continuously changed based on the values in the *Ar* and *Ac* arrays until all elements have been processed.

A simple illustration of this is shown in Table 5.3, consider a key *K*(*n*) as a vector of 16-elememts with a single dimension. The corresponding Addition indices vector *A*(*n*) is calculated by modularising the key *K*(*n*) for modulo equal to the total number of elements, i.e. 16. The permutation indices vector *P*(*n*) is initially filled with indices in sequence from 1 to the total number of elements, 16. Then the sequence of indices are changed based on the corresponding values in the *A*(*n*) vector. For example, for *A*(1) = 2 the index in *P*(1) is swapped with the index in *P*(2), for *A*(2) = 9 the updated index in *P*(2) is swapped with the index in *P*(9), and the generation process are continue in the same manner until processing all *P*(*n*) elements with respect to *A*(*n*) elements, where the last row represents the new indices of the elements at that round.

By taking the same parameters and the same secret key as the example in the first proposed structure, the cipher key is derived by following the same procedures by padding it to the required length. A summary of all round keys generated in addition to the cipher key is shown in Table 5.4. The generation process for the first three round keys is explained in Table 5.5.

Many important properties are used in the key schedule designs in order to make successful related-key and other types of attack more difficult. Such properties include the use of a round number to eliminate symmetries, and the generation of any

round key depending on all previously generated round keys and the cipher key. This means that knowing part of the cipher key or one or more full round key will not be sufficient to generate the cipher key or other round keys. In addition, high diffusion and non-linearity are achieved.

Table 5.3: Generation of key-dependent permutation index array

$K(n)$	34	25	45	70	97	1	24	97	56	29	13	119	5	80	51	5
$A(n)$	2	9	13	6	1	1	8	1	8	13	13	7	5	16	3	5
$P_1(n)$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$P_2(n)$	2	1														
$P_3(n)$	5	9	13	6	2	4			1				3			
.	4					5	8	7								
.	7							4								
.								1	4	3			10			
.							12				10	8	11			
.			15		11								2	16	13	14
.					14											11
$P_j(n)$	7	9	15	6	14	5	12	1	4	3	10	8	2	16	13	11

Table 5.4: Round keys generation using second technique

K_c	34	25	45	70	97	1	24	97	56	29	13	119	5	80	51	5
K_{r1}	103	107	77	2	73	88	43	119	49	40	93	35	56	117	11	95
K_{r2}	70	121	8	118	105	44	50	38	91	112	20	40	34	48	91	52
K_{r3}	2	80	74	17	54	25	26	83	25	66	43	61	39	104	61	36
K_{r4}	100	97	29	42	83	23	86	51	104	70	14	64	8	98	90	61
K_{r5}	118	54	78	111	0	100	52	54	25	38	50	48	36	123	76	83
K_{r6}	114	2	67	58	60	85	32	8	35	96	12	43	22	24	86	36
K_{r7}	27	82	108	124	33	14	43	51	21	125	37	109	27	3	113	12
K_{r8}	121	19	108	52	110	38	77	41	79	67	38	115	99	58	20	8
K_{r9}	74	19	96	121	41	112	30	14	81	25	44	78	89	103	30	47
K_{r10}	64	60	84	53	93	62	51	25	2	21	60	26	125	7	52	49

Table 5.5: Three round keys generation using second technique

	Round 1				Round 2				Round 3			
Input Key	34	97	56	5	103	73	49	56	70	105	91	34
	25	1	29	80	107	88	40	117	121	44	112	48
	45	24	13	51	77	43	93	11	8	50	20	91
	70	97	119	5	2	119	35	95	118	38	40	52
Elements Substitute	63	31	70	67	71	47	68	70	54	12	27	63
	37	124	113	73	57	50	55	2	16	126	49	29
	96	32	8	11	110	75	65	107	39	95	116	27
	54	31	98	67	51	98	118	23	40	14	55	86
Kd Permut. Row Index	2	3	2	1	1	1	4	2	2	1	3	3
	3	3	4	1	4	2	4	3	4	2	4	1
	1	4	3	1	2	3	3	4	3	3	1	1
	4	2	2	4	1	2	3	1	4	2	2	4
Kd Permut. Col. Index	3	1	4	3	4	1	1	1	4	1	1	3
	2	3	3	2	2	3	3	2	4	3	1	3
	1	4	4	4	4	4	3	4	2	4	2	4
	1	2	1	2	2	2	1	3	3	2	1	2
Kd Addition Row Index	3	3	2	3	3	3	4	2	2	4	3	3
	1	4	1	1	1	2	3	2	4	2	1	1
	4	4	4	3	2	3	1	3	3	3	4	3
	2	3	2	3	3	2	2	3	4	2	3	2
Kd Addition Col. Index	2	1	4	1	2	3	1	1	4	1	1	3
	1	1	1	4	1	1	1	2	2	3	2	2
	1	4	1	3	2	3	2	2	3	1	2	2
	2	1	3	1	4	1	3	4	3	4	3	1
Kd Permutation	113	96	73	70	70	71	51	57	29	54	39	116
	32	8	98	31	98	55	118	75	86	49	40	27
	63	67	11	67	2	107	65	23	95	27	12	63
	54	124	37	31	47	50	110	68	55	126	16	14

Kd Addition	53	32	104	6	50	9	98	28	56	109	7	1
	85	62	24	37	21	76	120	24	85	89	22	9
	117	98	65	5	78	45	74	68	107	7	11	70
	116	114	61	21	115	71	103	9	71	8	27	99
NMNT	68	102	119	50	58	60	111	98	46	30	80	68
	81	86	10	36	114	80	41	103	78	16	9	110
	31	18	79	86	11	108	39	27	68	33	41	32
	58	21	42	89	44	74	11	77	78	80	118	8
Accumulated Key	35	98	57	6	12	45	107	63	83	24	72	98
	26	2	30	81	7	91	71	72	129	9	57	121
	46	25	14	52	124	69	108	64	6	120	129	29
	71	98	120	6	74	91	29	102	66	3	70	28
Adding Key	103	73	49	56	70	105	91	34	2	54	25	39
	107	88	40	117	121	44	112	48	80	25	66	104
	77	43	93	11	8	50	20	91	74	26	43	61
	2	119	35	95	118	38	40	52	17	83	61	36

5.4 Algorithm Implementation

The hardware implementation of the algorithm is discussed in detail in Chapter 7. However in this section the processing steps of encryption are explained in two examples of text message and an image using round keys generated in the last section. Tables 5.6 and 5.7 represent the processing steps for encrypting blocks of data, for instance the Plaintext = “Blue Lines” using the round keys listed in Table 5.2. The implementation of the round functions for the first three rounds are explained in detail in Table 5.6 using the actual block size. A summary of the encryption, including the intermediate results for all rounds, is given in Table 5.7. The first row represents the plaintext followed by the ASCII representation of the plaintext. Then the padding process starts by adding 1, as an indication of the end of the message followed by the number 0 so as to achieve the required length. Next, the conversion process converts the input into a number of blocks with *P*-1-bit element length.

Table 5.6: Three rounds encryption using round keys generated from first technique

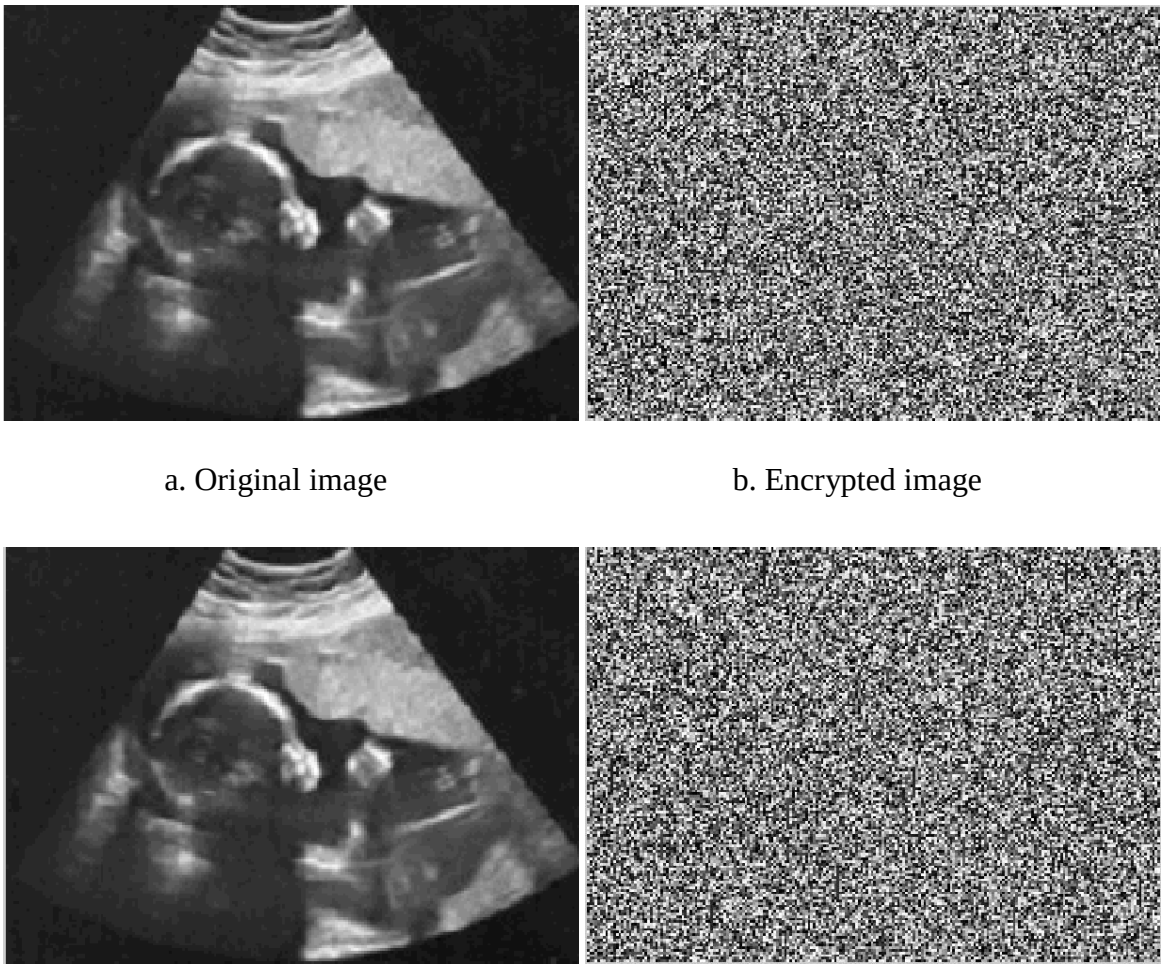
	Round 1				Round 2				Round 3			
Round's Input	50	122	82	33	80	1	47	23	28	33	11	46
	63	19	51	1	24	117	91	31	78	102	119	99
	94	25	70	55	18	36	96	13	17	120	76	24
	123	109	29	5	119	31	61	114	110	99	115	33
Elements Sub.	95	97	44	7	73	124	105	80	41	7	107	58
	33	56	11	124	32	2	27	112	123	87	98	85
	102	37	54	76	5	79	92	8	90	69	21	32
	30	60	113	67	98	112	89	43	52	85	111	7
Shifts Transf.	30	37	11	7	98	79	27	80	52	69	98	58
	95	60	54	124	73	112	92	112	41	85	21	85
	33	97	113	76	32	124	89	8	123	7	111	32
	102	56	44	67	5	2	105	43	90	87	107	7
Variable Add.	30	78	11	7	98	37	27	80	52	19	98	58
	95	11	54	124	73	4	92	112	41	93	21	85
	33	123	113	76	32	104	89	8	123	51	111	32
	102	115	44	67	5	8	105	43	90	52	107	7
NMNT	126	90	83	75	115	28	8	114	100	42	73	120
	30	55	14	27	27	0	49	89	113	28	11	12
	91	94	74	0	106	39	9	101	63	31	24	120
	74	106	91	10	34	119	59	62	2	28	11	65
Round Key	81	38	91	75	40	5	3	59	99	82	108	10
	121	62	77	4	51	102	70	10	59	108	119	70
	54	69	22	13	38	81	67	50	104	65	49	111
	45	52	97	104	76	107	56	98	55	18	67	92
Adding Key	80	1	47	23	28	33	11	46	72	124	54	3
	24	117	91	31	78	102	119	99	45	9	3	82
	18	36	96	13	17	120	76	24	40	96	73	104
	119	31	61	114	110	99	115	33	57	46	78	30

Table 5.7: Encryption processing steps using round keys generated from first technique

1	B	l	u	e		L	i	n	e	s						
2	66	108	117	101	32	76	105	110	101	115						
3	66	108	117	101	32	76	105	110	101	115	1	0				
4	16	38	49	53	25	18	1	12	26	22	57	37	28	48	4	0
5	50	63	94	123	122	19	25	109	82	51	70	29	33	1	55	5
6	80	24	18	119	1	117	36	31	47	91	96	61	23	31	13	114
7	28	78	17	110	33	102	120	99	11	119	76	115	46	99	24	33
8	72	45	40	57	124	9	96	46	54	3	73	78	3	82	104	30
9	39	31	63	74	77	3	0	27	47	124	104	103	99	86	115	72
10	62	96	97	19	90	95	41	26	65	20	28	35	67	90	68	5
11	51	11	5	89	68	124	70	105	116	10	6	96	110	76	0	70
12	83	69	50	110	67	78	116	72	11	118	97	76	104	82	79	53
13	27	76	44	17	114	96	71	69	121	35	24	97	58	67	121	125
14	50	98	39	1	24	48	36	65	25	18	42	45	71	89	5	118
15	15	107	29	67	38	107	70	99	121	89	3	116	84	105	106	32
16	31	172	236	52	218	227	99	243	100	31	74	154	117	32		
17		¬	ì	4	Ú	ã	c	ó	d		J	š	u			
18	Û	=	Ñ	ô	É		}	+		Ú		A	â	Ž		

In this example the message only has one block and, for illustration purposes, the block is represented in one dimension. Row 5 is the result of the initial key addition layer which performs modular addition between the input prepared and the initial key. Data is then processed through successive rounds, where rows 6-15 are the outputs from each round after processing the round functions. In row 16 the output from the previous round is converted from the transform domain into 8-bit for each element. Row 17 is the ASCII representation of the Ciphertext, while the final row is the ciphertext for the same plaintext encrypted with round keys listed in Table 5.4. The decryption process is the same as encryption except that the layers are implemented in reverse order, in addition inverse round functions are required and the round keys are used in reverse order.

Figure 5.3 illustrates a case of image encryption, where the image in Figure 5.3.b is the result of the encryption procedure for the image in Figure 5.3.a. The image in Figure 5.3.c is the recovered image after decrypting the encrypted image using the correct key. The encrypted image is decrypted again using an incorrect key with only one bit changed from the correct key. The image as shown in Figure 5.3.d is completely unrecognizable, reflecting the power of this system.



a. Original image b. Encrypted image

c. Decrypted image with the correct key d. Decrypted image with incorrect key

Figure 5.3: Image encryption and decryption with the correct and incorrect key

5.5 Test Vectors

In this section a number of test vectors are run to verify the correctness of the algorithm implementation. It includes three categories of tests; the KAT, the MMT, and the MCT. These validation tests were initially designed by NIST [106] to validate the implementations under test (IUT) for conformance to the AES algorithm as specified in the FIPS 197: AES [16]. In addition to determining conformance, it is used to detect implementation flaws, including pointer problems, the insufficient allocation of space, improper error handling, and other incorrect behaviour [106]. The complete results of the tests for encryption (which can also be applied to the decryption phase) for block size and key length of $16\times P$, (P = 7) and the ECB mode of operation are listed in Appendix C. Descriptions and sample results for each test are given below.

5.5.1 The Known Answer Test

Two types of KAT are applied:

1. Variable Text

   In this test, the cipher key is fixed to zero for all iterations and the plaintext is variable. It starts at zero and is changed by setting one bit to 1 at each iteration within the bits of the elements from left to right starting from the first element. For instance, in Table 5.8, the first element in the plaintext for the first iteration is set to 64 which is equivalent to 1000000 in binary form and all other elements are zero. At the second iteration only the first element is changed and become 96 which is equivalent to 1100000 in binary form. The first element continues to change at each iteration until it becomes 126, which is equivalent to 1111110 in binary form, and the final bit is not set to 1 due to the modulo. Next, the same procedure is repeated for the second element until all elements have been processed.

   It is obvious from the results of this test that if a single bit in the input is changed, the resulting output (ciphertext-*CT*) is completely different. This reflects the power of the system in diffusing the data and producing totally different outputs for nearly the same inputs, even in the worst cases where most of the elements are zero for both the key and the plaintext. The following is a sample of the test:

Table 5.8: Sample of the variable text known answer test

Kc	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PT	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	52	104	96	49	102	66	37	120	23	8	4	38	104	91	100	20
PT	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	66	87	59	38	14	103	38	50	105	80	1	79	94	126	97	35
PT	112	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	80	29	122	35	21	1	26	6	83	122	93	35	61	120	120	110
PT	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	44	4	50	114	18	13	120	114	29	122	35	50	107	19	43	103
PT	124	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	117	52	0	31	68	33	94	81	22	99	44	72	77	45	78	72
PT	126	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	106	108	77	46	44	71	73	4	109	77	70	5	25	79	121	85
PT	126	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	15	122	74	91	71	97	62	100	16	117	6	14	122	94	98	70
PT	126	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	106	69	110	4	99	90	110	46	68	56	118	82	79	112	81	111
PT	126	112	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	120	29	113	19	8	109	13	54	22	26	85	72	59	121	68	94
PT	126	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	70	103	28	36	97	93	22	108	74	82	54	14	7	124	93	13

2. Variable Key

   This test follows the same procedure as the previous one, replacing the positions of the cipher key and the plaintext. The plaintext is accordingly set to zero for all iterations and the key is variable, the latter being changed based on the procedure explained in the previous test. The results again confirm the power of the system, since changing a single bit in the key results in drastic changes in the output. Below is a sample of the test.

Table 5.9: Sample of the variable key known answer test

PT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Kc	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	114	120	13	82	8	8	59	107	19	13	40	18	46	71	2	3
Kc	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	55	49	36	51	75	92	88	89	81	96	78	106	100	36	101	53
Kc	112	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	53	57	38	69	116	66	50	119	94	71	50	47	31	53	62	57
Kc	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	32	31	29	74	20	9	31	121	81	112	21	102	11	76	57	100
Kc	124	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	114	95	101	119	56	113	37	61	92	19	74	40	114	13	96	72
Kc	126	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	96	102	21	20	74	53	91	125	111	115	64	23	126	107	15	68
Kc	126	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	92	73	35	70	22	93	58	42	0	98	116	106	78	84	97	106
Kc	126	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	97	13	21	13	0	122	99	21	56	110	102	77	33	25	64	23
Kc	126	112	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	78	21	28	95	26	104	8	96	8	26	46	64	86	52	57	65
Kc	126	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	115	118	90	90	42	113	89	31	41	6	26	123	65	59	71	9
Kc	126	124	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	47	84	46	83	19	7	93	52	33	90	16	40	82	92	110	79
Kc	126	126	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	97	78	12	75	2	68	86	59	32	13	105	87	51	117	122	31

5.5.2 The Multi-block Message Test

The MMT is applied to check the capability of the implementation to process messages with many blocks. Here, depending on the mode of operation, a processing between the blocks could be imposed. The test is run on 10 different messages whose lengths are different. The first message consists of one integral block of data, and the length of the second message is expanded into two full blocks of data, so that the length of the message is incremented by full one block of data after each following message. Thus the final message is built up from ten integral blocks of data. The following is a sample data set where the same key is used in the encryption of all of the messages.

Table 5.10: Sample of the multi-block message test

Kc	45	118	71	35	98	63	121	56	72	51	49	104	37	64	89	93

1 Block

PT	49	48	49	119	119	67	117	47	89	50	35	123	110	77	47	94
CT	14	123	66	69	99	84	34	54	35	37	17	116	16	85	14	10

2 Blocks

PT	93	67	72	53	37	87	73	68	83	67	125	64	78	83	77	50
	47	45	73	39	35	48	110	65	62	109	49	50	104	100	97	78
CT	24	6	81	84	41	31	45	31	55	62	1	90	35	35	2	89
	1	26	43	69	76	90	109	62	23	0	98	85	93	25	7	118

3 Blocks

PT	77	80	80	123	44	74	110	77	120	37	75	78	120	112	117	115
	65	41	98	119	53	67	120	81	42	80	114	54	122	84	68	41
	62	78	71	101	49	78	58	47	83	63	68	39	59	87	86	67
CT	48	50	13	116	87	81	60	49	57	21	35	111	114	19	50	81
	114	15	99	6	101	30	42	92	15	120	29	73	42	96	25	92
	124	109	11	16	85	40	30	59	17	36	39	11	0	73	61	10

5.5.3 The Monte Carlo Test

In the MCT 100 pseudorandom texts are encrypted. The initial values of both the key and the plaintext are pseudo randomly generated, and next each text is processed 1000 times, where the output of each iteration is assigned as the input for the next iteration. The final output resulting from processing the text 1000 times is passed on as the next new text, and the new key is generated by XORing its value with the output, so that in total 100000 iterations are processed for all texts.

The basic idea behind the MCT is to give indication that no combinations of irregular inputs exist that participates in terminating the test oddly. In addition, the test verifies that neither the key nor the plaintext would be exposed if the implementation goes wrong. Moreover, any obvious operational errors can be identified. The sample data set for the MCT is shown in Table 5.11.

Table 5.11: Sample of the Monte Carlo test

Count = 1																
Kc	78	88	64	55	85	64	68	101	115	119	59	70	65	77	55	79
PT	74	58	110	97	48	69	103	100	95	83	90	40	126	86	37	48
CT	110	21	88	67	42	123	115	37	51	4	122	56	110	93	2	59
Count = 2																
Kc	32	77	24	116	127	59	55	64	64	115	65	126	47	16	53	116
PT	110	21	88	67	42	123	115	37	51	4	122	56	110	93	2	59
CT	12	55	15	73	114	1	72	42	121	65	109	100	38	22	88	7
Count = 3																
Kc	44	122	23	61	13	58	127	106	57	50	44	26	9	6	109	115
PT	12	55	15	73	114	1	72	42	121	65	109	100	38	22	88	7
CT	61	28	61	28	0	10	61	58	83	64	6	98	0	12	29	5
Count = 4																
Kc	17	102	42	33	13	48	66	80	106	114	42	120	9	10	112	118
PT	61	28	61	28	0	10	61	58	83	64	6	98	0	12	29	5
CT	76	109	26	11	77	31	11	63	27	122	0	32	106	126	83	73

5.6 Algorithm Complexity

The complexity of the algorithm is calculated by counting the total number of operations. The complexity of the forward NMNT for the split-radix fast algorithm can be derived from equations 5.9 and 5.10 for the numbers of multiplication and addition operations, respectively [107].Therefore, the complexity for transform lengths of 4, 8, and 16 is displayed in Table 5.12. Accordingly, the complexity of the overall forward algorithm (for encryption) for different block sizes is shown in Table 5.13.

$$M_N = (2N/3)\log_2 N - (19N/9) + 3 + (-1)^m/9 \tag{5.9}$$

$$A_N = (4N/3)\log_2 N - (14N/9) + 3 + (-1)^m 5/9 \tag{5.10}$$

where: $m = \log_2 N$

Table 5.12: Complexity of the NMNT

N	NMNT	
	A_N	M_N
4	8	0
8	22	2
16	64	12

Table 5.13: Complexity of the overall encryption algorithm

Block Size	Addition (A_N)	Multiplication (M_N)	Shift
16×P	536	0	30
32×P	1272	80	60
64×P	2544	160	70
128×P	6608	960	140
256×P	13216	1920	150

The complexity of the inverse algorithm (for decryption) is exactly the same as the forward algorithm, the only difference is in the number of multiplications, where the transformed elements are multiplied by 1/N. In addition, subtraction is applied instead of addition in the variable and key addition layers.

The number of different S-boxes used in the algorithm depends on the word length. For the default length of $P = 7$ only 7×7 S-boxes are used, but for word lengths greater than 7 both 7×7 S-boxes and 8×8 S-boxes are used.

5.7 Discussion of the System

Flexibility is one of the pillars of the design. For instance, key length, block length and word length are variable and can be set by the user or the system to provide efficient implementation and high levels of security. The default values are $P = 7$ for word length and $16 \times P$ for block and key length. However, if these parameters are set by the system, they are initialised based on the processor register length and message length. These parameters can be encrypted and embedded in a random location depending on the key, making it almost impossible for cryptanalysis to attack the system without prior knowledge of these parameters.

After determining the above-mentioned parameters, a byte with a value of 1 is added at the end of the message. This indication of the end of the massage will be useful in the recovery process. Then the massage is padded if necessary to make its length a multiple of the chosen block size. Next, the plaintext is converted into a number of blocks with word length of P-bit. However, because the modulus is a power of two minus one, the problem may arise that the zero value and the M_p value are retrieved as a zero. This would only occur at the plaintext conversion stage, while all round transformations apply modular arithmetic operations. This problem can be avoided by using the CTR, OFB, or CFB mode as a mode of operation; otherwise, in order to overcome this possible problem two solutions have been proposed. The first solution imposes the insertion of one bit with a value of 0 at the beginning of each element, while the second solution depends on placing the addresses of elements with the value of M_p at the end of the ciphertext. The first method is accomplished by splitting the message into a number of blocks with P-1-bit elements and inserting a bit with a value of 0 at the beginning of each element, so that the element word length becomes P-bit. In this case the padding is based on the P-1-bit. This method guarantees that no elements enter the transform with the value of M_p; however, the actual data transfer in the block is a function of P, as shown in equation 5.11. For instance, for $P = 7$, the actual data transfer represents 85.7% of the total block size, and for $P = 31$ it is 96.8%. The second method is based on placing the

element addresses that have the value of M_p at the end of the message, in order to update their values at the final stage of the decryption phase.

$$ADT = (\frac{p-1}{p}) \times 100\% \tag{5.11}$$

where *ADT* is the actual data transferred.

At this stage the blocks are ready for processing, which starts by the initial key addition followed by a round transformation that runs *rd* times. The layers of the round functions are selected very carefully to ensure high diffusion throughout the rounds. After element substitution, a shifts transform is applied in order to shuffle the elements such that each resulting row holds elements from all rows. Then variable values are added to the elements at location *N*/4+1 from each row in the array in order to further complicate the correlation between the related elements and to eliminate cases that provide low diffusion in the transform so as to provide higher diffusion. Next, the NMNT is applied to each row individually to mix all row elements, where each output element is a function of all input elements. Finally, a round key is added, which is derived from the key schedule algorithm.

Using the transform accompanied by other layers in the process of generating round keys excludes the possibility that two different user-supplied keys may yield the same round keys. In addition, adding round numbers [3] will eliminate symmetries and thus prevent potential weak-key and related-key attacks.

For CTR, OFB and CFB modes, the decryption procedures are applied in exactly the same as those for encryption, while for other modes of operation; they are applied in the reverse order. In addition, inverse functions are used instead, and the round keys are utilised in the reverse order.

The default value for word length is appropriate for different platforms, such as on 8-bit processor. For larger lengths of processor register, such as in the case of 32-bit, it is possible to retain the default value or utilise larger values of *P* such as *P* = 31, so as to increase the actual data transfer at each block and at the same time reduce the complexity which results from using a larger block size for the same amount of data. Accordingly, a larger amount of data is encrypted with higher levels of security and lower complexity. Likewise, for a 64-bit processor, an even larger *P* can be used, i.e. 61.

5.8 Algorithm Based on FNT

The other proposed algorithm is exactly the same as the one proposed earlier regarding its structure in both directions and the generation of the keys; the only difference is in the transform that is used to provide local diffusion. In this algorithm the FNT is used instead of the NMNT and all other layers remain unchanged. The FNT is explained in detail in Chapter 3, and from the levels of diffusion power found in analysis it has been proven that the transform has good diffusion power [98].

The integer t is chose to be 3, and therefore the modulus is $F_t = 2^{2^t} + 1 = 257$ and word length $P = 2^t + 1 = 9$ bits. The selection of these parameters has a clear advantage during the implementation of the algorithm. The need for the conversion of elements at the beginning and end of the algorithm is eliminated, hence reducing the complexity resulting from this layer.

The complexity of the forward transform using the split-radix fast algorithm is shown in equations 5.12 and 5.13 for the number of multiplications and additions, respectively [108]. Accordingly, the numbers of multiplications and additions for transform lengths 4, 8, and 16 are shown in Table 5.14. Consequently, the complexity of the overall forward algorithm (encryption) for different block sizes is shown in Table 5.15. By setting the value of the kernel α to 2 or a power of 2, the multiplication operation can be implemented by shift and addition operations.

$$M_N = (N/2)\log_2 N - (3N/2) + 2 \tag{5.12}$$

$$A_N = (3N/2)\log_2 N - (5N/2) + 4 \tag{5.13}$$

Table 5.14: FNT complexity

FNT		
N	A_N	M_N
4	6	0
8	20	2
16	60	10

Table 5.15: Complexity of the overall encryption algorithm (FNT)

Block Size	Addition (A_N)	Multiplication (M_N)	Shift
16×P	456	0	30
32×P	1192	80	60
64×P	2384	160	70
128×P	6288	800	140
256×P	12576	1600	150

The round keys generated by applying the key schedule algorithms are listed in Tables 5.16 and 5.17 for the first and second proposed techniques, respectively. The cipher key is “Dell Laptop”, which is used earlier in the example of the algorithm that is based on NMNT.

A summary of encryption, including the intermediate results for all rounds, is given in Table 5.18 and the detailed implementation of the round transformation covering all round functions for the first three rounds is shown in Table 5.19 using the round keys listed in Table 5.16. The implementation is exactly the same as that with NMNT; the only difference is in eliminating the rows related to word length conversion. Row 16 in Table 5.18 is the ciphertext for the plaintext “Blue Lines” in row 1, while row 17 is the ciphertext for the same input encrypted with the round keys generated in the second technique as listed in Table 5.17.

Table 5.16: Round keys generation using first technique based on FNT

K_c	68	101	108	108	32	76	97	112	116	111	112	180	25	133	241	17
K_{r1}	238	25	137	161	121	198	66	42	187	148	7	67	19	162	146	214
K_{r2}	95	142	8	249	157	92	214	98	172	207	65	45	76	167	198	157
K_{r3}	206	61	256	55	187	41	82	227	230	70	155	119	22	174	247	225
K_{r4}	163	47	43	9	71	249	28	78	103	116	96	237	190	12	88	222
K_{r5}	133	256	74	137	226	211	205	179	207	190	107	52	137	193	56	154
K_{r6}	8	65	5	177	249	146	72	62	34	178	193	159	241	80	16	28
K_{r7}	227	138	75	54	182	169	171	227	201	130	206	167	76	93	138	237
K_{r8}	138	209	104	62	100	131	145	151	1	48	196	160	100	38	205	4
K_{r9}	11	71	115	253	226	12	113	112	71	184	109	36	38	254	197	174
K_{r10}	26	195	103	253	37	137	78	27	160	215	203	33	97	167	52	109

Table 5.17: Round keys generation using second technique based on FNT

K_c	68	101	108	108	32	76	97	112	116	111	112	180	25	133	241	17
K_{r1}	207	224	189	40	43	242	147	72	190	93	1	112	74	212	26	140
K_{r2}	45	29	83	65	129	150	43	221	169	179	173	32	78	197	124	48
K_{r3}	158	157	114	141	109	91	174	155	77	55	94	71	116	112	198	64
K_{r4}	52	23	214	48	255	54	91	173	228	121	180	66	248	25	147	43
K_{r5}	146	126	79	1	173	40	209	58	217	31	90	204	87	186	3	63
K_{r6}	179	249	99	95	135	113	100	208	11	157	72	171	116	21	180	204
K_{r7}	34	152	256	4	149	70	43	63	185	131	162	122	51	66	154	21
K_{r8}	110	26	98	29	118	34	233	93	210	227	177	40	165	51	243	60
K_{r9}	229	254	153	137	71	39	144	75	188	128	67	122	173	254	39	233
K_{r10}	111	37	53	104	248	150	150	48	138	50	22	112	22	7	38	117

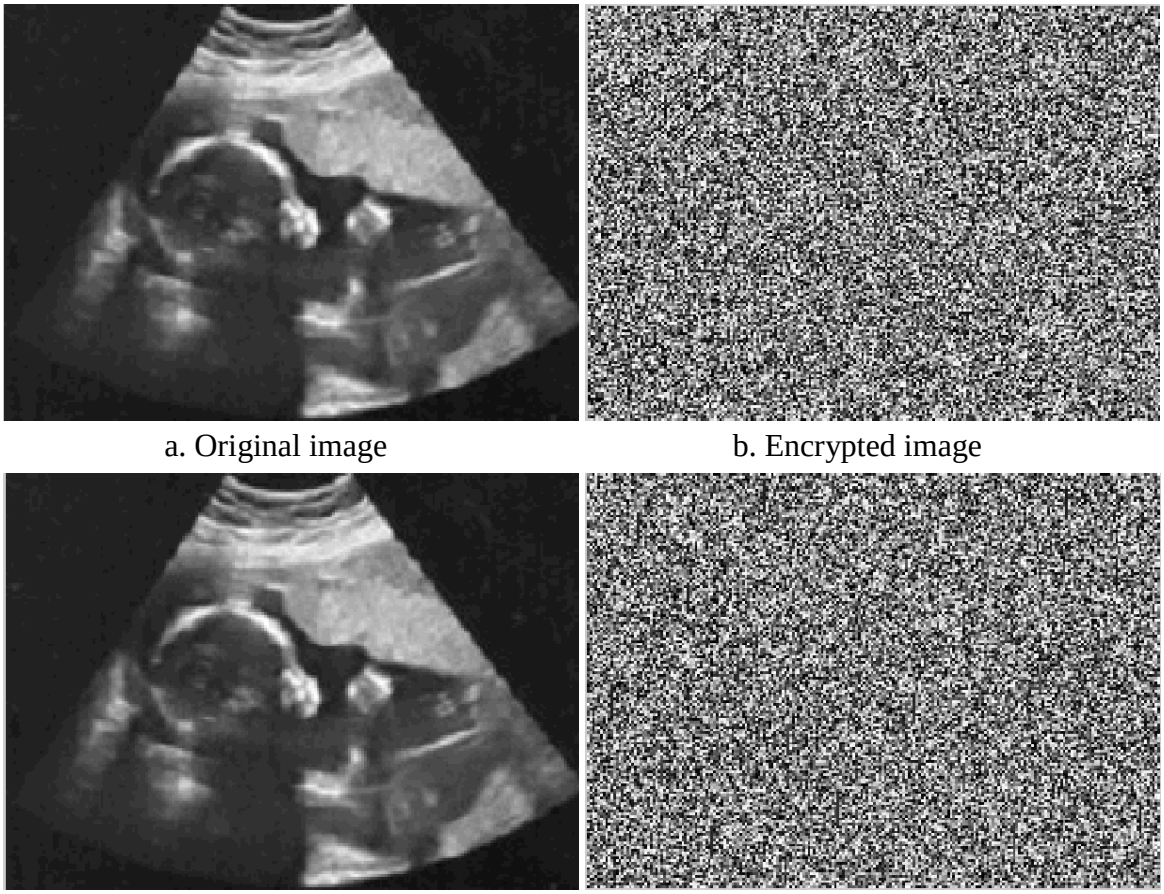
Table 5.18: Encryption step for algorithm based on FNT

1	B	l	u	e		L	i	n	e	s						
2	66	108	117	101	32	76	105	110	101	115						
3	66	108	117	101	32	76	105	110	101	115	1	0	0	0	0	0
5	134	209	225	209	64	152	202	222	217	226	113	180	25	133	241	17
6	129	226	61	91	158	188	137	164	210	152	232	225	59	239	174	225
7	17	198	254	149	169	32	124	7	247	228	105	71	38	198	96	221
8	126	36	105	6	153	34	239	2	90	82	215	236	187	200	130	102
9	193	69	245	22	136	88	139	170	38	94	157	74	90	117	187	248
10	102	25	11	248	84	172	42	193	42	78	47	124	245	27	11	106
11	51	136	152	226	27	214	166	99	215	251	245	140	242	72	57	133
12	105	233	91	137	49	197	94	61	143	114	11	252	226	252	150	1
13	201	114	1	174	104	78	213	48	143	213	197	135	45	246	102	248
14	227	164	108	6	238	190	102	31	60	94	33	247	219	186	33	16
15	20	227	127	87	112	107	40	104	50	209	19	58	68	239	28	236
16		ã		W	p	k	(	h	2	Ñ		:	D	ï		ì
17	M	‘		?	w	Đ	«		š	8	[	Õ	È		,	Š

Table 5.19: Three rounds encryption based on FNT using keys from first technique

	Round 1				Round 2				Round 3			
Round's Input	134	64	217	25	129	158	210	59	17	169	247	38
	209	152	226	133	226	188	152	239	198	32	228	198
	225	202	113	241	61	137	232	174	254	124	105	96
	209	222	180	17	91	164	225	225	149	7	71	221
Elements Sub.	68	9	53	212	12	11	181	226	130	211	104	247
	62	70	152	151	152	101	70	223	180	183	105	180
	248	116	163	161	39	167	155	228	187	16	249	208
	62	29	141	130	57	73	248	248	42	197	160	193
Shifts Transf.	62	116	152	212	57	167	70	226	42	16	105	247
	68	29	163	151	12	73	155	223	130	197	249	180
	62	9	141	161	152	11	248	228	180	211	160	208
	248	70	53	130	39	101	181	248	187	183	104	193
Variable Add.	62	236	152	212	57	83	70	226	42	40	105	247
	68	76	163	151	12	180	155	223	130	187	249	180
	62	74	141	161	152	132	248	228	180	72	160	208
	248	13	53	130	39	203	181	248	187	238	104	193
NMNT	148	37	23	40	179	12	75	219	177	223	117	165
	201	247	4	77	56	197	21	31	232	250	12	26
	181	71	225	28	246	167	40	155	106	157	60	140
	187	122	158	11	157	166	26	64	208	32	117	134
Round Key	238	121	187	19	95	157	172	76	206	187	230	22
	25	198	148	162	142	92	207	167	61	41	70	174
	137	66	7	146	8	214	65	198	256	82	155	247
	161	42	67	214	249	98	45	157	55	227	119	225
Adding Key	129	158	210	59	17	169	247	38	126	153	90	187
	226	188	152	239	198	32	228	198	36	34	82	200
	61	137	232	174	254	124	105	96	105	239	215	130
	91	164	225	225	149	7	71	221	6	2	236	102

The algorithm can also be used efficiently to encrypt images, Figure 5.4 shows an example of the encryption of a medical image, where the image in Figure 5.4.b is the result of the encryption of the image in Figure 5.4.a, and the image in Figure 5.4.c is the decrypted image which is exactly the same as the original using the correct key. While in Figure 5.4.d the encrypted image is decrypted with the correct key. However, one of the key bits is flipped, and the decrypted image is totally different from the original.



a. Original image b. Encrypted image

c. Decrypted image with the correct key d. Decrypted image with incorrect key

Figure 5.4: Image encryption/decryption with correct/incorrect key based on FNT

5.9 Conclusions

A new iterated symmetric-key block cipher based on the NTT has been designed and a number of test vectors have been run to verify its correctness. The flexibility of the design makes the cipher suitable for implementation on different platforms and levels of security required. The cipher has a variable block size and key length, with a size range

of (16, 32, 64, 128, or 256)×*P*, where the word length *P* can be 7, 31, 61, or 127. The possibility that the system can secretly select an appropriate block size and word length increases the security of the overall system by hiding those parameters from the attacker. The cipher is fully parameterised for word length, key length and block size. Even the number of rounds, although identified here as ten, it can be changed to adhere to security requirements, so as to ensure practical usage for the proposed lifetime. The general structure of the cipher is based on the SPN, which alternately implementing confusion layers represented by the S-boxes and diffusion layers which process the NTT to provide a high local diffusion, in addition, to the shifts transformation to disperse the elements.

Chapter 6

Cryptanalysis

In this chapter the main types of cryptanalytic attacks, namely those arising from the use of differential and linear cryptanalysis, are described and considered in relation to the proposed NMNT based cipher. The chapter is organised as follows: after the introduction a classification of types of attacks is presented in section 6.2. Then, in section 6.3 the criteria of attacks are listed while in sections 6.4 and 6.5 general descriptions of differential and linear cryptanalysis are given, respectively. Section 6.6 discusses in detail the differential and linear cryptanalytic attacks on the proposed algorithm. Sections 6.7-6.9 explain related-key, slide and brute-force attacks, respectively. Weak keys are discussed in section 6.10, and finally the conclusions of the chapter are presented in section 6.11.

6.1 Introduction

The objective of an attack is to recover a plaintext from a given ciphertext without prior knowledge of a key, or it may attempt to find a cipher key in order to recover all ciphertexts encrypted with this key. Generally, a system is considered broken if the cryptanalyst can succeed in recovering the plaintext having used less computational

effort than that required for an exhaustive key search, even if this success is only theoretical, since as unfeasible amount of data, memory or time could be required. Therefore, the cryptanalyst's efforts are focused on analysing the cipher in order to search for possible weaknesses that could be exploited in attacking it at a level of complexity less than an exhaustive key search [109].

Two well-known powerful types of attacks can be considered: differential and linear cryptanalysis. Differential cryptanalysis [43] is a chosen-plaintext attack which mainly observes the differences in ciphertext pairs as a function of a particular difference in a plaintext pair. This can then be exploited to allocate probabilities to potential keys. On the other hand, linear cryptanalysis [56] is a known-plaintext attack which tends to construct an expression of linear approximations for a given algorithm, resulting in a high probability of estimating one or more of the key bits.

Consideration of these two types of attack plays a crucial role in the design and development of the infrastructure of the cipher ranging from selecting the components and their order to producing the final model. The possibility of other types of attacks may later require a minor modification to the final design [3] or the addition of extra rounds [110].

Resistance to differential and linear cryptanalysis have now almost become benchmarks in assessing the robustness of an algorithm. Hence, for any suggested algorithm, a verification of robustness against such attacks is considered essential.

6.2 Classification of Attacks

In chapter 2, attacks were classified on the basis of what resources are available to the cryptanalyst, considering that the cryptanalyst has full details of the algorithm. In this section attacks are classified in another way, based on additional information related to the physical properties of the implementation. So, in general, attacks can be classified into three categories as follows:

1. Brute-force attacks

   This type of attack can be applied to any block cipher regardless of its strength. It does not depend on the internal structure of the cipher, but instead the attacker considers the cipher as a black box. The complexity of such attacks depends on the length of the key. The attack, also known as an exhaustive key search, requires one or a few known plaintext-ciphertext pairs encrypted with the same key. The key will be recovered by trying every possible key until the correct text

is obtained. In the worst cases, all possible keys would be checked, i.e. 2^{nk}, where *nk* is the number of key bits, and on average half of all possible keys are tried until the key is recovered, 2^{nk-1}. Therefore the key space of the system should be large enough to frustrate this attack, and this represents the upper bound for the security of the system. The lower bounds for computationally equivalent key sizes for different cryptosystem have been recommended with an acceptable security margin for the years ranging from 1982 to 2050 [111], as shown in Table 6.1. The authors of this study predicted that the computational complexity required to attack an 86-bit symmetric key in 2020 would be considered impracticable, as was the case with the DES in 1982.

Table 6.1: Equivalent key sizes

Year	Symmetric Key Size	Classical Asymmetric Key Size	Elliptic Curve Key Size
1982	56	417	105
1990	63	622	117
2000	70	952	132
2010	78	1369	146
2020	86	1881	161
2030	93	2493	176
2040	101	3214	191
2050	109	4047	206

Despite the recommended key lengths cited in Table 6.1 [111], current cryptosystems have larger key spaces, providing more security. In the current standard the AES, the key space is 128-bit and it can support 192 and 256-bit.

Table 6.2 shows the time required to recover a key for different algorithms and various key spaces as calculated in a subsequent study [12]. The results in column 3 are calculated by assuming that each decryption is performed in 1 microsecond, whereas the results in the right-hand column consider that the key can be recovered by processing 1 million decryptions per microsecond, assuming that the operations are performed in parallel with access to a network with a huge number of machines. It is clear from the results that the DES algorithm can currently be subject to exhaustive key search attacks and,

accordingly, it is no longer considered secure. The other calculations are for the AES and for the proposed algorithm, where for the latter the results are computed for a default word length of $P = 7$. For longer words even longer times are obtained.

Table 6.2: Require time for exhaustive key search

Algor.	Key size (bits)	Key Space	Time required at 1 decryption/μs	Time required at 10^6 decryption/μs
DES	56	$2^{56} = 7.2 \times 10^{16}$	2^{55} μs = 1142 years	10.01 hours
AES	128	$2^{128} = 3.4 \times 10^{38}$	2^{127} μs = 5.4×10^{24} years	5.4×10^{18} years
	192	$2^{192} = 6.3 \times 10^{57}$	2^{191} μs = 9.9×10^{43} years	9.9×10^{37} years
	256	$2^{256} = 1.2 \times 10^{77}$	2^{255} μs = 1.8×10^{63} years	1.8×10^{57} years
NMNT	112	$2^{112} = 5.2 \times 10^{33}$	2^{111} μs = 8.2×10^{19} years	8.2×10^{13} years
	224	$2^{224} = 2.7 \times 10^{67}$	2^{223} μs = 4.3×10^{53} years	4.3×10^{47} years
	448	$2^{448} = 7.3 \times 10^{134}$	2^{447} μs = 1.2×10^{121} years	1.2×10^{115} years
	896	$2^{896} = 5.3 \times 10^{269}$	2^{895} μs = 8.4×10^{255} years	8.4×10^{249} years
	1792	$2^{1792} = 2.8 \times 10^{539}$	2^{1791} μs = 4.4×10^{525} years	4.4×10^{519} years

2. Shortcut attacks

   Shortcut attacks are concerned with the internal structure of the cipher. The attacker mathematically analyses the interior components of the cipher to diagnose undesirable probabilistic characteristics that can be exploited in attacking the system in an attempt to deduce the key being used. The well-known attacks of differential and linear cryptanalysis fall into this category.

3. Side-Channel attacks

   This attack is also known as an implementation attack, exploiting the physical properties of the cipher's implementation in addition to its mathematical characteristic. For instance, a timing attack [112] investigates the execution time of the algorithm in order to derive the key. This type of attack is applicable to a cipher if the execution time is not the same for all processed data, but is a function of the data itself and/or the key. Another type of such an attack is power

analysis [113] which derives key information by considering the power consumption of the machine that is processing the encryption algorithm.

6.3 Criteria of Attack Success

The success of an attack is evaluated based on three criteria: the time, memory and data complexity required to conduct the attack successfully [114].

a. Time complexity refers to the number of computational steps required to attack an algorithm successfully.
b. Memory complexity concerns the amount of memory necessary for processing the attack on an algorithm.
c. Data complexity is the number of data items of known, chosen or adaptively chosen plaintext and/or ciphertext required by the cryptanalyst to attack the algorithm effectively.

Accordingly, an attack is considered successful and the cipher is considered broken if the key can be recovered in a period of time shorter than would be required for an exhaustive key search, i.e. requiring time complexity of significantly less than 2^{nk}, where *nk* denotes the number of key bits. On the other hand, the cipher is considered partially broken if part of the plaintext is revealed in the same time period.

In a similar manner, the cipher can be characterised by encrypting all different plaintexts with the same key, this lead to the determination of an upper bound of 2^{nb} for the data complexity to a successful attack, where *nb* is the block length in bits.

According to the above, an attack is considered successful if it requires a time complexity of significantly less than 2^{nk} or a data complexity of significantly less than 2^{nb} [114].

It is also possible to assess the robustness of an algorithm by quoting the maximum number of rounds an attacker has successfully broken the reduced-round version of the cipher. For instance, the AES algorithm has been attacked with a maximum of 7 rounds for reduced-round version AES-128 with 2^{110} time complexity and 2^{106} data complexity [49], a maximum of 8 rounds for reduced-round AES-192 with 2^{185} time complexity and 2^{126} data complexity [50], and also 8 rounds for reduced-round AES-256 with 2^{209} time complexity and 2^{126} data complexity [50].

6.4 Differential Cryptanalysis

Differential cryptanalysis is one of the most powerful types of attack against block ciphers, and was initially proposed by Biham and Shamir in 1990 [43] as a method of attacking the DES algorithm and a variety of DES-like cryptosystems.

Differential cryptanalysis is a chosen-plaintext attack which uses only the resultant ciphertext to derive the key; it requires a large number of plaintext-ciphertext pairs with a particular input difference to allocate the right values to the key bits. The technique investigates the consequence of certain differences in plaintext pairs on differences in the resulting ciphertext pairs, which can be exploited to assign probabilities to the values and locations of possible key bits.

The mechanism used in the attack can be described as follows. Firstly, an n-round characteristic with high probability has to be built. An n-round characteristic represents the values of difference gathered from XORing a pair of plaintexts, then XORing its resultant ciphertexts, and finally XORing its inputs to each round and its outputs from each round. The probability of the characteristic is then the probability that a plaintext pair with a particular difference has the round and ciphertext differences represented in the characteristic. Secondly, from the characteristics an intermediate difference is assumed at the input of the last round (in general) for each pair. This value along with the absolute value of the output pairs determines the number of key bits to be counted. If the assumed value and the absolute value match, then this pair is called the 'right pair', which suggests a number of right and wrong subkey values. The number of suggested values differs from pair to pair. Each suggested value is assigned to a counting table and, after processing a sufficient number of pairs, the right subkey can be identified since its value is suggested more frequently. Those pairs whose values of absolute output do not match the assumed values are called 'wrong pairs', and usually their suggested values are incorrect as they suggest random values.

An essential requirement for this attack to be successful is to find a high probability characteristic, in addition to provide a sufficient number of ciphertext pairs.

Differential cryptanalysis has succeeded in breaking the full DES version with a complexity of 2^{47} for chosen plaintext, which is less than that needed in an exhaustive key search which required in the worst case 2^{56} steps [40]. The attack also succeeded in breaking a 4-round reduced version of the current standard AES with a time complexity of 2^{40} [115].

6.5 Linear Cryptanalysis

Linear cryptanalysis [56] is a known-plaintext attack. It approximates the cipher in a linear manner by constructing expressions of an effective linear approximation for the entire algorithm with a high probability of identifying one or more of the key bits. The expression of linear approximation that the attacker seeks to construct is shown in equation 6.1. For a successful attack, the bias for a given expression should be large enough, that is, the probability should be away from 1/2. The bias, which represents the effectiveness of equation 6.1, is shown in equation 6.2.

$$PT[\oplus_p] \oplus CT[\oplus_c] = K[\oplus_k] \tag{6.1}$$

$$\epsilon = |P_r - 1/2| \tag{6.2}$$

where *PT*, *CT* and *K* stand for the plaintext, ciphertext and key, respectively. $[\oplus_x]$ denotes the bitwise XORing of a number of bits in fixed bit locations. ϵ and *Pr* represent the bias and probability of the expression.

Two algorithms were proposed in a previous study [56] for identifying the key bits from given expressions of a linear approximation holding a high probability. The concept of the two algorithms is the same where the algorithm counts the number of pairs from the total number of pairs that makes the left side of equation 6.1 equal to zero. Then the bit value of the key can be deduced based on the maximum likelihood method. In the second algorithm more than one bit from the key bits can be deduced resulting in practice in a more efficient algorithm.

Linear cryptanalysis has succeeded in breaking the full DES cipher by finding 14-bit of the key using 2^{47} known-plaintexts, where the remaining key bits can be recovered completely at a level of complexity less than that needed for an exhaustive key search.

For the AES algorithm, it has been concluded [116] that linear cryptanalysis can be effective in deducing key bits only for the reduced-round version and up to the first three rounds, due to the small bias of its S-box.

6.6 Differential and Linear Cryptanalytic Attacks

The nature of differential and linear cryptanalysis, as explained in sections 6.4 and 6.5 has been taken into consideration in the present design. The robustness of the algorithm against these two types of attacks has been examined based on the calculation of two main parameters: the non-linear properties of all of the S-boxes involved; and the lower bounds for the number of active S-boxes at each round. The S-box is considered active if its input difference pattern or output selection pattern are non-zero.

The proposed cipher is designed based on the wide trail design strategy [3]. This ensures that both the maximum probability of the differential characteristics and the IOC_{max} of the linear characteristics are low. To achieve this, it is important to design an S-box with highly non-linear properties, such that the DPP_{max} and the IOC_{max} are both at a minimum, in addition to finding a mechanism to maximise the number of active S-boxes [3].

In order to build a system which is secure against differential cryptanalysis, the probability of the characteristic should be smaller than 2^{1-nb}, where *nb* is the block size in bits. To achieve this the number of rounds should be chosen in such a way that there are no differential characteristics with a weight lower than *nb* [3], where a differential characteristic consists of a sequence of difference patterns whose weight represents the sum of the weights of all patterns concerned. In other words, the weight of the differential characteristic can be computed by summing the weights of all active S-boxes involved. This can be reformulated as the result of multiplying the number of active S-boxes and the minimum differential weight per S-box.

Furthermore, the system can resist linear cryptanalysis if the amplitude of the input-output correlation is smaller than $2^{-nb/2}$. This can be achieved by increasing the number of rounds such that there are no linear characteristics with a correlation over $n_k^{-1}2^{-nb/2}$ [3], where n_k stands for the key length in bits. Similarly to the differential characteristic, a linear characteristic consists of a sequence of patterns whose weight represents the sum of the weights of all patterns concerned. The weight of the linear characteristic can be computed by summing the weights of all active S-boxes involved. Therefore, it is the result of multiplying the number of active S-boxes and the minimum correlation weight per S-box.

Therefore, to construct a secure system, it is required first to count the number of rounds necessary to make the system secure against differential and linear cryptanalysis, and then to add a few additional rounds to guard against other possibly attacks and for

security margins. For instance, the analysis of the security of the AES algorithm with a 128-bit key length indicated that the algorithm is secure against differential and linear cryptanalysis from round four; however a saturation attack [117] can break the reduced-rounds versions up to six rounds. Hence, additional rounds are added that provide double full diffusions as a security margin, making in total 10 rounds.

In order to determine the number of rounds necessary to make the cipher secure against differential and linear cryptanalysis, it is first required to calculate the non-linear properties of all S-boxes involved. This can be achieved, as explained in Chapter 4, by building up the XOR distribution and linear approximation tables. From these, the DPP_{max} and the IOC_{max} can be calculated. These two values have already been computed in Chapter 4 for the S-boxes concerned, which are the 7×7 S-box and the 8×8 S-box. For the 7×7 S-box, the $DPP_{max} = 2^{-6}$ and the $PB_{max} = 2^{-3.678}$ or the $IOC_{max} = 2^{-2.678}$. For the 8×8 S-box, the $DPP_{max} = 2^{-6}$ and $PB_{max} = 2^{-4}$ or the $IOC_{max} = 2^{-3}$.

In the proposed cipher the arithmetic operations are performed modulo M_p, because the value of the modulus is 2^P-1, hence there is a possibility that a value of 2^P will appear after mapping the elements through the S-boxes. Therefore, to avoid such a possible problem from arising, all S-boxes involved are modified so that the maximum value is swapped to map to itself in order to prevent its appearance after mapping.

The swapping in the 7×7 S-box and its inverse is as follows:

```
s_box(82) = 44;          inv_s_box(44) = 82;
s_box(127) = 127;        inv_s_box(127) = 127;
```

For 8×8 S-box and its inverse the swapping is as follows:

```
s_box(125) = 22;         inv_s_box(22) = 125;
s_box(255) = 255;        inv_s_box(255) = 255;
```

Although one element is only swapped at each S-box, the non-linear properties then need to be recalculated since this change could have a positive or negative effect on those properties. Accordingly, the XOR distribution and linear approximation tables for both the 7×7 S-box and the 8×8 S-box have been rebuilt, and the DPP_{max} and the IOC_{max} are recalculated. For the 7×7 S-box, the DPP_{max} is 2^{-5}, or in weight form it is equal to 5, observing that the weight is the negative of the binary logarithm of DPP_{max}. The IOC_{max} from the linear distribution table is 20/128, equivalent to 2.678 in weight form. For the

modified 8×8 S-box, the DPP_{max} is 6/256, or in weight form equal to 5.415, and the IOC_{max} is 36/256, equivalent to 2.83 in weight form. For larger values of *P*, the weight is calculated by summing the weights of the S-boxes involved. It is clear from the results that the swapping of elements has slightly reduced the non-linear properties of the S-boxes. This, however, cannot be avoided yet at the same time the resulting non-linear properties can still be considered to be good.

Once the non-linear properties have been calculated, it is then possible to compute the minimum number of active S-boxes required to make the system secure against differential and linear cryptanalysis. This can be achieved directly using equations 6.3 and 6.4, respectively, which are derived from [3]. The minimum number of active S-boxes required for different block sizes is shown in Table 6.3:

$$\beta_{min}(DC) = \lceil (nb - 1)/W(DPP_{max}) \rceil \tag{6.3}$$

$$\beta_{min}(LC) = \left\lceil -log_2\left(n_k^{-1}2^{-nb/2}\right)/W(IOC_{max}) \right\rceil \tag{6.4}$$

where B_{min} is the minimum active S-boxes for differential (DC) and linear (LC) cryptanalysis, *nb* is the block size, *nk* is the key size, and *W* is the weight of the DPP_{max} and IOC_{max}.

The next step is to determine the lower bounds for the number of active S-boxes at each round; this can be achieved by using equation 3.18. This step is implemented by processing only the linear parts of the algorithm in addition to excluding the initial key addition and the round key addition layers. The simulation results for an input weight of 2 and for different block sizes are shown in Table 6.4. The input weight of 2 was chosen since it produces the worst level of diffusion.

By comparing the results in Table 6.4 with those in Table 6.3, it can be concluded that the algorithm is secure against differential and linear cryptanalysis from round three for all block sizes.

The simulation results also confirm that no 3-round differential characteristic and linear characteristic exists with a probability greater than the approximate results listed in Table 6.5 for different block sizes and word lengths.

Table 6.3: Minimum number of active S-boxes required

Block Size	Minimum number of active S-boxes required to make the system secure against:	
	Differential cryptanalysis	Linear cryptanalysis
16×*P*	23	24
32×*P*	45	45
64×*P*	90	87
128×*P*	179	171
256×*P*	359	339

Table 6.4: The lower bounds of the number of active S-boxes per round

Round Number	Block Size				
	16×*P*	32×*P*	64×*P*	128×*P*	256×*P*
1	2	2	2	2	2
2	10	18	50	98	226
3	26	50	112	223	477
4	42	81	174	350	728
5	57	113	238	477	947
6	73	144	302	603	1197
7	88	176	365	729	1449
8	104	208	429	855	1701
9	120	239	493	980	1951
10	135	271	557	1105	2204

Table 6.5: 3-Round differential and linear characteristics

Block Size	*P* = 7		*P* = 31		*P* = 61		*P* = 127	
	DPP	IOC	DPP	IOC	DPP	IOC	DPP	IOC
16 × *P*	2^{-130}	2^{-70}	2^{-552}	2^{-290}	2^{-1094}	2^{-577}	2^{-2242}	2^{-1173}
32 × *P*	2^{-250}	2^{-134}	2^{-1062}	2^{-558}	2^{-2104}	2^{-1109}	2^{-4311}	2^{-2256}
64 × *P*	2^{-560}	2^{-300}	2^{-2379}	2^{-1251}	2^{-4712}	2^{-2485}	2^{-9657}	2^{-5054}
128 × *P*	2^{-1115}	2^{-597}	2^{-4738}	2^{-2490}	2^{-9383}	2^{-4947}	2^{-19228}	2^{-10064}
256 × *P*	2^{-2385}	2^{-1277}	2^{-10134}	2^{-5327}	2^{-20070}	2^{-10582}	2^{-41129}	2^{-21526}

To evaluate the results obtained, the same procedures are followed with the Rijndael AES algorithm, Table 6.6 represents the number of active S-boxes necessary to make the Rijndael AES secure against differential and linear cryptanalysis, and Table 6.7 gives the simulation results for the lower bounds of the number of active S-boxes at each round, for an input weight equal to 2. By comparing the results in Tables 6.6 and 6.7, it can be concluded that, for a block size of 128-bit, four rounds is enough to secure the algorithm from these two attacks, while for a block size of 256 a further round is required. This is the reason for increasing the number of rounds with larger key or block sizes with the Rijndael AES, which is not the case with the proposed algorithm.

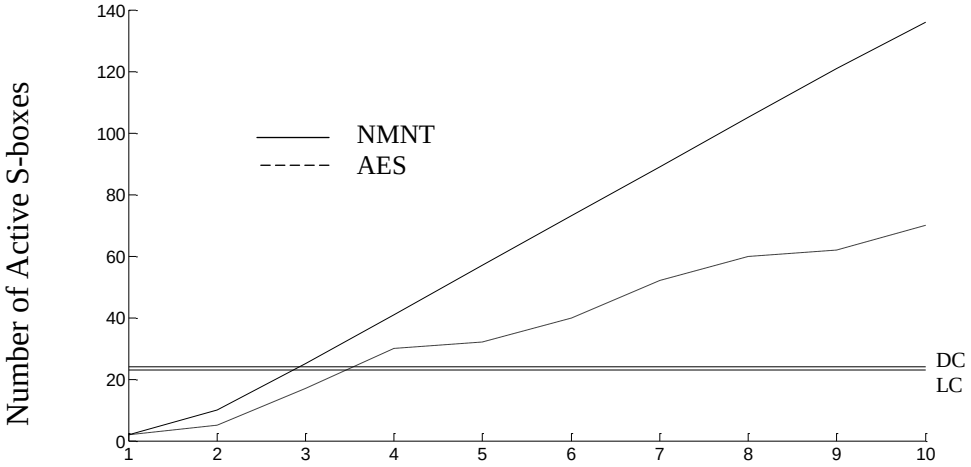
Table 6.6: Minimum number of active S-boxes required for Rijndael AES

Block Size	Minimum number of active S-boxes required to make the system secure against:	
	Differential cryptanalysis	Linear cryptanalysis
128	22	24
256	43	46

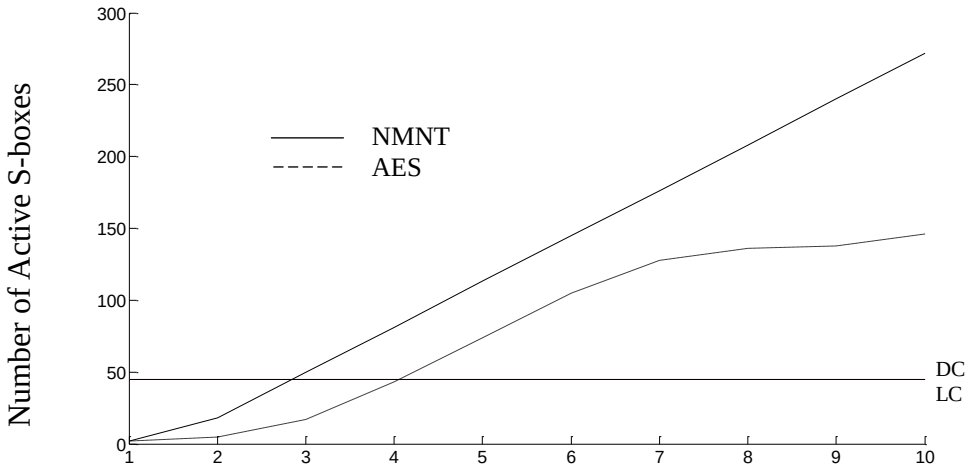
Table 6.7: The lower bounds of the number of active S-boxes per round for Rijndael AES

Round Number	Block Size	
	128	256
1	2	2
2	5	5
3	17	17
4	30	43
5	32	74
6	40	105
7	52	128
8	60	136
9	62	138
10	70	146

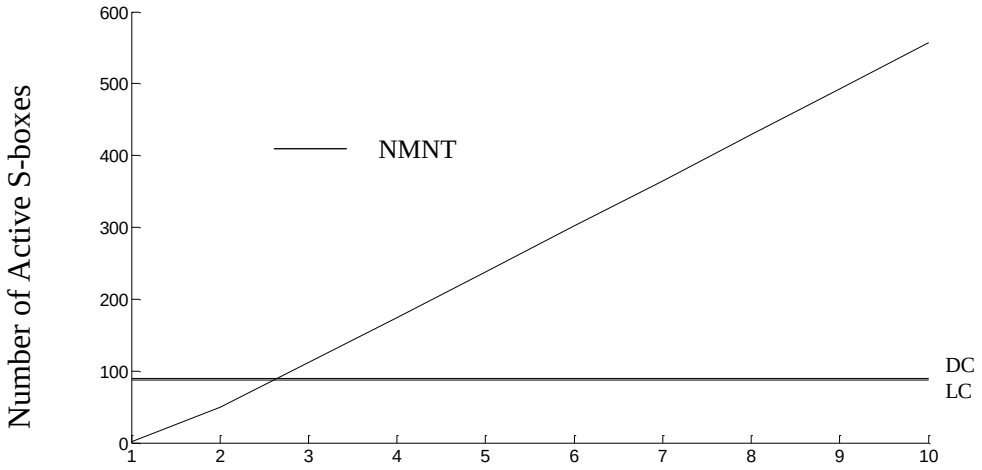
In Figure 6.1 the results from both the proposed algorithm and the Rijndael AES algorithm for all block sizes are represented graphically. The horizontal lines represent the boundaries for differential and linear cryptanalysis, which are nearly the same for both algorithms. The system is considered secure when the curves pass these boundaries. It is clear from the figure that the proposed algorithm exhibits good diffusion and is secure against both differential and linear cryptanalysis from round three, regardless of block size.



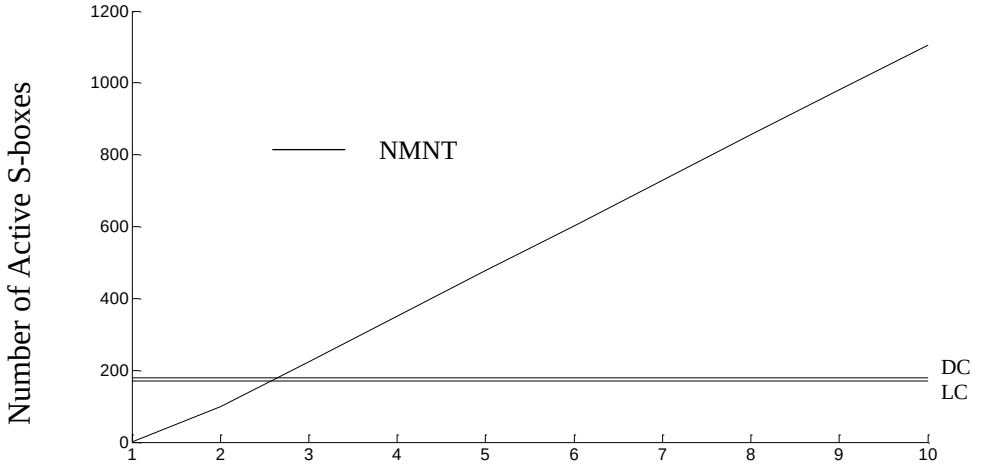
a. Block size: 16 elements



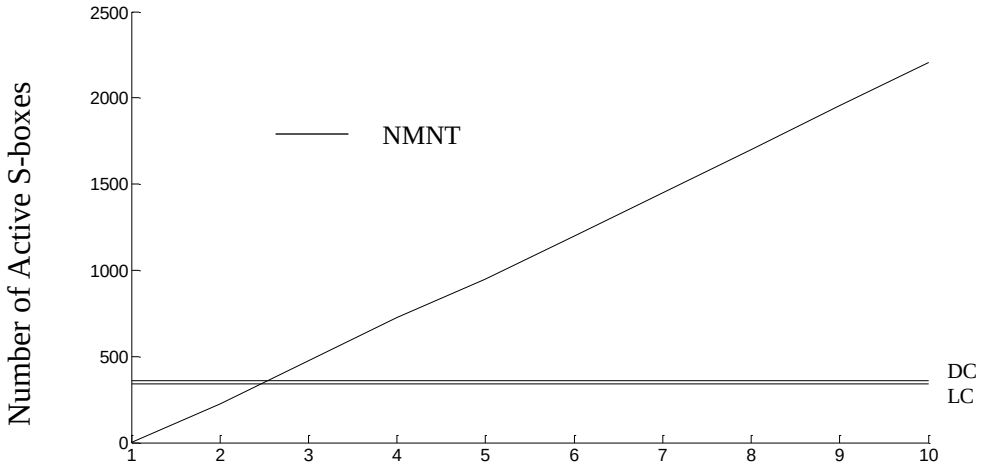
b. Block size: 32 elements



c. Block size: 64 elements



d. Block size: 128 elements



e. Block size: 256 elements

Figure 6.1: Minimum number of active S-boxes for different rounds and block sizes

6.7 Related-key Attacks

A different type of attack had been developed [118] which is based on the structure of the key schedule algorithm. This attack exploits the relationship between the keys to attack the cipher, and it demonstrates the importance of the key schedule algorithm to the overall security of the cipher so that care in this regard should be taken in the design. The attack was successful in attacking the LOKI [34] and Lucifer [119] algorithms. However it does not work with the DES [2] algorithm since the shift pattern used in the key schedule algorithm differs from round to round. In the AES [3] algorithm, round numbers are added in the process of the generation of the key schedule algorithm, which eliminate symmetries so as to prevent potential related-key attacks.

In the proposed cipher, the key schedule algorithm is designed to be strong enough to survive against this type of attack. It is designed with high diffusion and non-linearity, and in addition each round key generated depends on all previously generated keys as well as the cipher key and the round number. Together these measures exclude the possibility of weak-key and related-key attacks.

6.8 Slide Attacks

The slide attack [120, 121] is a powerful attack applicable to iterative block ciphers with a high degree of self-similarity, where the attack is independent of the number of rounds. In order to build a cipher resilient to such an attack, it is important to eliminate symmetry in the key schedule or in the round transformation in the main algorithm.

The proposed cipher is designed to be immune to such an attack by adding a variable addition layer in the round functions. The value of the variable is a function of the row number, the round number and the block number, and the value changes accordingly from round to round as well as from block to block. In addition, a round constant is added in the key schedule algorithm.

6.9 Brute-Force Attacks

The complexity of exhaustive key search attacks depends on the length of the cipher key, which is the key length to the power of 2 or on average half of this amount. Table 6.8 summarises the general levels of complexity associated with the exhaustive key search for different block sizes and variable word length.

Table 6.8: Exhaustive key search complexity

Block Size	$16\times P$	$32\times P$	$64\times P$	$128\times P$	$256\times P$
Complexity	$2^{16\times P}$	$2^{32\times P}$	$2^{64\times P}$	$2^{128\times P}$	$2^{256\times P}$
Average	$2^{(16\times P)-1}$	$2^{(32\times P)-1}$	$2^{(64\times P)-1}$	$2^{(128\times P)-1}$	$2^{(256\times P)-1}$

6.10 Weak Keys

Weak keys are those which produce output with detectable weaknesses, such as could occur if the non-linear parts of the cipher are key-dependent, which is not the case with the proposed cipher. A good example of weak keys are those found in the IDEA [122].

6.11 Conclusions

The security of the proposed algorithm against both differential and linear cryptanalysis has been considered, and security has been evaluated against the upper bounds of the probabilities of maximum differential characteristics and maximum linear characteristics, based on the DPP_{max}, IOC_{max}, and the minimum numbers of active S-boxes. The simulation results confirm that the cipher is secure against such attacks from round three regardless of the block sizes and word lengths

Chapter 7

Architecture Design and FPGA Implementation

In this chapter efficient hardware architectures for the proposed cipher are designed and implemented using Xilinx FPGA technology. The chapter is organised as follows. Section 7.1 gives a brief introduction to the topic. Section 7.2 lists the specifications of the target FPGA device. The design of the system is described in detail in section 7.3 and its correctness is verified in section 7.4. The implementation of the system is carried out in section 7.5 and device configuration is outlined in the following section. The results are discussed in section 7.7, and the complexity of the design is mentioned in section 7.8. Finally, the conclusions of the chapter are drawn in section 7.9.

7.1 Introduction

A hardware description language (HDL), such as VHDL or Verlog is usually used in the design flow to program FPGA boards. However, in order to reduce the complexity and design time, Xilinx System Generator is used, which allows a rapid development of the algorithm using the Matlab and Simulink data flow environment. System Generator is a powerful development tool that allows the designer to model a design entirely in a graphical environment. It consists of a Simulink library called the Xilinx Blockset which has direct mapping to HDL. The steps in the design are illustrated below.

7.2 Device Specifications

The target device is Virtex-6 XC6VLX130T-2FF484, which is chosen based on the amount of resources the design required. This device is implemented in 40nm CMOS process technology with a core voltage of 1.0V. It is volatile, since the device is based on SRAM technology, and therefore device configuration file must be reloaded whenever the FPGA is powered up.

The internal structure of this device can be summarised as follows. Each configurable logic block (CLB) consists of two slices, and each slice contains 4 LUTs, 8 flip-flops, a multiplexer and arithmetic carry logic. The LUT can be configured as either 6-input LUT (64-bit ROMs) with one output, or as two 5-input LUTs (32-bit ROMs) with 2 outputs. The LUTs of up to half of the available slices also can be used as a distributed 64-bit RAM or as a 32-bit shift register (SRL32) or dual SRL16s. The device consists of the following resources:

- 10,000 CLB (20,000 Slices and 1740 Kb Distributed RAM).
- 480 DSP48E1 Slices, each containing a 25×18 bit 2's complement multiplier, adder and a 48-bit accumulator, which can operate at 600 MHz.
- 9504 Kb Block RAM Blocks; each Block RAM is either 36Kb or 18Kb.
- 5 CMTs (Clock Management Tiles).
- 4 Ethernet MACs (Media Access Control).
- 20 GTX Transceivers (ultra-fast serial data transmission that combines transmitter and receiver and can operate at a data rate up to 6.6 Gbits/sec).
- 15 I/O Banks (40 pins per bank).

7.3 System Design

The proposed algorithms are efficiently designed using Xilinx FPGA System Generator which is a visual programming environment. This is a system-level design tool that allows system developer to utilise graphical data entry.

7.3.1 Design of the Components

The description of all components and layers involved in the design of the encryption and decryption modules as well as in the generation of the round keys for block size of 16×7 bits are illustrated below.

- **Padding and Preparing the Plaintext**

In this layer the plaintext is read from memory and converted from serial to parallel form with a predetermined block size of 12 bytes, where the plaintext should enter the cipher as a sequence of full blocks. The last block is padded with a byte of the value of 1 as an indication of the end of the message, and then it is padded with a number of bytes with a value of 0 if necessary to fill the block. This layer is illustrated in Figure 7.1.

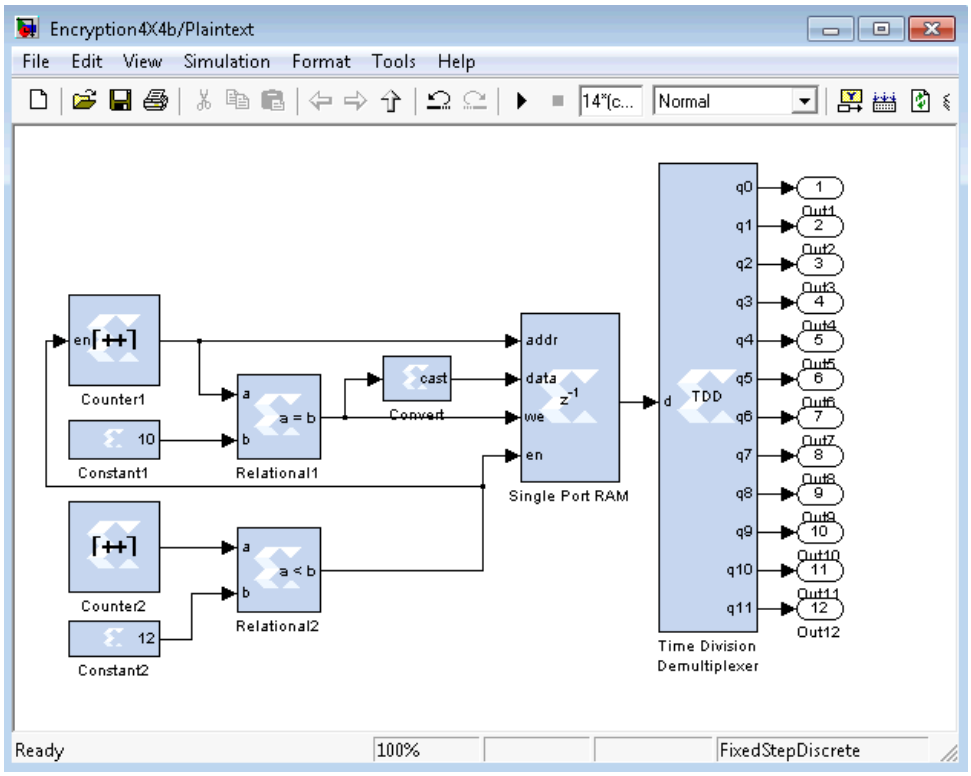


Figure 7.1: Preparing the plaintext

First of all, sufficient locations are reserved for the plaintext and the padded bytes in the Xilinx Single Port RAM shown in Figure 7.1. This is achieved as shown in Figure 7.2 by setting the depth of the memory according to the following formula: $12 \times ceil\left(\frac{numel(Plaintext)+2}{12}\right)$. These two extra bytes are added to the length of the message, so as to take into account the insertion of a byte which indicates the end of the message and another which is imposed as a result of that insertion. The actual values of the plaintext are assigned to the memory through the initial value vector.

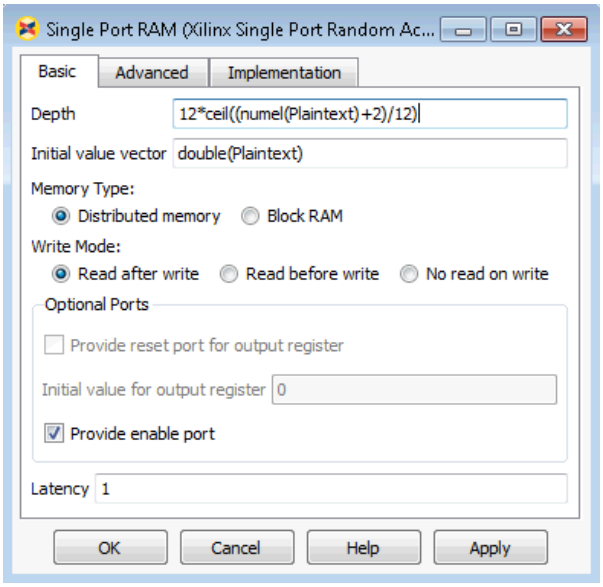


Figure 7.2: Memory setting

The Xilinx Counter block, which is titled ‘Counter1’ in Figure 7.1, is used to allocate locations in the memory for read/write data. It counts from 0 up to the total number of bytes in the plaintext including the padded bytes. The setting of the counter is shown in Figure 7.3, for instance the parameter ‘Count to value’ is set according to the same formula used in setting the depth of the memory minus 1, as the counter counts from 0.

The Xilinx Relational block ‘Relational1’ is used as a comparator. It compares the counter’s value of ‘Counter1’ with the constant ‘Constant1’ whose value is set to the total number of bytes of the plaintext without considering the padded bytes. If there is no matching, the data is continuously read from the memory according to the addresses pointed to by the counter. If they match, that is, at the end of the message, a byte of the value of 1 is written in the memory at that location. This can be achieved by passing the value of 1 to the memory read/write control signal to set write mode and to the data input port to write this value after converting the data type from Boolean to an unsigned integer number 8-bit in length through the casting block.

The Xilinx Relational block ‘Relational2’ compares the counter’s value of ‘Counter2’ that counts from 0 to 13 with the constant ‘Constant2’ whose value is set to 12 based on the block size which is 12 bytes. Both memory and first counter ‘Counter1’ remain enabled if the value of Counter2 is less than the value in Constant2. This step is used to make the number of cycles required for reading and converting each input block from serial to parallel 14 instead of 12 cycles.

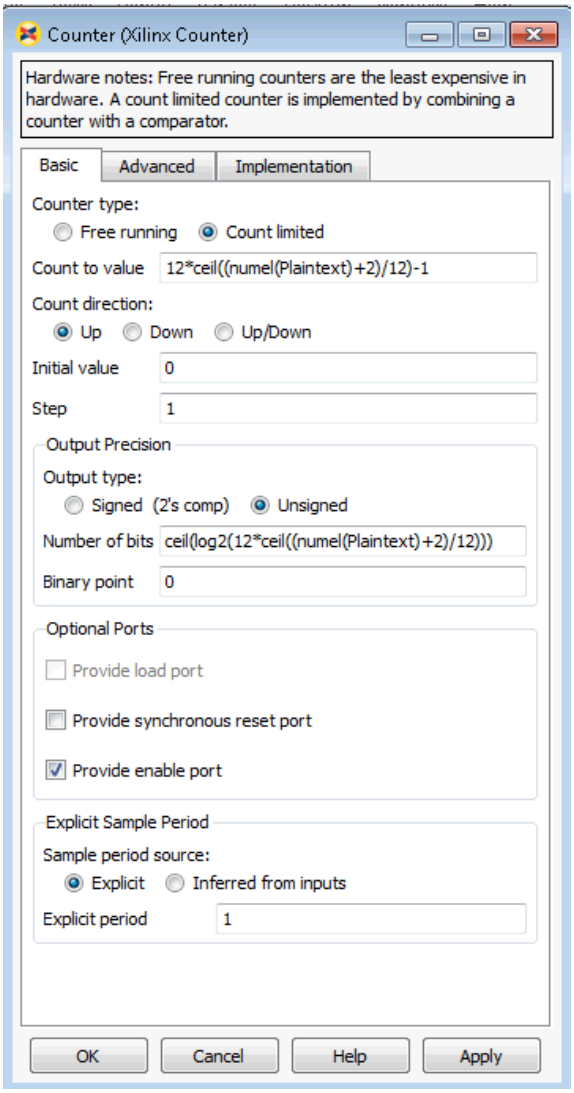


Figure 7.3: Counter1 setting

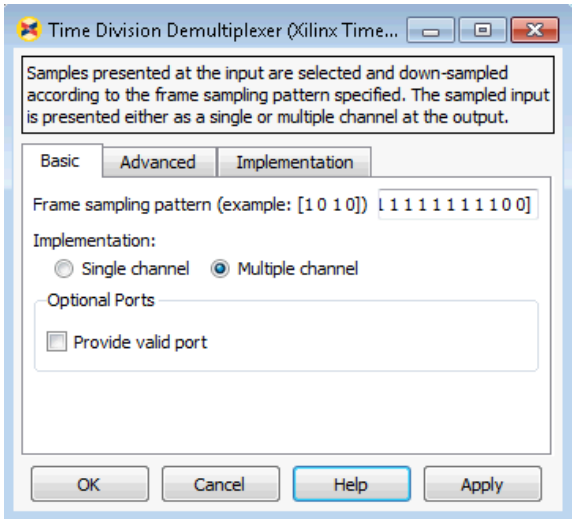


Figure 7.4: Time division demultiplexer setting

The bytes of each block are then down-sampled by using the Xilinx Time Division Demultiplexer block, which accepts input serially and presents it to parallel outputs. The last two bits from the 14-bit in the frame sampling pattern are set to 0, as shown in Figure 7.4, therefore only the first 12 input values from each input data frame (block) are down sampled at a rate of 14. This step, along with that related to Counter2, are essential to make the required time for reading each input block 14 instead of 12 cycles, so as to be compatible with the time required for reading one block of key or ciphertext data.

- **Conversion of Elements**

In this layer the bytes of the block are converted from 8 to P-1 bits (6-bit). This is achieved by converting the bytes into a binary form, concatenating them all and then slicing them into the required length, as shown in Figure 7.5. The Xilinx Concat block

performs a concatenation of the 12 input bytes of the unsigned integer numbers. The slicing is then achieved using the Xilinx Slice block, which slices off a sequence of 6-bit. The result of this operation is a block of 16 elements of 6-bit each. The setting of the slice block is shown in Figure 7.6, where the parameter 'Offset of top bit' is set to 0 for the first sliced element, -6 for the second, -12 for the third, and so on.

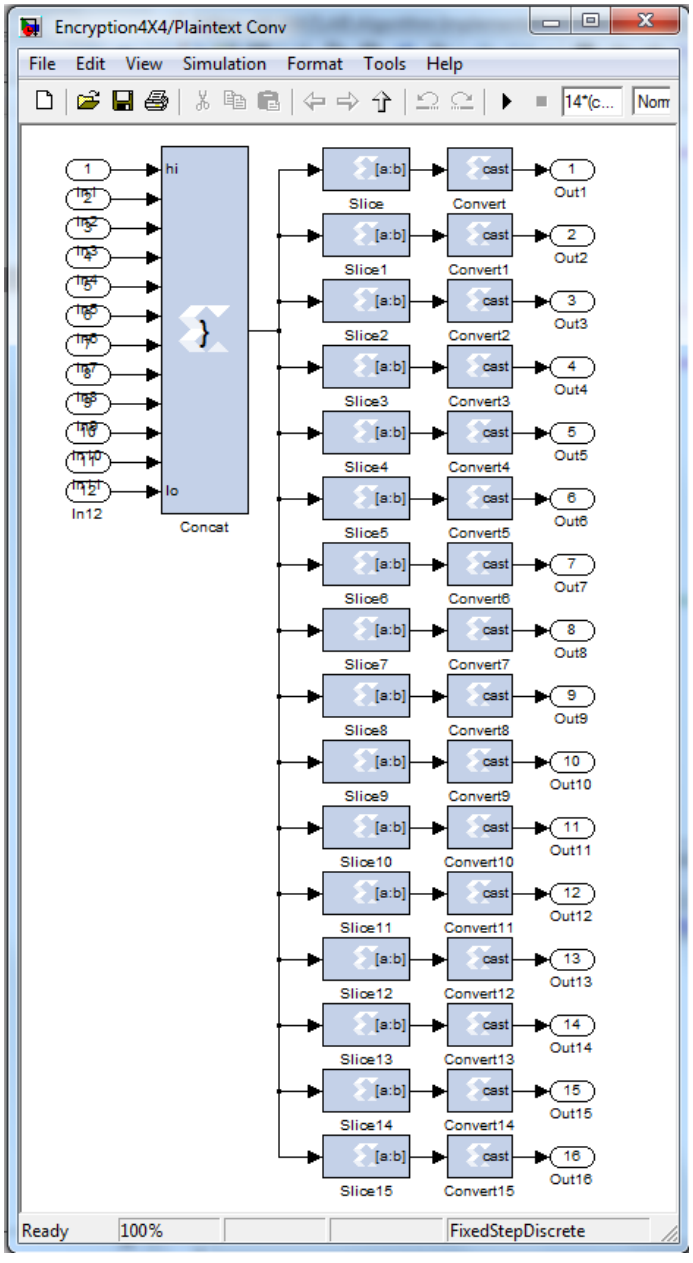


Figure 7.5: Conversion of elements

The last step in this layer is to convert the length of the elements from 6 to 7-bit by inserting a bit with the value of 0 at the most significant bits (MSB) from each element. This step can be achieved using the Xilinx Convert block (cast), which is set according to Figure 7.7.

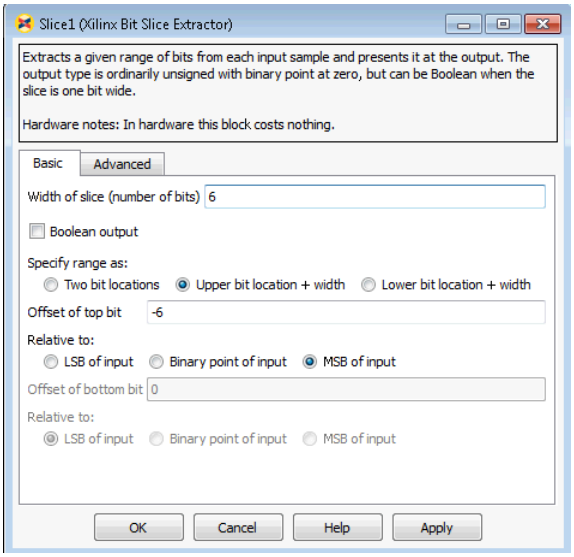


Figure 7.6: Slice setting

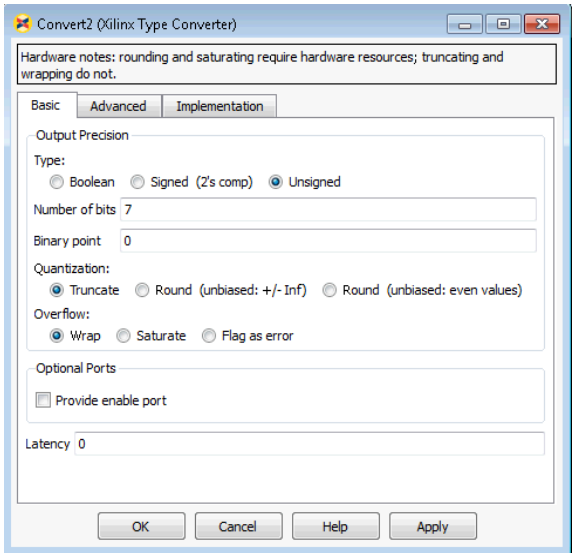


Figure 7.7: Convert setting

- **Elements Addition**

All arithmetic operations performed in the cipher are modular operations. Since in NMNT the modulus M_p is a power of 2 minus 1, modular addition is achieved by adding the carry bit after the addition, as shown in Figure 7.8 [123, 124].

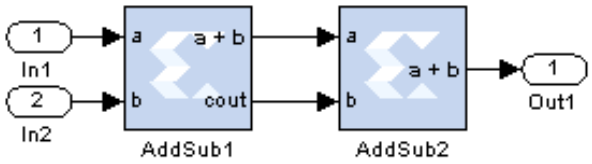


Figure 7.8: Modular addition

For instance, consider the case for P = 7. The modulus is then M_p = 2^7-1 = 127, and let a = 100 and b = 30. Accordingly: $\langle a + b \rangle_{M_P} = \langle 100 + 30 \rangle_{127} = \langle 130 \rangle_{127} = 3$

In hardware this operation is performed as follows:

100 in binary form is: 1100100

30 in binary form is: 0011110 +

Their sum : 1 0000010

By adding the carry : 1 +

0000011 = 3 in decimal form

For FNT the modulus is a power of 2 plus 1, since the modular addition is achieved by adding the elements and then subtracting the carrier instead [81, 125].

- **Elements Subtraction**

In cases where the cipher is based on the NMNT, the modular subtraction is implemented in exactly the same way as when implementing modular addition, first negating the subtrahend simply by complementing it, as shown in Figure 7.9.

For instance, for M_p = 127, a = 100 and b = 30, $\langle a - b \rangle_{M_P} = \langle 100 - 30 \rangle_{127} = 70$

In hardware this operation is performed as follows:

30 in binary form is: 0011110

100 in binary form is: 1100100

-30 : 1100001 +

1 1000101

By adding the carry : 1 +

1000110 = 70 in decimal form

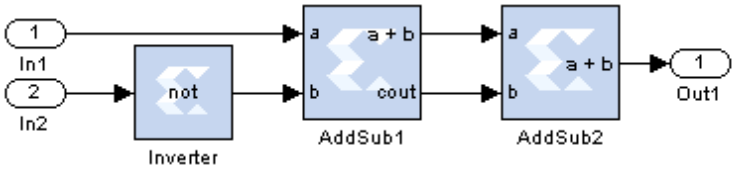


Figure 7.9: Modular subtraction

For FNT negating is achieved by first complementing the positive integer and then adding 2.

- **Element Substitution**

In this layer the elements are non-linearly mapped into other elements through the S-box. The S-box is represented as a look-up table and stored in a read-only memory (ROM) using the Xilinx ROM block. The ROM is initialised by storing the values of the elements of the S-box in the parameter 'Initial value vector', as shown in Figure 7.10. For an inverse S-box a different look-up table is used, and accordingly the parameter 'Initial value vector' is replaced by the relevant values.

- **Shifts Transform**

The elements in the first three columns of the array are cyclically shifted over different numbers of offset in this layer. In hardware this layer as well as the inverse shifts transform layer is simply hardwired without involving logics.

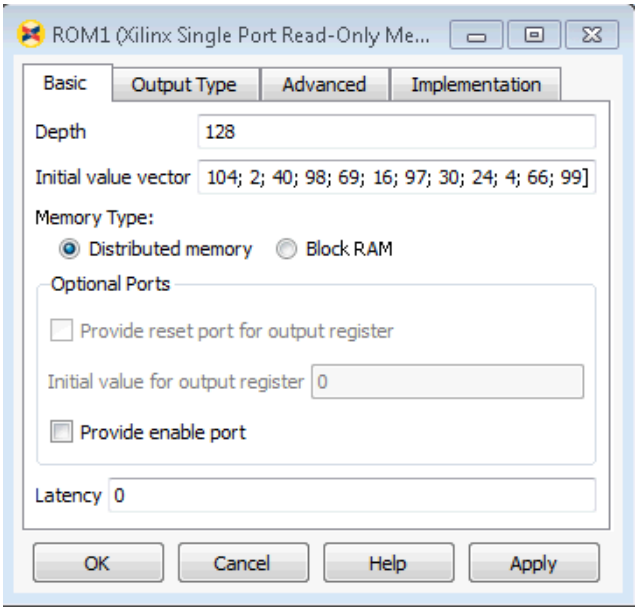


Figure 7.10: ROM setting for S-box

- **Variable Addition**

In the variable addition layer, the variable values are added to the elements at location *N*/4+1 from each row in the array and then the results are multiplied by 2. The value of the variable is a function of a row number, round number and a block number. The values of the row and round numbers are pre-computed and added to the intermediate elements as a constant, then the block number is added to the results, and finally the output is multiplied by 2, as shown in Figure 7.11. The multiplication is achieved by using the Xilinx Shift block which shifts the input signal according to the required direction and offset. In this case it is left-shifted 1-bit and then the outputs of the two registers are added after slicing them, as shown in Figures 7.12. The setting of the properties of the Xilinx shift block is explained in Figure 7.13.

- **Variable Subtraction**

The steps of the variable addition are here processed in reverse order and inverse functions are used instead. Hence it starts by multiplying the input element by ½, and then the block number is subtracted from the result which is then subtracted from the constant as shown in Figure 7.14. Multiplying by ½ is achieved by left-shifting the elements by 6-bit (2^{-1+7}) and then adding the values of the two registers after slicing them. If the elements are right-shifted 1-bit instead, the outputs are only correct if the result of their multiplication is below the modulus value.

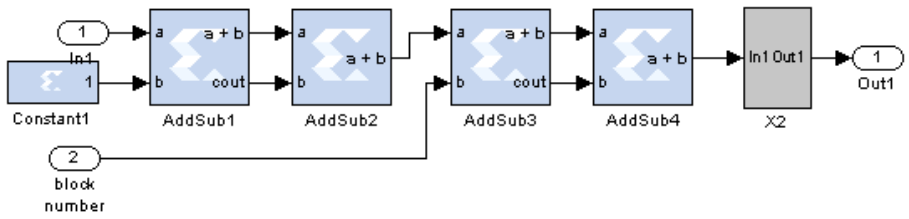


Figure 7.11: Variable addition

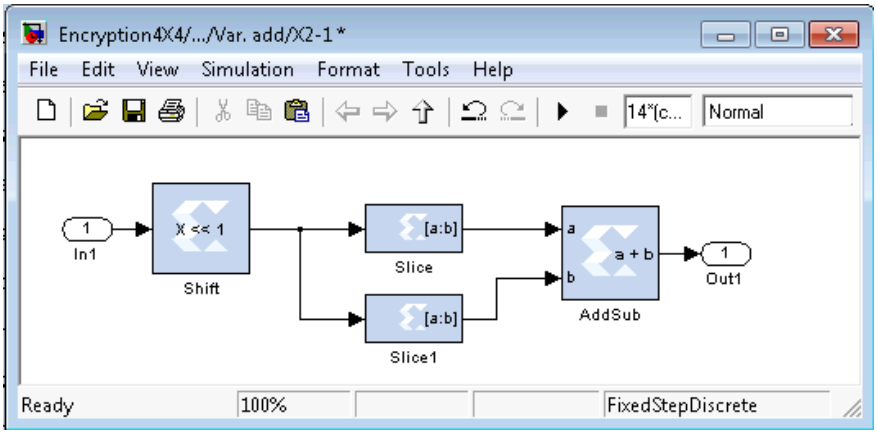


Figure 7.12: Multiplication by 2

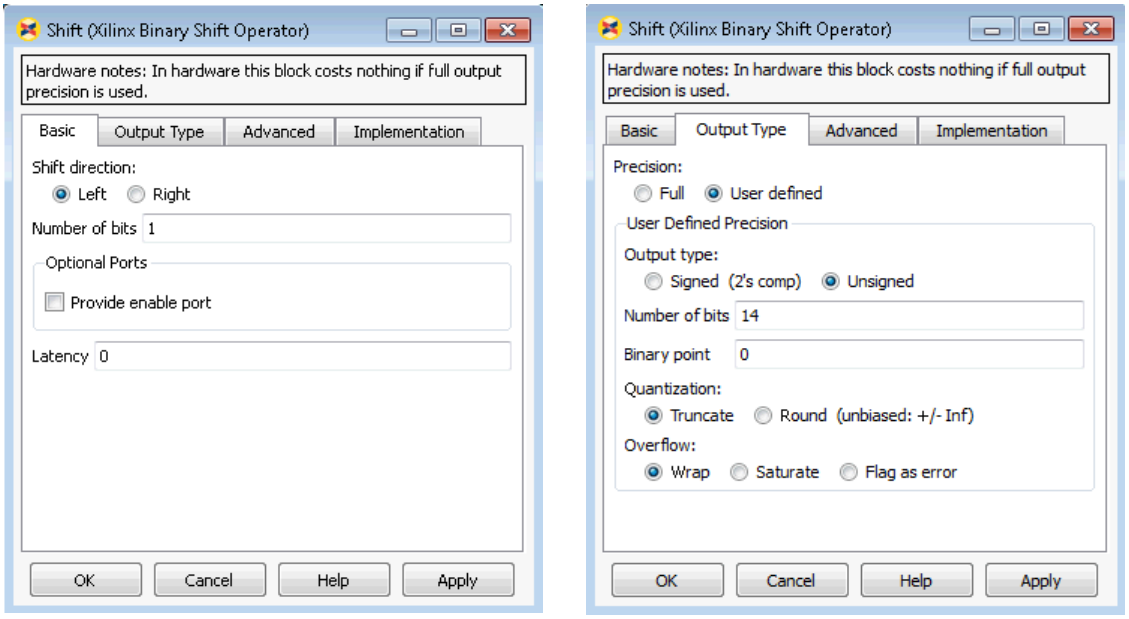


Figure 7.13: Properties setting of Xilinx shift's block

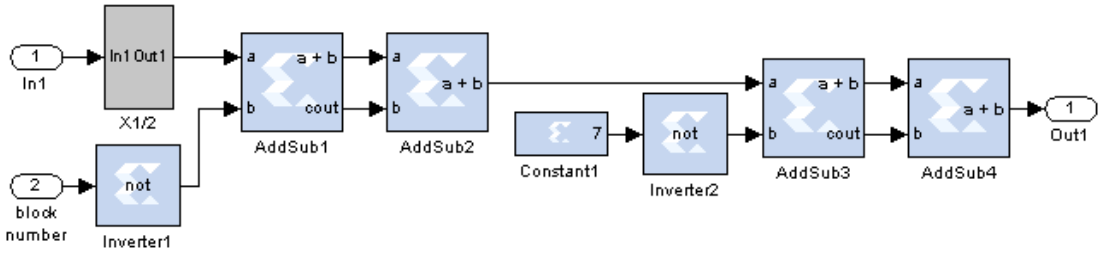


Figure 7.14: Variable Subtraction

- **New Mersenne Number Transform**

In the present work, the NMNT is implemented by applying the split-radix decimation in time (DIT) algorithm, which was initially introduced for the fast calculation of the DFT [126]. This algorithm is considered one of the most efficient algorithms in computing fast transforms. It applies the radix-2 decomposition to even-indexed samples and radix-4 decomposition to odd-indexed samples of $X(k)$.

Therefore, the calculation of the forward NMNT in equation 3.2 is modified according to equation 7.1 [127].

$$X(k) = \langle X_{ev}(k) + X_{od}(k) \rangle_{M_P} \tag{7.1}$$

where X_{ev} (k) and X_{od} (k) are the even and odd-indexed samples, respectively, given below:

$$X_{ev}(k) = \left\langle \sum_{n=0}^{N/2-1} x(2n)\beta(2nk) \right\rangle_{M_p} = X_{2n}(k) \tag{7.2}$$

$$X_{od}(k) = \left\langle \sum_{n=0}^{N/4-1} x(4n+1)\beta((4n+1)k) + \sum_{n=0}^{N/4-1} x(4n+3)\beta((4n+3)k) \right\rangle_{M_p} \tag{7.3}$$

The X_{od} (k) in equation 7.3 can be further decomposed as shown in equation 7.4 using the trigonometric identity represented in equation 7.5.

$$X_{od}(k) = \left\langle \begin{matrix} [X_{4n+1}(k)\beta_1(k) + X_{4n+1}(N/4-k)\beta_2(k)] + \\ [X_{4n+3}(k)\beta_1(3k) + X_{4n+3}(N/4-k)\beta_2(3k)] \end{matrix} \right\rangle_{M_p} \tag{7.4}$$

$$\beta(m+n) = \langle \beta_1(n)\beta(m) + \beta_2(n)\beta(-m) \rangle_{M_P} \tag{7.5}$$

Accordingly, $X(k)$ can be written as:

$$X(k) = \left\langle \begin{matrix} X_{2n}(k) + [X_{4n+1}(k)\beta_1(k) + X_{4n+1}(N/4-k)\beta_2(k)] + \\ [X_{4n+3}(k)\beta_1(3k) + X_{4n+3}(N/4-k)\beta_2(3k)] \end{matrix} \right\rangle_{M_p} \tag{7.6}$$

where $X_{2n}(\cdot)$ is $N/2$-point 1-D NMNT and $X_{4n+1}(\cdot)$ and $X_{4n+3}(\cdot)$ are N/4-point 1-D NMNTs.

The other transformations $X(N/4+k)$, $X(N/2+k)$ and $X(3N/4+k)$ were also derived [127]:

$$X(N/4+k)=\left\langle \begin{array}{l} X_{2n}(N/4+k)- \\ [X_{4n+1}(k)\beta_2(k)-X_{4n+1}(N/4-k)\beta_1(k)]+ \\ [X_{4n+3}(k)\beta_2(3k)-X_{4n+3}(N/4-k)\beta_1(3k)] \end{array} \right\rangle_{M_p} \quad 7.7$$

$$X(N/2+k)=\left\langle \begin{array}{l} X_{2n}(k)-[X_{4n+1}(k)\beta_1(k)+X_{4n+1}(N/4-k)\beta_2(k)]- \\ [X_{4n+3}(k)\beta_1(3k)+X_{4n+3}(N/4-k)\beta_2(3k)] \end{array} \right\rangle_{M_p} \quad 7.8$$

$$X(3N/4+k)=\left\langle \begin{array}{l} X_{2n}(N/4+k)+ \\ [X_{4n+1}(k)\beta_2(k)-X_{4n+1}(N/4-k)\beta_1(k)]- \\ [X_{4n+3}(k)\beta_2(3k)-X_{4n+3}(N/4-k)\beta_1(3k)] \end{array} \right\rangle_{M_p} \quad 7.9$$

The in-place butterfly of the 1-D NMNT for split-radix DIT algorithm is shown in Figure 7.15 [127].

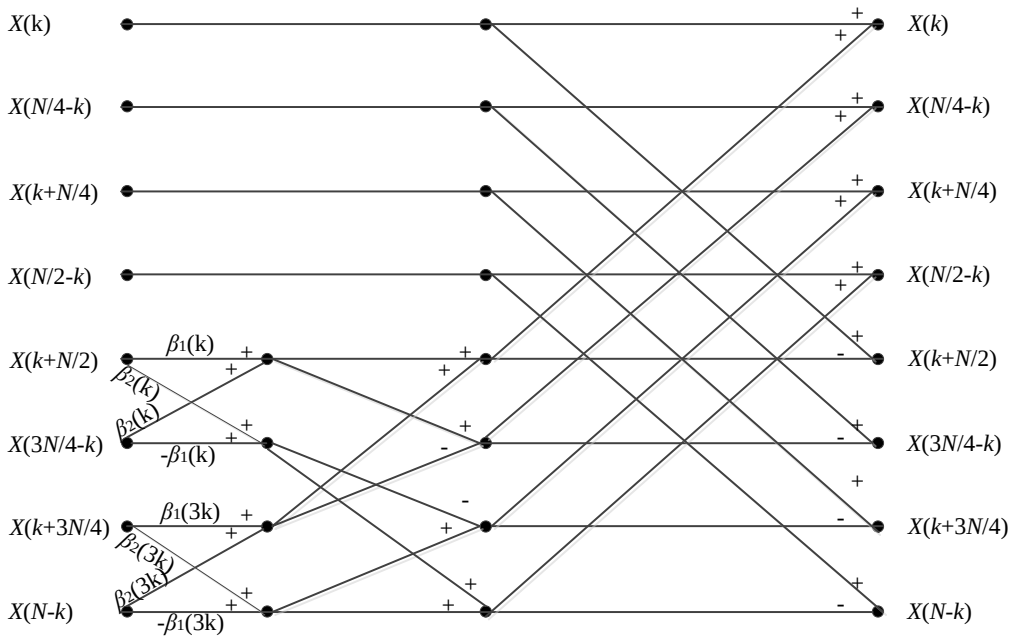


Figure 7.15: In-place butterfly of the split-radix DIT for the 1-D NMNT

The 4-points 1-D NMNT signal flow graph is shown in Figure 7.16 and the architecture design of the NMNT following this graph is presented in Figure 7.17.

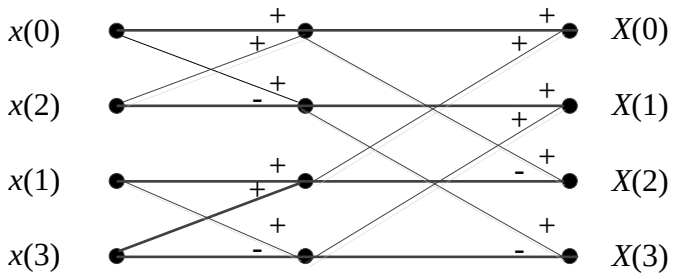


Figure 7.16: 4-points NMNT signal flow graph

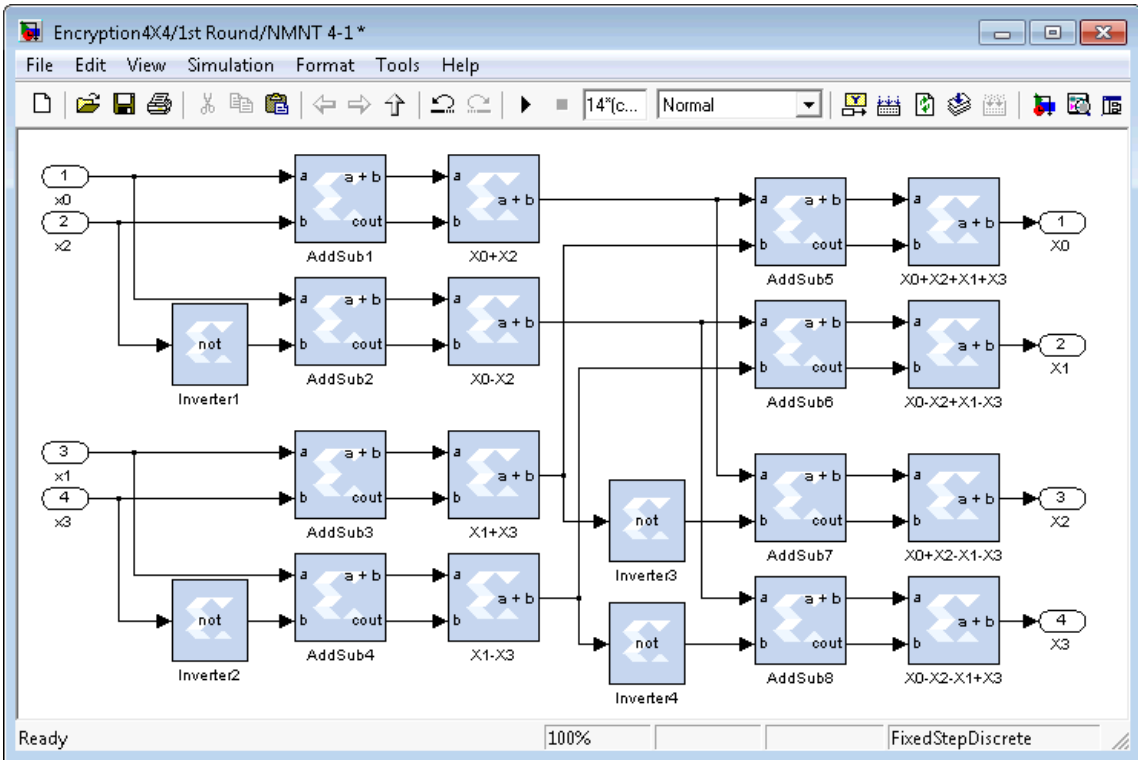
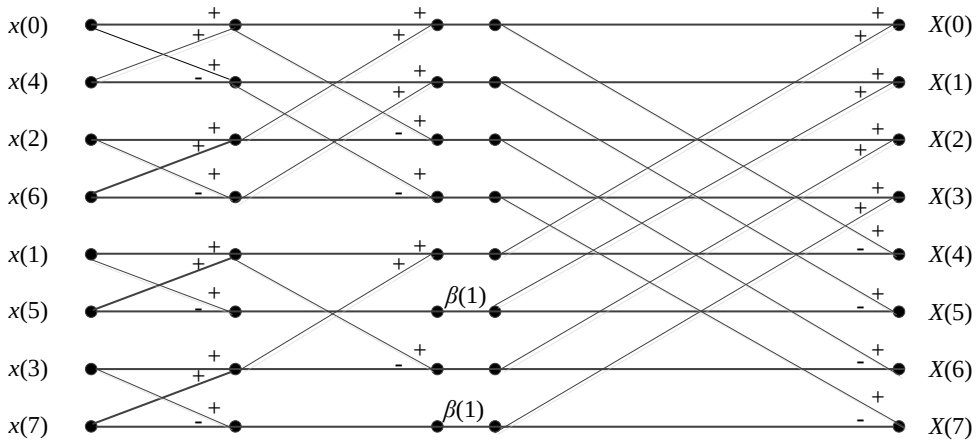


Figure 7.17: NMNT design for transform length (N) = 4

The signal flow graph of the 8-points 1-D NMNT is shown in Figure 7.18 and its architecture design is explained in Figure 7.19.

The top left quarter of the signal flow graph represented in Figure 7.18 is exactly similar to that in Figure 7.16, hence in the design shown in Figure 7.19 this part is replaced by a block titled NMNT 4 for transform length 4, which is based on Figure 7.17. The bottom left quarter of Figure 7.18 is designed and combined in a subsystem as shown in Figure 7.19, and the internal structure of this subsystem is displayed in Figure 7.20.



$\beta(1) = -2^{(P+1)/2} = -16$ for $(P = 7)$

Figure 7.18: 8-points NMNT using the split-radix DIT algorithm ($M_p = 127$)

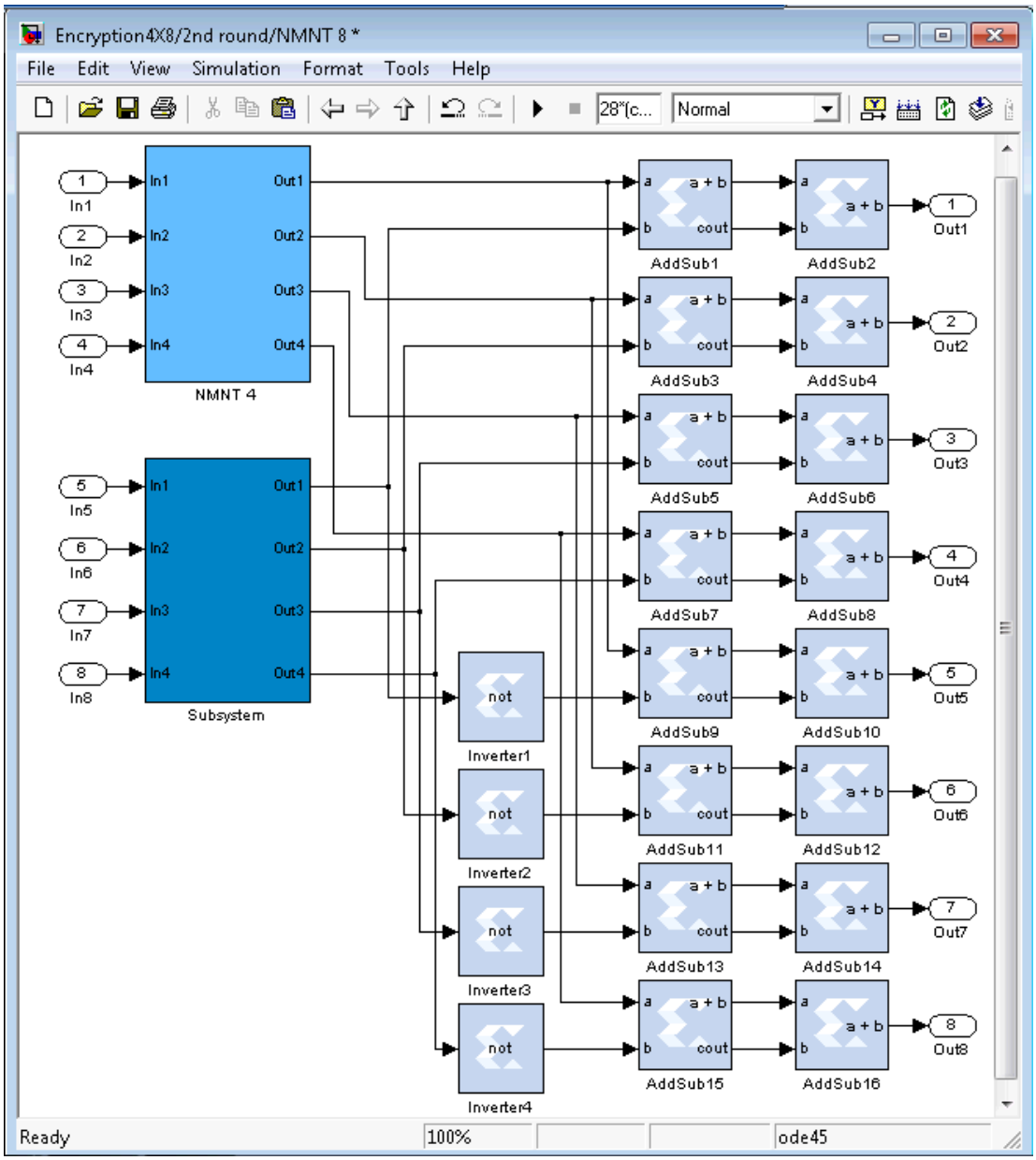


Figure 7.19: NMNT design for $N = 8$ using the split-radix DIT algorithm

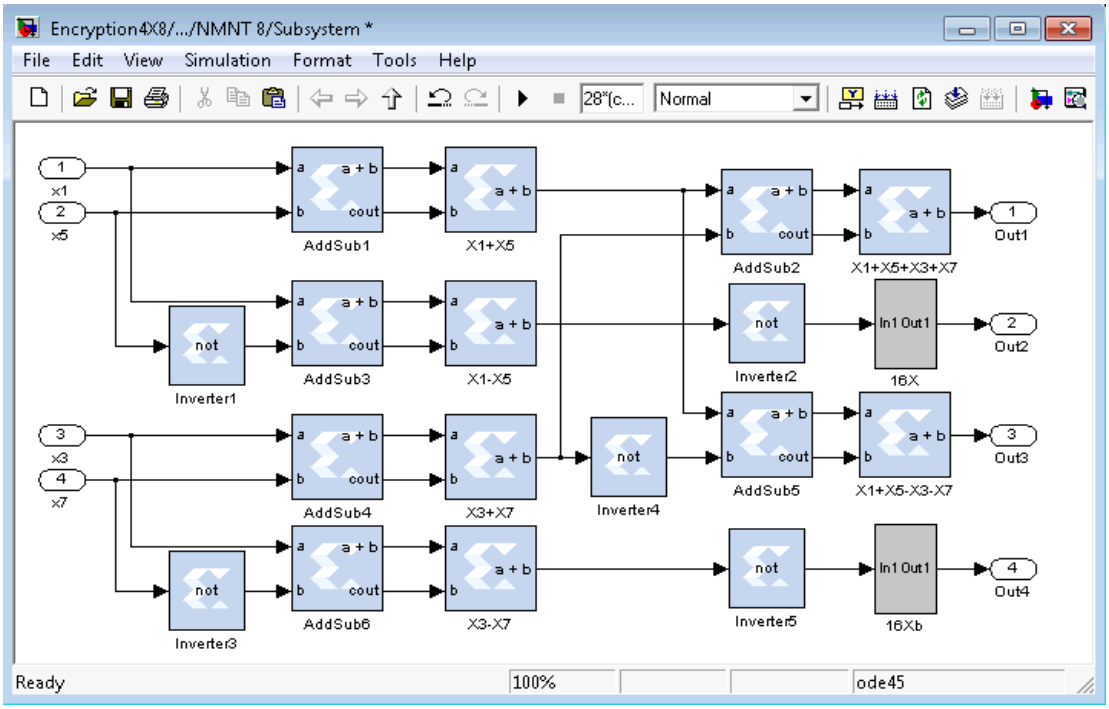
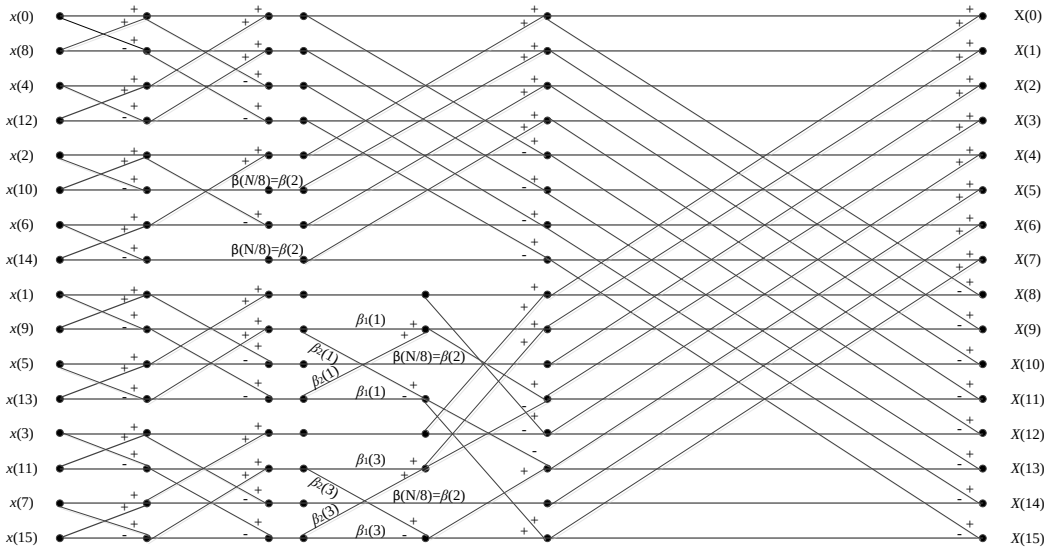


Figure 7.20: The internal structure of the Subsystem

In Figure 7.18 the signal $x(1)$-$x(5)$ as well as $x(3)$-$x(7)$ are multiplied by $\beta(1)$, which has a value of -16. This can be achieved in the design by first negating these signals using inverters as shown in Figure 7.20 (inverters 2 and 5), and then multiplying by 16. Because 16 is an integer of a power of 2, this multiplication can be replaced by a left shift operation with a 4-bit offset.

Finally, the 16-points 1-D NMNT signal flow graph is shown in Figure 7.21 and its architecture design is presented in Figure 7.22.



$\beta_1(1) = 106,\ \beta_2(1) = -24,\ \beta_1(2) = \beta_2(2) = -8,\ \beta(2) = \beta_1(2)+\beta_2(2) = -16,\ \beta_1(3) = -24,\ \beta_2(3) = -21$

Figure 7.21: 16-points NMNT using the split-radix DIT algorithm ($M_p = 127$)

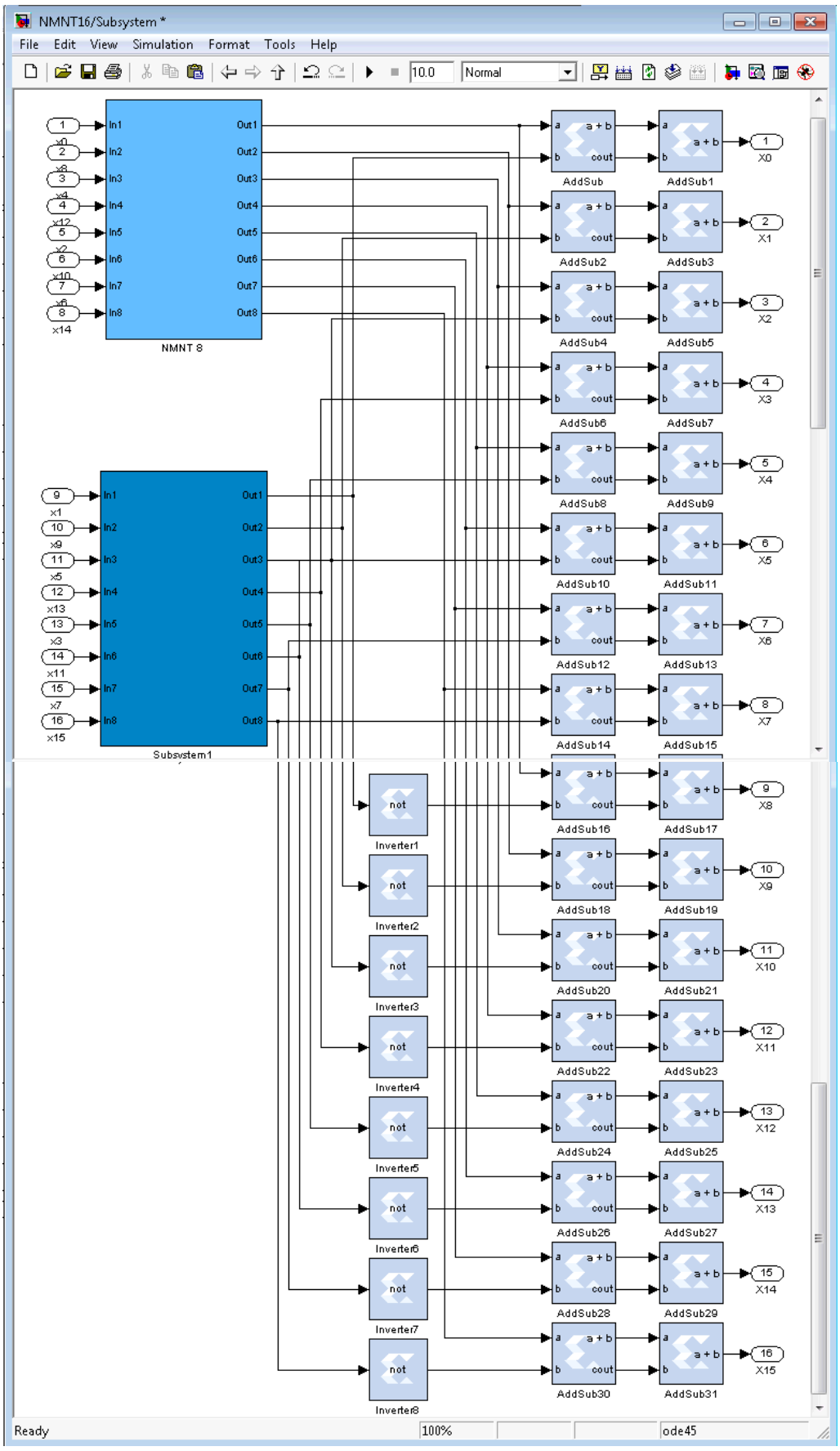


Figure 7.22: NMNT design for $N = 16$ using the split-radix DIT algorithm

The top left quarter of the signal flow graph is an NMNT of length 8, and accordingly a block of the NMNT with this length is placed in the design as shown in Figure 7.22.

The bottom left quarter of Figure 7.21 which is represented as subsystem1 on the design in Figure 7.22 is further described in detail in Figure 7.23.

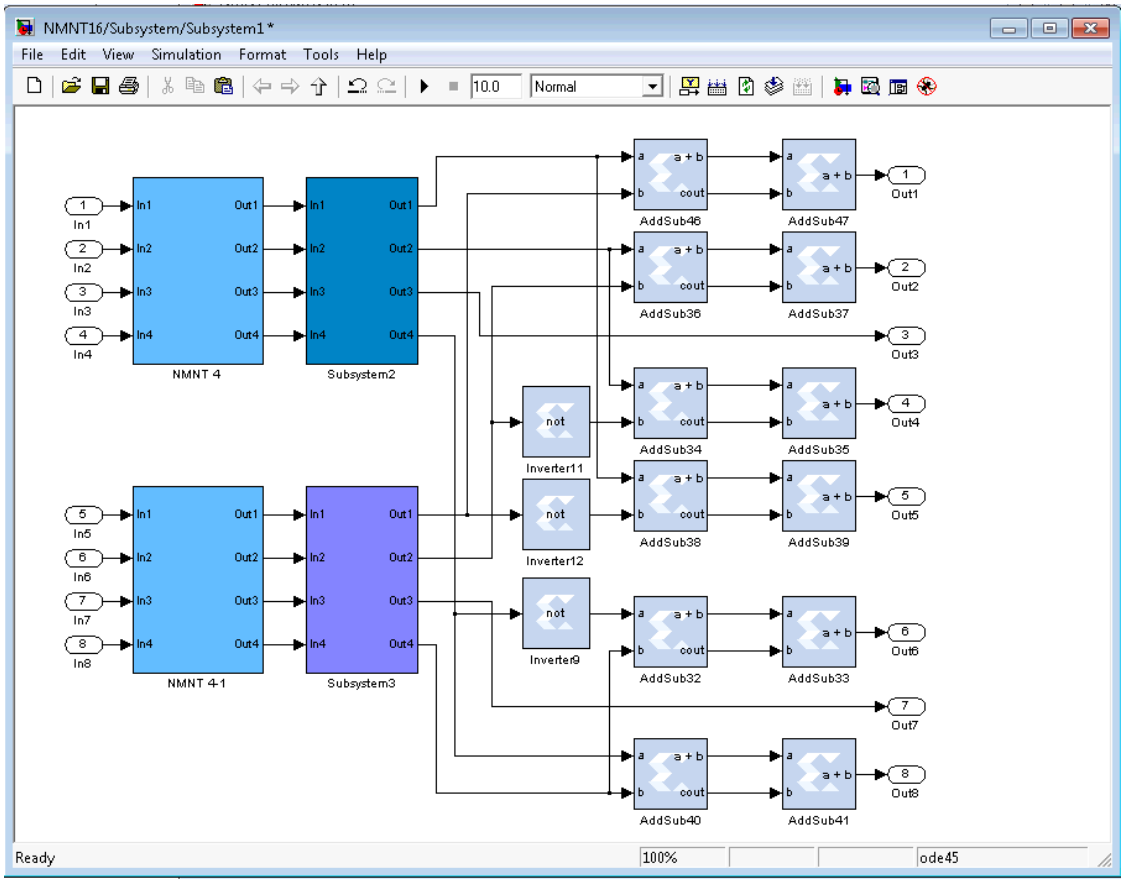


Figure 7.23: The internal structure of the Subsystem1

The internal structure of Subsystem2 is explained in Figure 7.24 while Subsystem3 has the same structure, swapping the external multiplication blocks with the internal, i.e. 24 and 21.

Multiplying by 24 or 21 can be achieved either by using the Xilinx multiplier block as shown in Figure 7.25 to multiply the two elements and then the values of the two registers are combined after slicing them, or by using a sequence of shift and addition operations. The former option requires 3 latencies, as shown in the parameter setting in Figure 7.26. The second option costs 0 latency, and can be achieved by decomposing the constants into a number of integers of the power of 2, and therefore the integer 24 can be decomposed into $16+8 = 2^4+2^3$ and integer 21 into $16+4+1 = 2^4+2^2+2^0$.

Accordingly, multiplying by 24 can be achieved as shown in Figure 7.27 by left-shifting the element 4-bit and then the original element is left-shifted 3-bit and the result is achieved by combining the shifted elements. The same applies for multiplying by 21, which is achieved by left-shifting the element 4-bit and then the original element is left-shifted this time 2-bit, the shifted elements are combined and the value of the original element is added to the combination to achieve the result as illustrated in Figure 7.28.

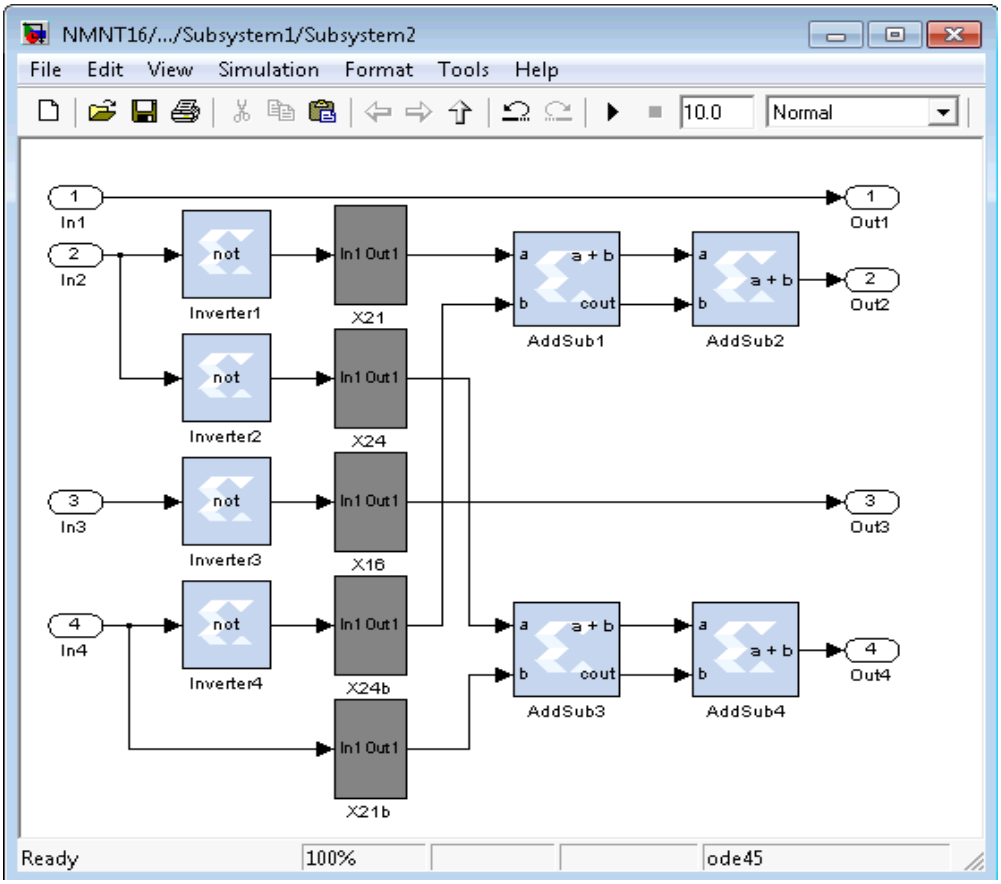


Figure 7.24: The internal structure of the Subsystem2

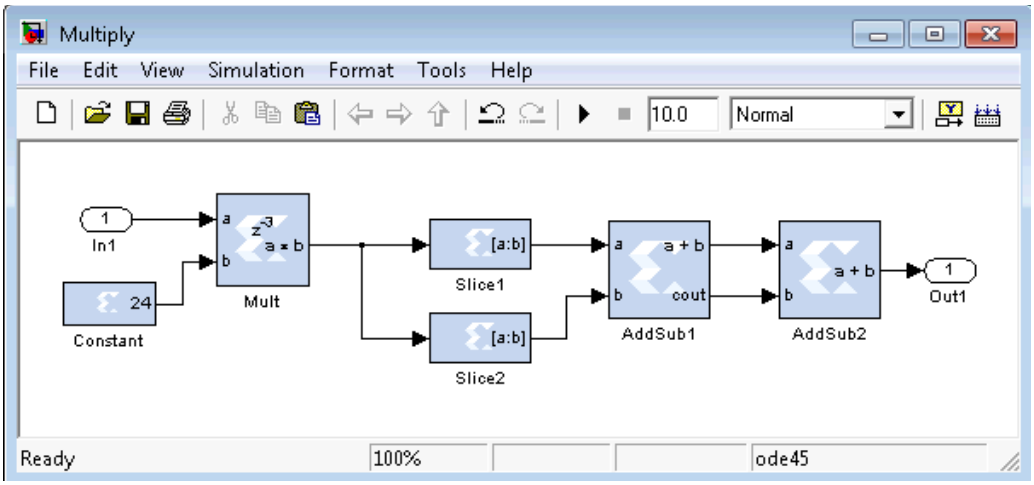


Figure 7.25: Modular multiplication

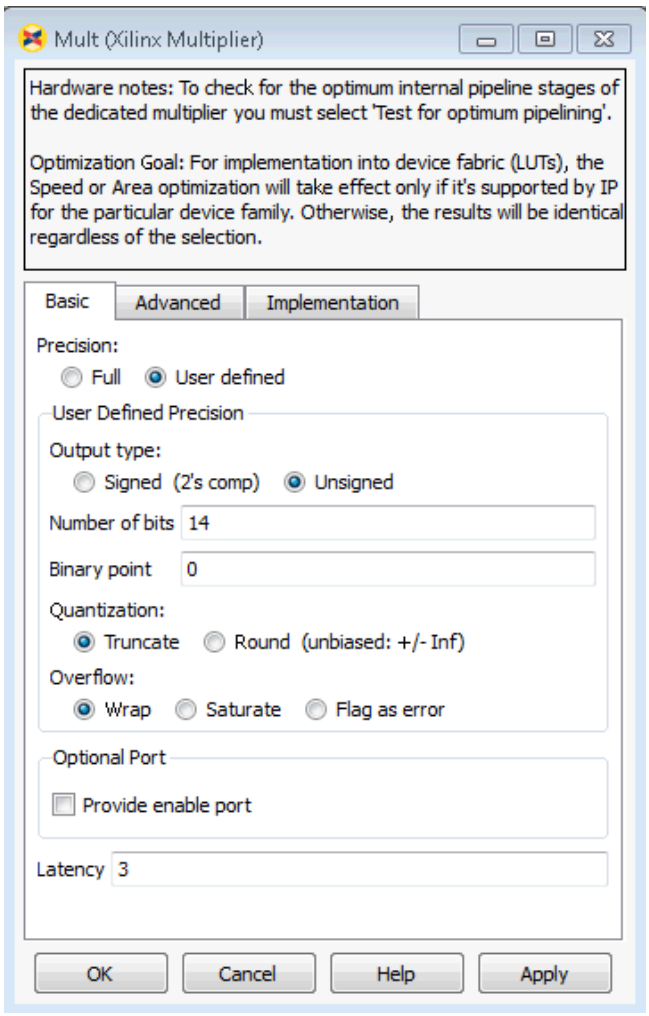


Figure 7.26: Xilinx multiplier block setting

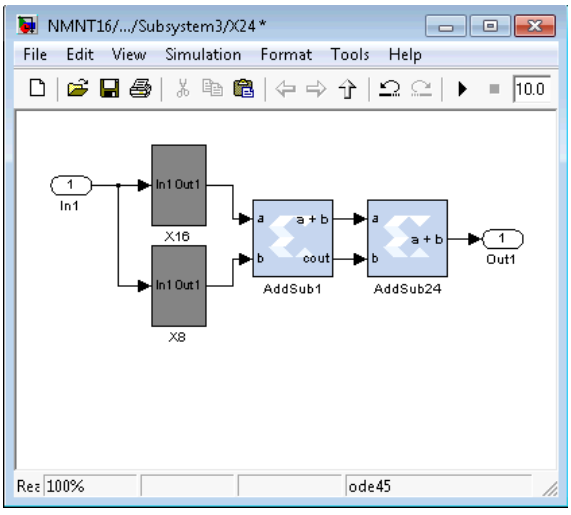


Figure 7.27: Design for multiplying by 24

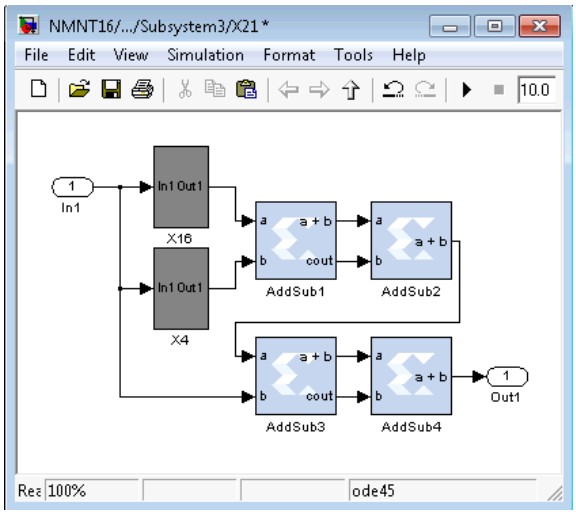


Figure 7.28: Design for multiplying by 21

- **Round Transformation**

The round transformation, as mentioned in Chapter 5, consists of five layers: element substitution, shifts transform, variable addition, NMNT, and finally, round key addition. The design of one round transformation as shown in Figure 7.29 can be achieved by combining the layers previously described in this section, where the elements of each layer are processed in parallel. Each S-box block composes of four look-up tables, the shifts transform is hardwired, and the variable addition block is explained in Figure 7.30. Finally the NMNT blocks are based on the design of Figure 7.17, and the elements of the intermediate results are added to the round key elements through the round key block which is explained in Figure 7.8.

The design of the round transformation for the decryption circuit is given in Figure 7.31, which follows the same steps in designing the encryption circuit in reverse order and inverse functions are used instead. It starts with a round key subtraction which is the same as addition by negating the subtrahend using an inverter. Then the processing of the inverse NMNT, this is achieved by implementing the forward NMNT followed by a multiplication with a $1/N$, which is a 5-bit left-shifting (2^{-2+7}). This is followed by the variable subtraction layer as described in Figure 7.32 and inverse shifts transform which is hardwired. Finally, the inverse S-box which maps elements using the look-up table.

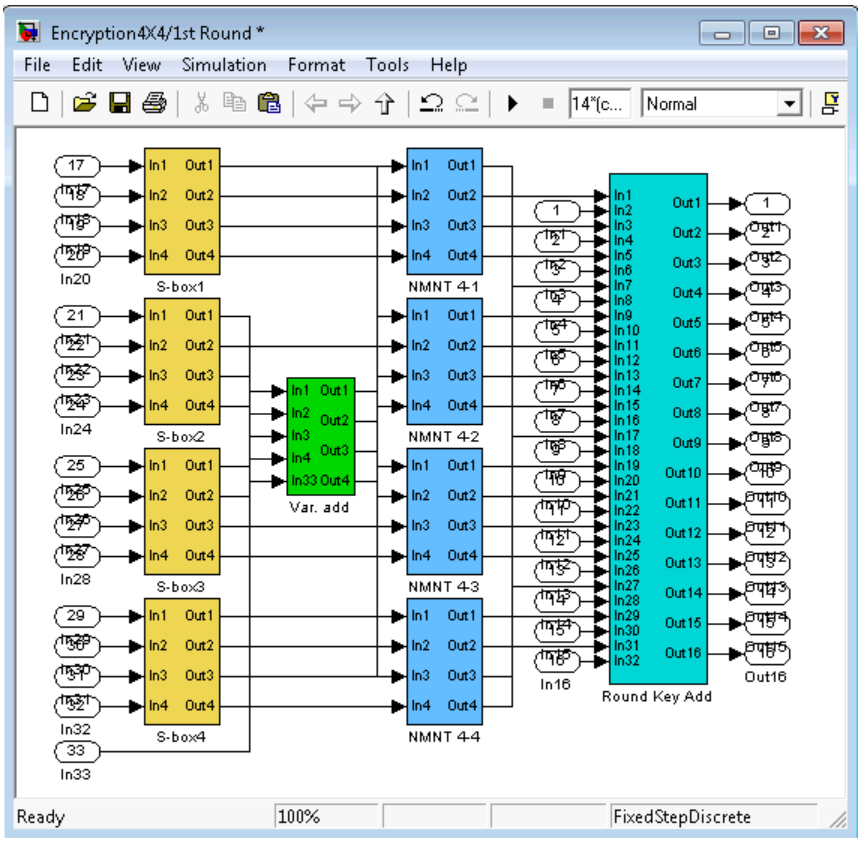


Figure 7.29: Design of forward round transformation

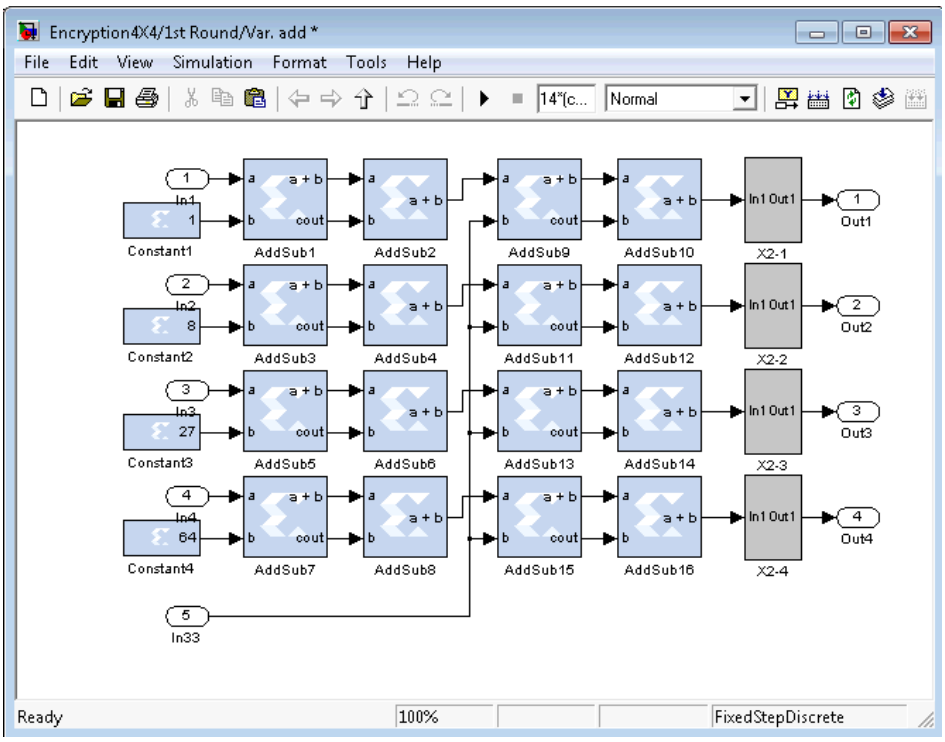


Figure 7.30: Design of variable addition layer

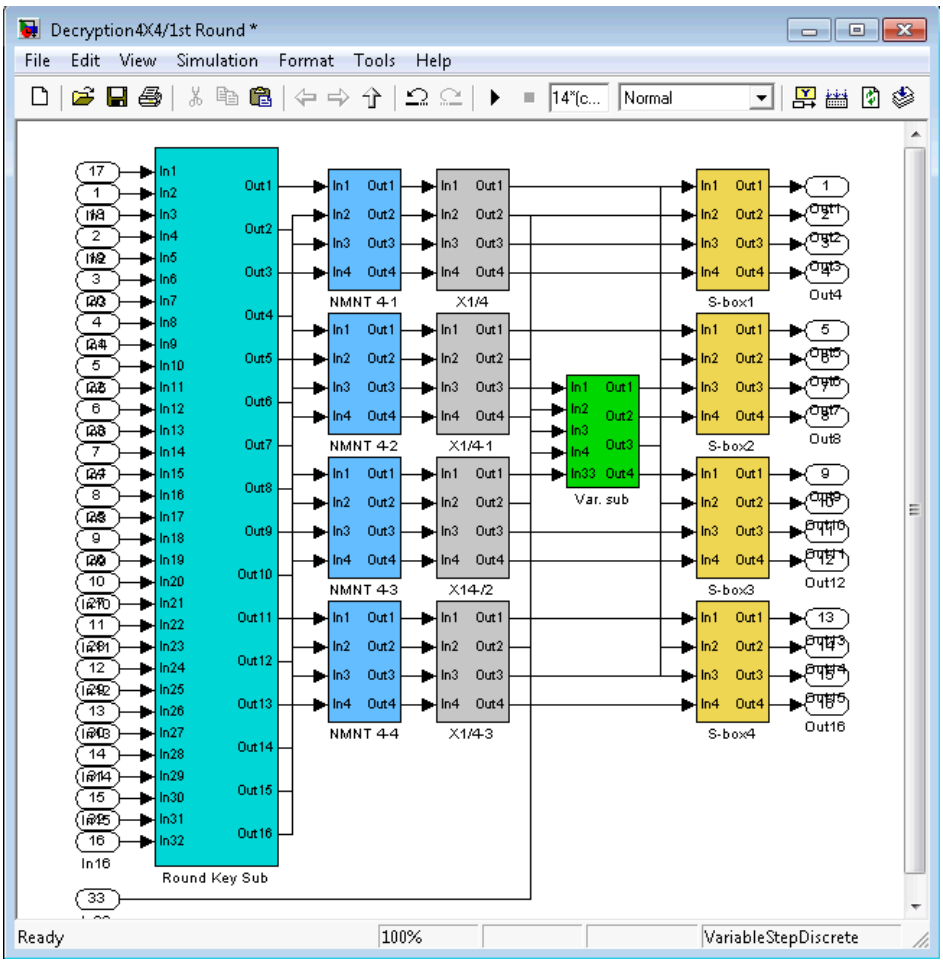


Figure 7.31: Design of reverse round transformation (decryption)

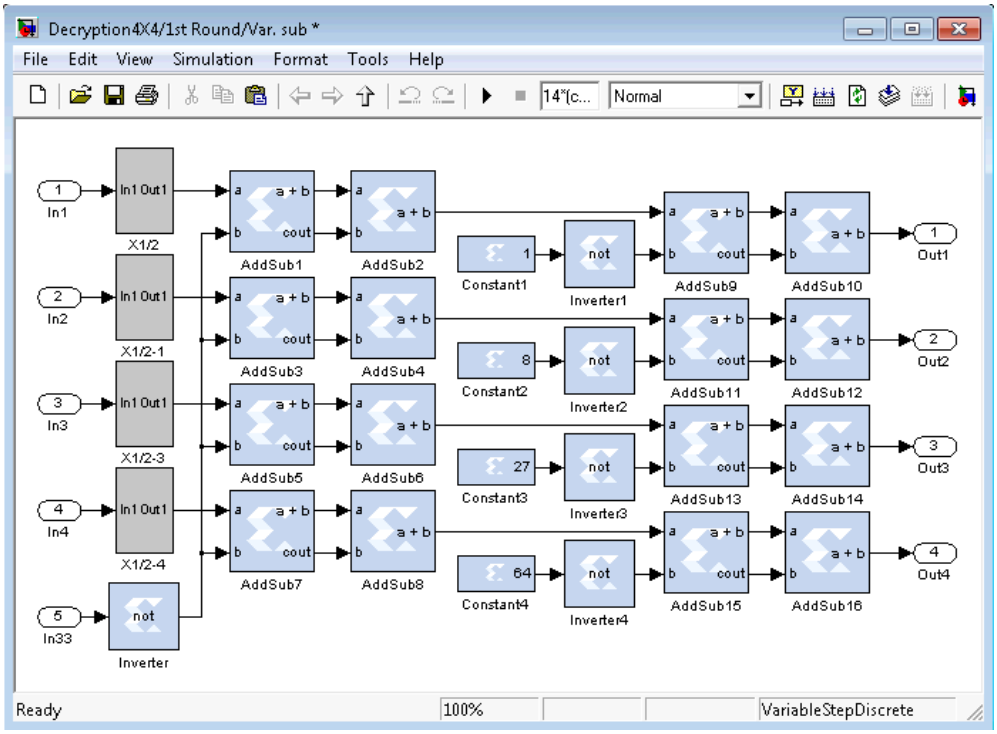


Figure 7.32: Design of variable subtraction layer

7.3.2 Design Based on Loop Unrolling Architecture for Block Size of 16×7-bit

In the design the focus was on throughput rather than on area, and therefore this architecture has been chosen which duplicates the hardware required to implement each round. The complete encryption circuit of block size 16×7-bit accompanied by the key generation is shown in Figure 7.33. The blocks in the bottom row (blue) is the encryption part, where the first block is used to pad and prepare the plaintext as illustrated in Figure 7.1. The plaintext is read from memory and converted from serial to parallel form according to the specified block size. In the following block the elements are converted from 8 to 6-bit, and then to 7-bit by inserting a bit with the value of 0 at the MSB of each element, making plaintext blocks ready for processing. This is explained in Figure 7.5. The third block in Figure 7.33 is the initial key addition layer, which simply adds the cipher key elements to the elements of the plaintext. The fourth block up to block thirteen are the rounds transformations presented in Figure 7.29. These blocks are identical apart from the values of the variable addition layer and the round keys. The inputs to these blocks as shown in Figure 7.33 are in addition to the elements from the previous round and the round keys a value from a counter which represents the value of the block number that is needed in the calculation of the variable value in the variable addition layer. As this counter counts for the block number, its

value should be incremented every 14 cycles, and accordingly the parameter explicit period should be set to 14 in the counter setting window.

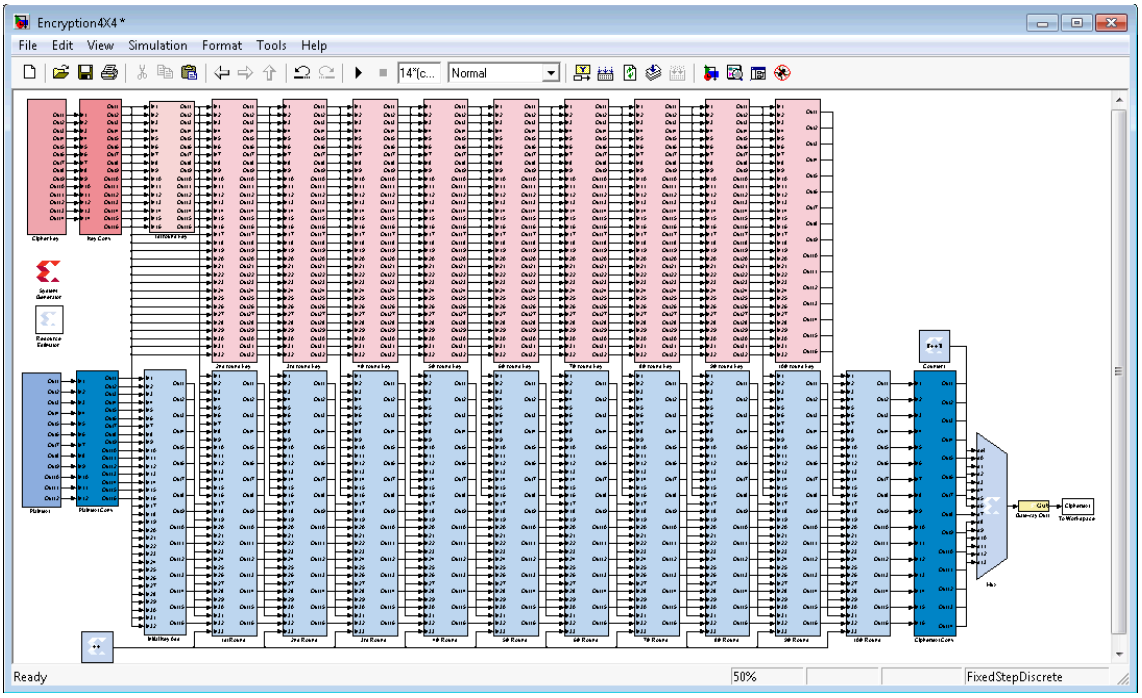


Figure 7.33: Algorithm design for encryption based on loop unrolling architecture

After processing all rounds, the elements of the block are converted from the transform domain length to bytes. This is achieved by concatenating all block elements and slicing them off into 8-bit each, which is shown in Figure 7.33 by the dark blue block. In the following final stage, the elements are converted from parallel into serial form using the Xilinx bus multiplexer block.

The round keys are derived from the cipher key by means of the key schedule algorithm which is processed on-the-fly in parallel with the encryption algorithm. This is achieved by processing the blocks in the top row (pink). First the secret key or password supplied by the user, depending on its length, is either padded or truncated if necessary to the required key length which is 14-byte. This step is performed in software by implementing the codes shown in Algorithm 7.1, which is run in the hardware model by specifying this function in the callback pane of the model initialisation properties. Next, the 14-byte generated for the cipher key are read from the memory and converted to parallel form and then to a transform length which is 7-bit for each element. These steps are performed in the first two blocks in the diagram. After that, ten rounds of transformation are applied, and at each round one round key is generated, with the design of the round transformation shown in Figure 7.34.

```
function SecretKey = SecretKeyGen

% This function is used to input secret key and pad or truncate its
% length according to the required length

%Input Secret key
Key = double(input('Enter Secret Key up to 14 strings : ', 's'));

while numel(Key) < 2
    Key = double(input('Enter Secret Key up to 14 strings : ', 's'));
end

nc = numel(Key);

if nc > 14
    SecretKey = Key(1:14);
else
    %pad password to the required length rNxNxP/8
    for i= nc+1:14
        Key(i)=mod(Key(i-1)+Key(i-nc),256);
    end
    SecretKey = Key;
end
```

Algorithm 7.1 Secret Key Generation

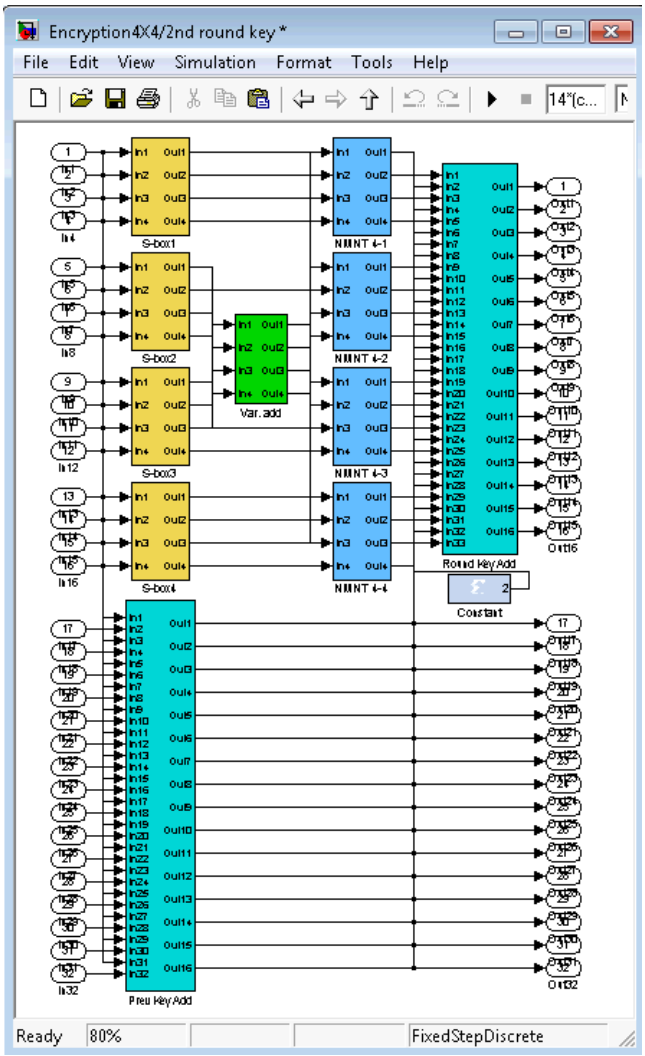


Figure 7.34: Design of round transformation for round keys schedule algorithm

The round transformation used in the design of the key schedule algorithm is similar to that used in the design of the encryption algorithm. The block number, which is one of the parameters used in computing the value of the variable in the variable addition layer, is eliminated. Then, rather than adding a single round key at each round in the round key addition layer, an accumulation of all previously generated round keys are added in addition to the cipher key and the round number, which is achieved in the design by adding a 'Prev Key Add' block. In this block, at each round, the value of the last generated key is added to the values of all previously generated keys.

The design of the decryption algorithm is shown in Figure 7.35. It has the same structure as that in the encryption algorithm but is processed in reverse order as well as the order of the rounds key. In addition inverse transformations are used instead. The ciphertext is read from memory and every 14-byte are converted from serial to parallel, which is achieved in the first block. In the second block the element length is changed from bytes to 7-bit. Then ten rounds are processed, and each round block is based on the round transformation illustrated in Figure 7.31. After that, the initial key is subtracted from the processed elements. Next, the elements are converted from 7 to 6-bit so as to remove the 0 bit added at the beginning of each element, and then the elements are converted to bytes. Finally, the elements are converted from parallel to serial form representing the resulting plaintext.

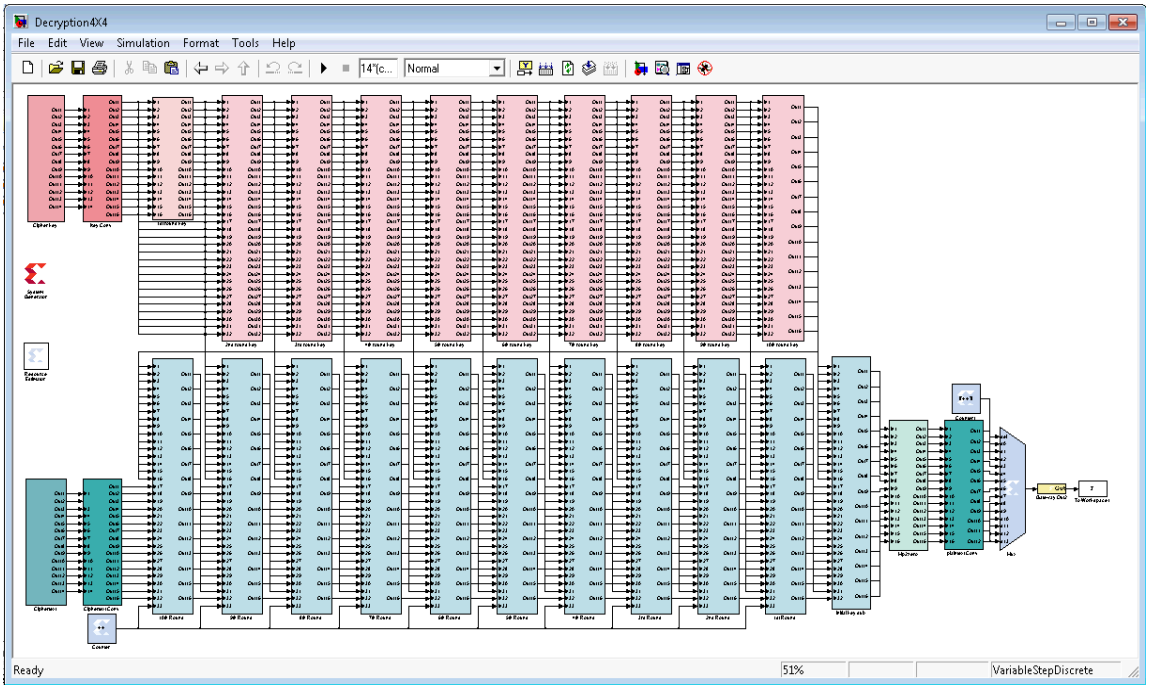


Figure 7.35: Algorithm design for decryption based on loop unrolling architecture

The Xilinx Resource Estimator block shown in Figure 7.35 is used to provide an estimation of the resources necessary to implement a model in hardware. The other block, the System Generator (token) is used to ensure control over the system during simulation and to handle the specified code generation and simulation.

7.3.3 Design Based on Loop Unrolling Architecture for Block Size of 32×7-bit

The design of the round transformation circuits for both encryption and decryption are shown in Figures 7.36 and 7.37, respectively. The differences between these circuits and those previously designed for a block size of 16×7-bit shown in Figures 7.29 and 7.31 are that the former involves an NMNT of length 8 instead of 4, and uses nearly double the resources.

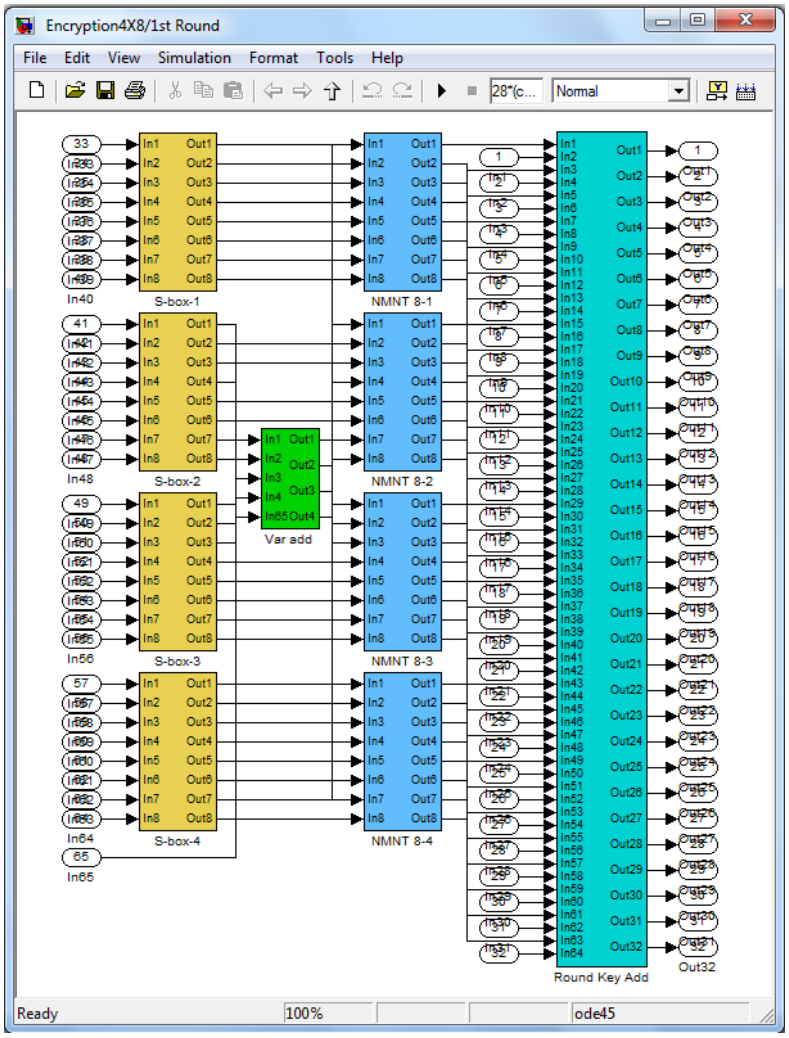


Figure 7.36: Design of round transformation for block size 32×7

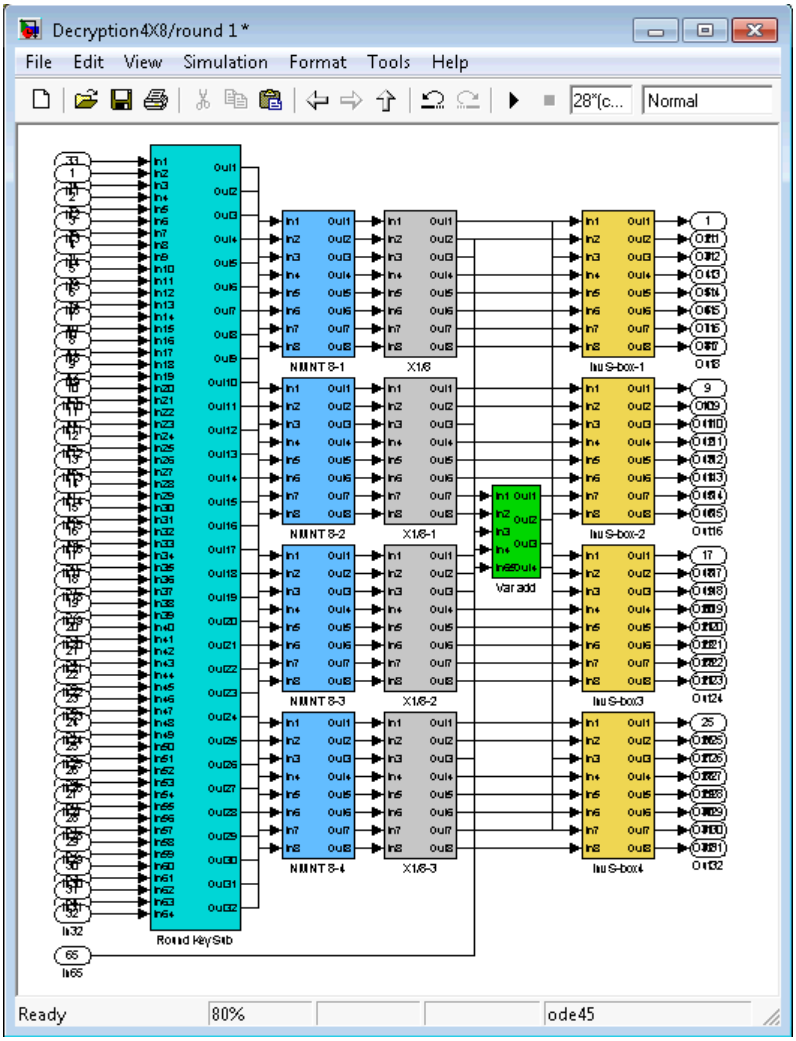


Figure 7.37: Design of inverse round transformation for block size 32×7

7.4 Verification and Simulation

This step is carried out to verify the correctness of the architecture according to the specifications in the design. It operates by simulating the model in the Simulik environment. If the results are unsatisfactory or the simulation is interrupted, for instance due to unassigned block connections, a design revision is necessary. Otherwise the design is ready for implementation. To make the process of checking easy to follow and to correct any errors pointed out by the verification, individual completed functions have to be verified first, then a combination of functions, until the whole system is verified.

7.5 Implementation

The following step after verifying the performance of the design is the implementation, which translates the final design into an HDL code (VHDL or Verilog) or into a form

that can be used later to configure the device. This is achieved by clicking ‘Generate’ in the Xilinx system generator block which automatically processes a sequence of operations starting from producing a bit-accurate and cycle-true synthesised netlist to translate, map, and place and route operations.

The parameter specific to the system generator block consists of compilation and clocking options, as shown in Figure 7.38. Compilation options allow the designer to choose the appropriate: compilation, part, target directory, synthesis tool, HDL, and the possibility to create a testbench if required. Compilation specifies the type of output, which could be Netlist files, bitstream, hardware co-simulation, or timing and power analysis. The appropriate FPGA device is selected through the part option, and the possible tools that could be used to synthesise the design are: Synplify Pro, Synplify, and Xilinx’s XST. Finally, in the HDL option the type of hardware description language to be used for the compilation of the design is specified, which could be VHDL or Verilog. In the clocking options, for instance, the designer can set the appropriate system clock period whether for a single or multi-rate design. In addition, the FPGA system clock period can be derived from the Xilinx digital clock manager (DCM) input block period which is an external hardware-defined input.

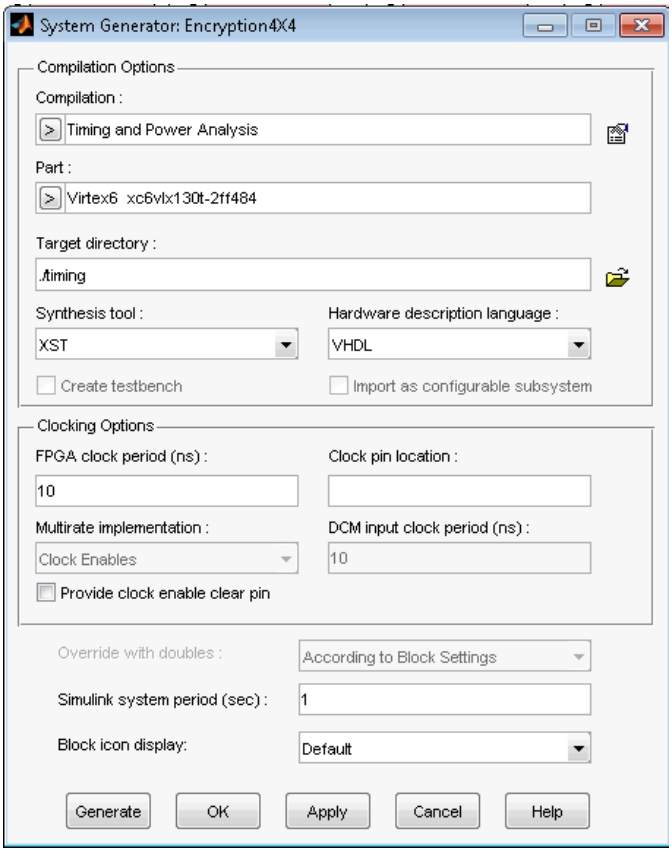


Figure 7.38: Parameter specific to the system generation block (token)

7.6 Configuration

After the FPGA design is completely placed and routed, the final step in the design flow is to configure the FPGA by generating a bitstream file and downloading it into the internal memory (PROM) of the target device using the iMPACT software which is part of the Xilinx tools installation. Once this file is downloaded, the FPGA is configured every time it is powered up. The procedures for configuring the FPGA device are illustrated below:

- Run the Generate Programming File from the ISE Project Navigator in the Processes pane (see Figure 7.39), which in turn will run the Xilinx bitstream generation program (BitGen) to prepare the design information and produces a bitstream file to configure the FPGA device. Before running the generation it is possible to set the properties of the process, such as for example, allow an encryption option or enable bitstream compression.
- Open ISE iMPACT software which is used to configure the target device as well as to generate configuration files for the Xilinx FPGA and PROMs. Through the Boundary Scan add the target device, or initialise JTAG chain to assign the target device from all detected devices on the chain, in this case the FPGA device has to be connected to the computer through a Xilinx download cable.
- Generate a PROM file (.mcs) which contains the PROM configuration information. This can be generated each time the Configure Target Device process is run by setting the Automatically Run Generate Target PROM / ACE

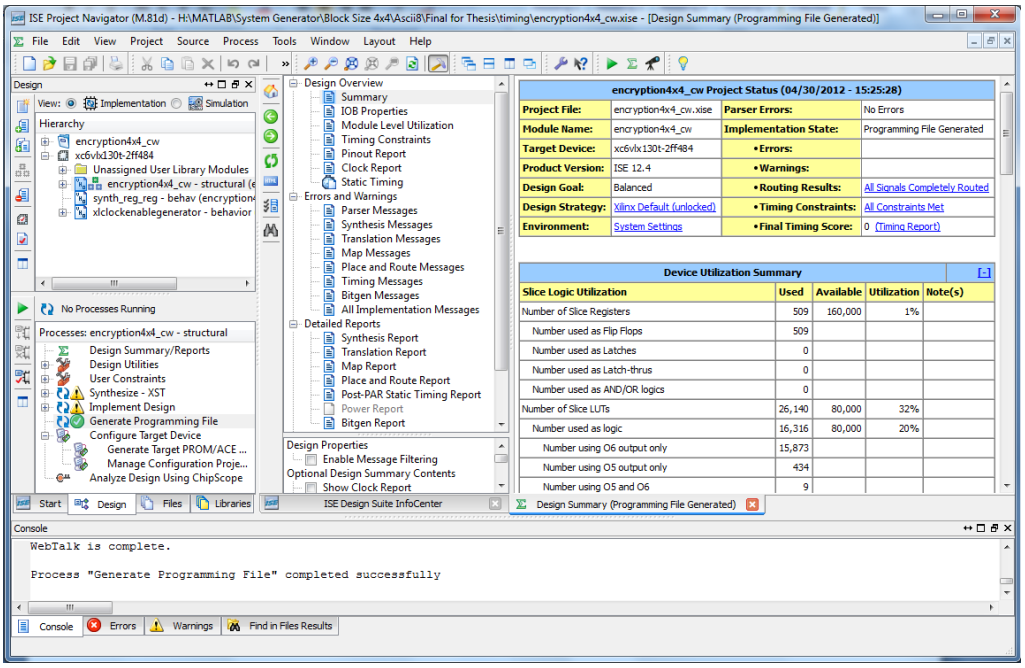


Figure 7.39: ISE project navigator

file process property. Alternatively, the appropriate storage device can be selected through the Create PROM File Formatter in the iMPACT Flow panel, as shown in Figure 7.40. This will format the configuration bitstream file into a PROM which is used later to configure the FPGA device once it is powered up. By running the Generate File in the iMPACT processes panel, as shown in Figure 7.41, iMPACT will generate the PROM files depending on the specifications that have been identified.

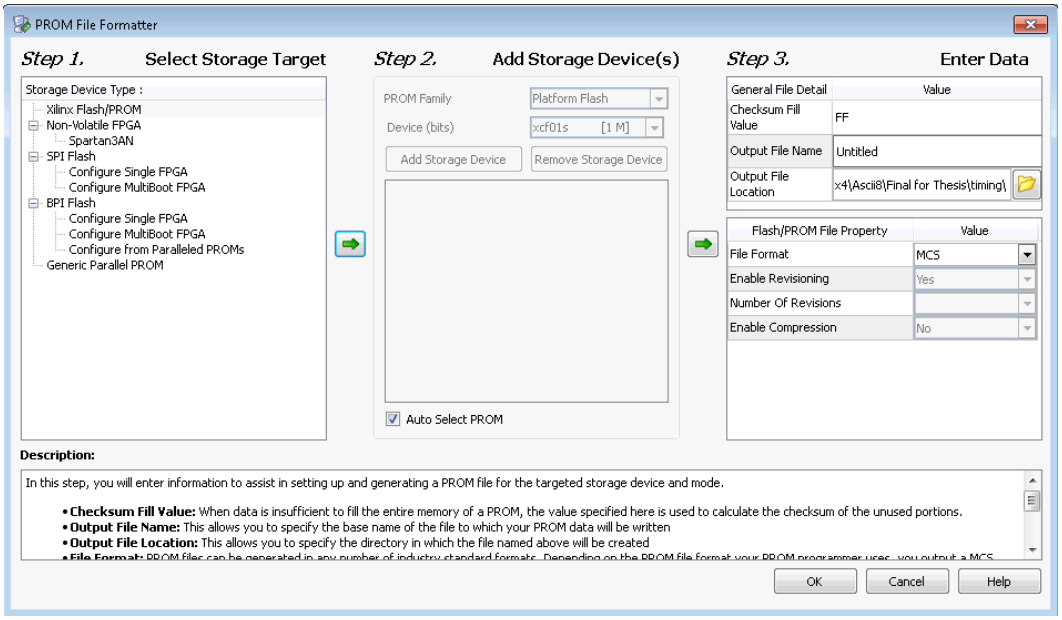


Figure 7.40: PROM File Formatter

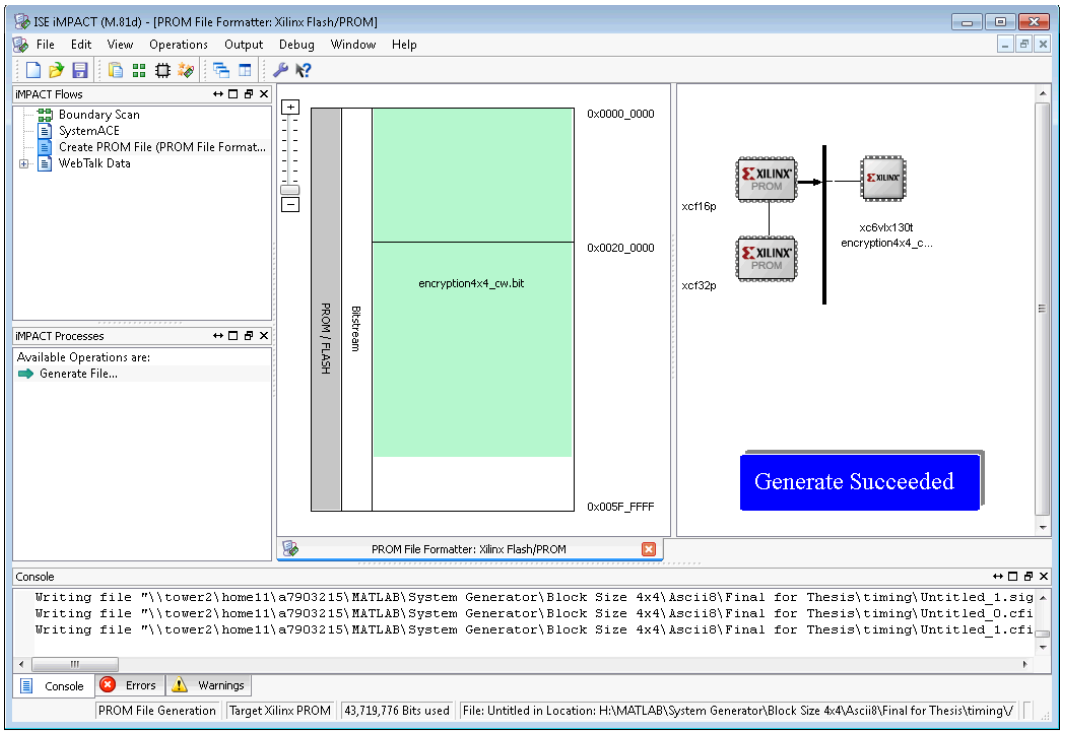


Figure 7.41: Generating PROM File

7.7 Results

The hardware implementation of the proposed cipher is carried out on the target FPGA device of: family, Xilinx 6; port, xc6vlx130t; package, ff484; and speed grade, -2. Two circuits are designed; one for encryption and the other for decryption for a block size of 16×7 bits. In addition another two circuits are designed for a block size of 32×7 bits. The architectures of block size of 16×7 bits are implemented also on a smaller device to compare the effect of device on throughput and power consumption. The results of hardware implementations are summarised in Tables 7.1 and 7.2 for encryption and decryption circuits, respectively, which contain the amount of resources utilised, maximum clock frequency, power consumption, throughput and efficiency. These results, in addition to encryption and decryption, include the generation of the round keys. In the decryption circuits, a delay is imposed due to executing the round keys before starting decryption, as the key should be used in reverse order. In contrast, in the encryption circuit both parts are run in parallel. The results are obtained from the ISE 12.4 Project Navigator shown in Figure 7.39, and a power analysis is shown in Figure 7.42 for an encryption circuit of block size 16×7 bits.

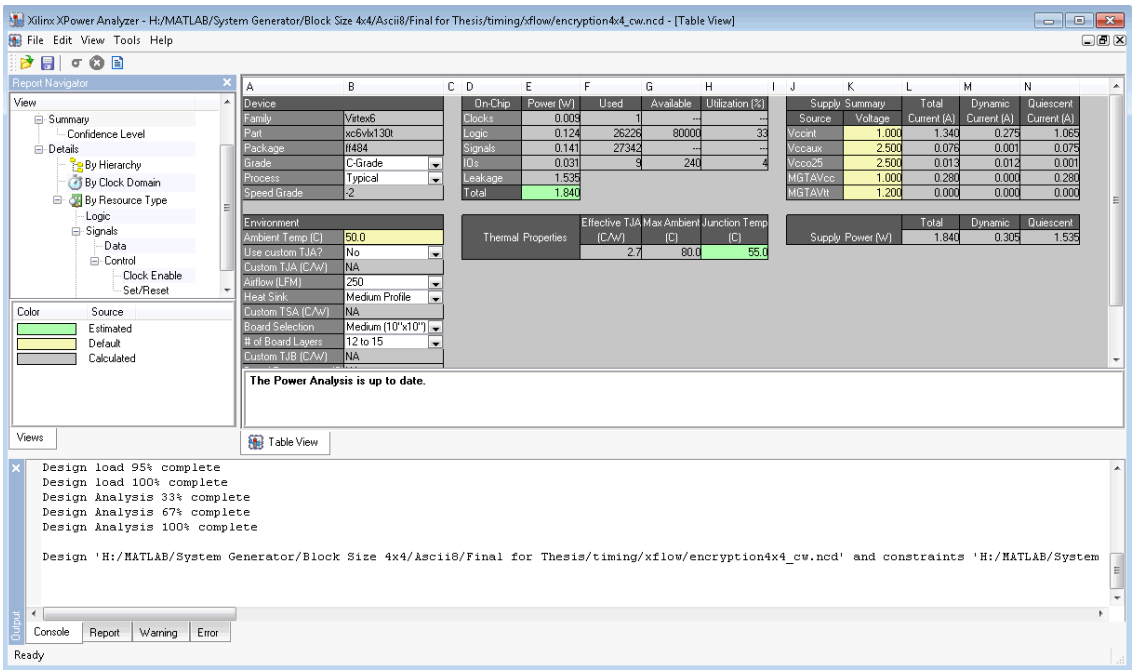


Figure 7.42: Power analysis for encryption circuit of block size 16×7 bits

Table 7.1: FPGA implementation results for encryptions

Encryption	Block Size 16×7 bits		Block Size 32×7 bits
	FPGA Vertix 5 xc5vlx85t-3ff1136	FPGA Vertix 6 xc6vlx130t-2ff484	FPGA Vertix 6 xc6vlx130t-2ff484
No. of Registers	519 out of 51800	509 out of 160000	962 out of 160000
No. of LUTs	26132 out of 51800	26140 out of 80000	58176 out of 80000
No. of Slices	7371 out of 12960	8030 out of 20000	17685 out of 20000
No. of bonded IOBs	9 out of 480	9 out of 240	9 out of 240
Max. Frequency (MHz)	441.112	467.727	431.034
Dynamic Power (W)	0.355	0.295	0.633
Quiescent Power (W)	0.913	1.535	1.548
Total Power (W)	1.268	1.830	2.181
Output every (cycles)	1	1	1
Throughput (Gbits/sec)	42.347	44.902	82.759
Thro./Area (Mb/s/slice)	5.745	5.592	4.680
Energy/bit (nJ/bit)	0.03	0.041	0.026

Table 7.2: FPGA implementation results for decryptions

Decryption	Block Size 16×7 bits		Block Size 32×7 bits
	FPGA Vertix 5 xc5vlx85t-3ff1136	FPGA Vertix 6 xc6vlx130t-2ff484	FPGA Vertix 6 xc6vlx130t-2ff484
No. of Registers	515 out of 51800	518 out of 160000	1260 out of 160000
No. of LUTs	27364 out of 51800	27359 out of 80000	60904 out of 80000
No. of Slices	7706 out of 12960	8366 out of 20000	18392 out of 20000
No. of bonded IOBs	9 out of 480	9 out of 240	9 out of 240
Max. Frequency (MHz)	333.890	373.552	329.489
Dynamic Power (W)	0.386	0.299	0.676
Quiescent Power (W)	0.914	1.535	1.549
Total Power (W)	1.300	1.834	2.225
Output every (cycles)	1	1	1
Throughput (Gbits/sec)	32.053	35.861	63.262
Thro./Area (Mb/s/slice)	4.159	4.287	3.440
Energy/bit (nJ/bit)	0.041	0.051	0.035

The throughput which represents the speed of encryption or decryption process is calculated according to equation 7.10.

$$Throughput = \frac{No.of\ bits\ processed\ \times Clock\ Frequency}{No.of\ Clock\ Cycles\ per\ Output} \qquad 7.10$$

The throughput per area (TPA) which represents the efficiency of implementation in terms of performance, gives an indication of the hardware resource cost related to the throughput achieved.

The cipher itself is processed in one clock cycle regardless of direction (i.e., encryption or decryption) and block size. Although there is a delay of 14-cycle for block size 16×7-bit and 28-cycle for block size 32×7-bit for input conversion from serial to parallel form, this has not been considered in the calculation of the throughput since this step represents the preparation of the input for processing which can be performed in advance in software.

The results of FPGA implementation of the proposed architecture and comparison with AES approaches for encryption core with block size and key length of 128-bit are summarised in Table 7.3.

Table 7.3: Performance comparison of the proposed architecture and AES

Design	Device	Frequency (MHz)	Area (Slices)	Throughput (Gbits/sec)	TPA (Mbits/s/slice)
Sklavos [68]	XCV1000	28.5	17314	3.7	0.21
Hodjat [69]	XC2VP30	168.3	12450	21.5	1.70
Iyer [70]	XC2VP30	206.8	11720	26.5	2.26
Li [71]	XC2V2000	102.8	3223	1.3	0.40
Thongkhome [72]	XC2VP7X	481.3	3119	6.2	1.99
Standaert [128]	XCV3200E	145.0	15112	18.6	1.23
Ours	XC5VLX85T	441.1	7371	42.3	5.75
	XC6VLX130T	467.7	8030	44.9	5.59

7.8 Complexity

The overall complexity of the hardware design for the proposed architectures is summarised in Table 7.4. The look-up tables are used to store the values of the substitution and inverse substitution boxes which are used to non-linearly map the value of the elements in order to satisfy the concept of confusion. The NMNTs are designed based on the split-radix fast algorithm. For transform lengths 8 the multiplication operation is replaced by shift and addition operations, and accordingly the only operations used in the architectures are shift, addition and inverter (NOT).

Table 7.4: Overall system complexity

	Block size 16×7 bits			Block size 32×7 bits		
	Encryption	Decryption	Key	Encryption	Decryption	Key
Word size (bits)	7	7	7	7	7	7
Key size (bits)	16×7	16×7	16×7	32×7	32×7	32×7
Subkey size (bits)	1232	1232	1232	2464	2464	2464
Look-up tables/ Table size	160 (7×7)	160 (7×7)	160 (7×7)	320 (7×7)	320 (7×7)	320 (7×7)
Shift	40	200	40	120	440	120
ADD	1192	1352	1688	2744	3064	3816
NOT	160	386	160	520	922	520

7.9 Conclusions

Efficient hardware architectures for the proposed cipher have been designed and implemented on the target FPGA device XC6VLX130T-2FF484. The Xilinx System Generator of the ISE version 12.4 development suite, which is a system level modelling tool, has been used to facilitate hardware design and reduces design time providing efficient and fast FPGA prototypes.

The design of the proposed cipher, which includes encryption and decryption modules and the key schedule algorithm, is based on the parallel loop-unrolling architecture. Two block sizes and key lengths are considered of 16×7 and 32×7 bits, while the algorithm is designed to support a wide range of lengths. The mode of operation used in the design is a generalised electronic code book, where a block number with other parameters are used in the calculation of the values of the variables

in the variable addition layer, providing different variables for different blocks encrypted with the same key. With this novel improvement a simple mode of operation is implemented with a high level of security, which is reflected in the performance of the system regarding speed and complexity.

There is a trade-off between throughput and area represented by the amount of resources utilised. The cipher starts with an initial key addition followed by ten identical rounds. Each round consists of five sequential layers: element substitution, shifts transform, variable addition, NMNT, and round key addition. The design of the proposed architectures targets throughput, and hence the hardware resources of a single round including the S-boxes are duplicated ten times providing what is known as a loop-unrolling architecture. In addition, the elements within the layers are implemented in parallel so as to further increase the speed of implementation. The round keys are generated on-the-fly on the same device, thus allowing less complexity and resource utilisation. Using such architecture a high throughput is achieved at the cost of area, where a throughput of 44.9 Gbits/sec is achieved with a power consumption of 1.83 W for implementing the encryption module with a key and block lengths of 16×7 bits. This power can be reduced to 1.27 W using a smaller device and the throughput achieved is 42.3 Gbits/sec. For larger block and key lengths (32×7 bits), 82.8 Gbits/sec throughput is achieved for 2.18 W power consumption. In addition, the results of the implementation are compared with those of the AES, and it has been shown that the proposed cipher can run at a higher throughput with a reasonable usage of resources.

Chapter 8

Conclusions and Further Work

This chapter summarises the results of the research presented in this thesis and suggestions are proposed to extend the current research in further work.

8.1 Summary of Research and Results

The following are the major achievements made in the research:

- A new iterated symmetric-key block cipher based on the SPN and NTT with variable block size, key length and word length has been proposed. The block size and key length ranging from 16×P up to 256×P-bit with a power of two, for a word length P = 7, 31, 61 or 127-bit. These parameters can be efficiently determined by the system depending on the message length and processor registers length, by undisclosed these values the task of cryptanalysis becomes almost impossible.
- The NMNT and FNT have been extensively analysed in Chapter 3 to evaluate their diffusion power using two different techniques. The first technique is based on the branch number, while the second technique is based on probabilities. The results confirm that these transforms exhibit generally good diffusion power that in most cases is at a minimum of 50%. The cases that provide low levels of

diffusion less than 50% which obviously arise due to the symmetrical structure of the transforms are overcome in the design by including a variable addition layer. This layer participates also in providing two different ciphertext blocks for the encryption of two identical plaintext blocks using the same cipher key. Consequently, a simple mode of operation can be implemented with high levels of security, which will be reflected in the efficiency of the system regarding speed and complexity.

- Two solutions have been suggested to sort the problem that may arise due to the value of the modulus of the NMNT, where the zero value and the M_p value are retrieved as a zero. This can be avoided either by inserting one bit with a value of 0 at the beginning of each element or by reserving the addresses of the elements that have the M_p value at the end of the ciphertext in order to update their values at the final stage of decryption. In addition, all S-boxes involved in the design based on the NMNT are modified such that the maximum value is swapped to map to itself in order to prevent its appearance after mapping.
- The non-linear properties of all of the S-boxes in the previous standard DES, as well as in the current standard AES, have been analysed in Chapter 4. Consequently the DPP_{max} and IOC_{max} have been computed for each S-box after building up the XOR distribution and linear approximation tables. All S-boxes of the DES have the same DPP_{max} which is equal to 2^{-2}, but the IOC_{max} is not the same for all the 8 S-boxes. The best probability was found in S-box 6, which is $2^{-1.193}$, and the worst was in S-box 5 which is equal to $2^{-0.678}$, with the probabilities of the other S-boxes in between. For the AES S-box the $DPP_{max} = 2^{-6}$ and $IOC_{max} = 2^{-3}$.
- A new 7×7 S-box is generated with $DPP_{max} = 2^{-6}$ and $IOC_{max} = 2^{-2.678}$, by following the same procedures used in the construction of the AES S-box. Furthermore, other S-boxes are derived from the new S-box generated as well as from the AES S-box preserving the same non-linear properties but reordering the output elements with different offsets.
- Two different key schedule algorithms are proposed to generate round keys from a cipher key. The first applies the same round transformation of the cipher for ease of analysis and implementation, while the second is key dependent. It is achieved by replacing both the shifts transform and variable addition layers with key dependent permutation and key dependant addition layers. Both algorithms

extensively use the non-linear mapping and high diffusion mixing to exclude the potential of weak-key and related-key attacks.

- Three categories of test vectors are run in Chapter 5 to verify the correctness of algorithm implementation from any possible error. The results of all of the tests are promising. These tests include the KAT, the MMT, and the MCT, which were initially designed by the NIST to validate the IUT for conformance to the AES algorithm as specified in FIPS 197: AES.
- Differential and linear cryptanalysis have been analysed in Chapter 6 and it has been proven that the proposed cipher is secure against these two powerful well-known types of attack from round three for all block sizes and key lengths. Considering the possibility of these two attacks at the design stage has a major impact on selecting the layers and in constructing the final model. The subsequent consideration of all other types of attack may require minor modifications to the final design, which could be made secure by adding further rounds. However, in order to secure the cipher from other possible attacks and to allow for extra security margins, 10 rounds is determined.
- Parallel loop-unrolling architectures for the proposed cipher are designed and implemented using the FPGA system generator, the details of which are explained in Chapter 7. Using such architecture a high throughput at the cost of area is achieved. For instance, a maximum frequency of 467.727 MHz, a throughput of 44.902 Gbits/sec and a power consumption of 1.83W using 8030 slices is achieved for a block size and key length of $16\times P$, ($P = 7$) on Xilinx Virtex 6 (XC6VLX130T-2FF484) board.

8.2 Recommendations for Further Work

The following suggestions would extend the current research in further work:

- The number of rounds of the algorithm has been chosen to be 10 based on the results obtained from the analysis of differential and linear cryptanalysis, after adding extra rounds in case of stronger attacks and for security margins. However, to choose the right number of rounds precisely, it is required to consider all variants of differential and linear cryptanalysis as well as other types including side-channel attacks, such as power analysis, for example.

- The cipher is designed to work efficiently with variable block sizes and key lengths as well as word lengths. The analysis was carried out on five different sizes, however the hardware architectures are designed only for the first two block sizes with the standard cipher word length of $P = 7$. Implementing other possible word lengths or block sizes can be a task for further work. Where implementing larger block size, the same structure is used with extra resources. While for larger word lengths, in addition to resources, would require extra layers of a word length conversion before and after elements mapping through the element substitution layer.
- The focus in the hardware design was on achieving fast implementation, and parallel loop-unrolling architectures have been designed accordingly which are characterised by high throughput. However, it would be worth to implement the basic architecture as a trade-off between throughput and area usage. Implementing such a scheme with a smaller circuit would have the advantage of reducing the power consumption of the system.
- The FNT has been analysed and it has been proven that it has good diffusion power mostly over 50%, and in addition an example for image encryption has been provided. The structure of the cipher is the same as that based on the NMNT, eliminating the conversion layer, realising a practically faster cipher. A hardware implementation of this scheme can be another task for further work.
- To further complicate the task of cryptanalysis, it is recommended that a compression layer is added at the beginning before processing the encryption; hence the data will be compressed and then ciphered. This layer will provide extra protection against attacks by complicating the statistical analysis of the cipher through eliminating the redundancy in the data. In addition, it will reduce the amount of data processed, which will help in improving system performance.
- Finally, in order to build a complete practical system, it is suggested that the design should be expanded not only provide data confidentiality but also to cover data integrity for authentication purposes.

References

[1] S. Singh, *The Code Book: Science of Secrecy from Ancient Egypt to Quantum Cryptography*: Anchor Books, 2000.

[2] National Bureau of Standards. (1977). *U.S. Department of Commerce, FIPS PUB 46, Data Encryption Standard*. Available: http://www.itl.nist.gov/fipspubs/fip46-2.htm

[3] Joan Daemen and V. Rijmen, *The Design of Rijndael: AES-The Advanced Encryption Standard*. Berlin Heidelberg: Springer, 2002.

[4] R. Kippenhahn, *Code Breaking: A History and Exploration*. London: Constable, 1999.

[5] B. Schneier, *Applied Cryptography: Protocals, Algorithms and Source Code in C*, 2nd ed. New York: John Wiley & Sons, 1996.

[6] H. Delfs and H. Knebl, *Introduction to Cryptography: Principles and Applications* Berlin Heidelberg: Springer, 2007.

[7] R. L. Rivest, *et al.*, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, vol. 21, pp. 120-126, 1978.

[8] A. J. Menezes, *et al.* (1997). *Handbook of Applied Cryptography*. Available: http://www.cacr.math.uwaterloo.ca/hac/

[9] T. Elgamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *IEEE Transactions on Information Theory*, vol. 31, pp. 469-472, 1985.

[10] V. S. Miller, "Use of Elliptic Curves in Cryptography," *Advances in Cryptology, Crypto '85 Proceedings, Lecture Notes in Computer Science,* vol. 218, pp. 417-426, 1986.

[11] N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of Computation*, vol. 48, pp. 203-209, 1987.

[12] W. Stallings, *Cryptography and Network Security: Principles and Practices*, 4th ed. USA: Pearson Education International, 2006.

[13] P. E. Greenwood and M. S. Nikulin, *A Guide to Chi-Squared Testing*. New York: John Wiley & Sons, Inc., 1996.

[14] R. L. Rivest, "The RC4 Encryption Algorithm," *RSA Data Security, Inc.*, 1992.

[15] C. Shannon, "Communication Theory of Secrecy Systems," *Bell System Technical Journal*, vol. 28-4, pp. 656-715, 1949.

[16] Federal Information Processing Standards Publication. (2001). *FIPS197, Advanced Encryption Standard (AES)*. Available: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[17] B. Schneier, "The Blowfish Encryption Algorithm," in *Fast Software Encryption, Cambridge Security Workshop Proceedings, Lecture Notes in Computer Science 809*, ed: Springer-Verlag, 1994, pp. 191-204.

[18] K. Aoki, *et al.*, "Camellia: a 128-bit Block Cipher Suitable for Multiple Platforms - Design and Analysis," *7th Annual International Workshop, SAC 2000, Waterloo, Ontario, Canada, Proceedings, Lecture Notes in Computer Science, Springer-Verlag*, vol. 2012, pp. 39-56, 2001.

[19] C. M. Adams, "Constructing Symmetric Ciphers Using the CAST Design Procedure," *Designs, Codes and Cryptography*, vol. 12, pp. 283-316, 1997.

[20] C. M. Adams, *et al.*, "CAST256: a Submission for the Advanced Encryption Standard," presented at the First AES Candidate Conference (AES1), Ventura, California, USA, 1998.

[21] X. Lai, *et al.*, "Markov Ciphers and Differential Cryptanalysis," *Advances in Cryptology, Eurocrypt '91, Lecture Notes in Computer Science, Springer-Verlag*, vol. 547, pp. 17-38, 1991.

[22] X. Lai, "On the Design and Security of Block Ciphers," *ETH Series in Information Processing. Hartung-Gorre Verlag*, vol. 1, 1992.

[23] C. Burwick, *et al.*, "MARS - a Candidate Cipher for AES," presented at the First AES Candidate Conference (AES1), Ventura, California, USA, 1998.

[24] R. L. Rivest, "The RC5 Encryption Algorithm," *Fast Software Encryption: 2nd International Workshop. Leuven, Belgium. Proceedings, Lecture Notes in Computer Science, Springer-Verlag.*, vol. 1008, pp. 86-96, 1995.

[25] R. L. Rivest, *et al.*, "The RC6 Block Cipher, V1.1," presented at the First AES Candidate Conference (AES1), Ventura, California, USA, 1998.

[26] J. L. Massey, "SAFER K-64: A Byte-Oriented Block-Ciphering Algorithm," in *Fast Software Encryption '93, Lecture Notes in Computer Science 809*, ed: R. Anderson, Ed., Springer-Verlag, 1994, pp. 1-17.

[27] E. Biham, *et al.*, "Serpent: a New Block Cipher Proposal," *Fast Software Encryption, 5th International Workshop, FSE'98, Proceedings, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1372, pp. 222-238, 1998.

[28] National Institute of Standards and Technology. (1998). *U. S. Department of Commerce, Skipjack and KEA algorithm specications*. Available: http://csrc.nist.gov/groups/STM/cavp/documents/skipjack/skipjack.pdf

[29] B. Schneier, *et al.*, *The Twofish Encryption Algorithm*: John Wiley & Sons, 1996.

[30] H. Feistel, "Cryptography and Computer Privacy," *Scientific American*, vol. 228, pp. 15-23, 1973.

[31] A. Shimizu and S. Miyaguchi, "Fast Data Encipherment Algorithm FEAL," *Advances in Cryptology, Eurocrypt '87, Lecture Notes in Computer Science, Springer-Verlag*, vol. 304, pp. 267-278, 1988.

[32] Government Committee of the USSR for Standards (In Russian), "GOST, Gosudarstvennyi Standard 28147-89, Cryptographic Protection for Data Processing Systems," 1989.

[33] R. Merkle, "Fast Software Encryption Functions," *Advances in Cryptology, Crypto '90, Lecture Notes in Computer Science, Springer-Verlag,* vol. 537, pp. 476-501, 1990.

[34] L. Brown, *et al.*, "LOKI - a Cryptographic Primitive for Authentication and Secrecy Applications," *Advances in Cryptology, Proceedings Auscrypt'90, Lecture Notes in Computer Science, Springer-Verlag,* vol. 453, pp. 229-236, 1990.

[35] B. Schneier and J. Kelsey, "Unbalanced Feistel Networks and Block-Cipher Design," *Fast Software Encryption, 3rd International Workshop Proceedings, Springer-Verlag,* pp. 121-144, 1996.

[36] V. T. Hoang and P. Rogaway, "On Generalized Feistel Networks " *Advances in Cryptology, Crypto '10, Lecture Notes in Computer Science, Springer-Verlag,* vol. 6223, pp. 613-630, 2010.

[37] V. Rijmen, *et al.*, "The Cipher Shark," *Fast Software Encryption, 3rd International Workshop Proceedings, Lecture Notes in Computer Science, Springer-Verlag,* vol. 1039, pp. 99-111, 1996.

[38] J. Daemen, *et al.*, "The Block Cipher SQUARE," *Fast Software Encryption, 4th International Workshop Proceedings, Lecture Notes in Computer Science, Springer-Verlag,* vol. 1267, pp. 149-165, 1997.

[39] American National Standards Institute (ANSI) X3.92, "American National Standard for Data Encryption Algorithm (DEA)," 1981.

[40] E. Biham and A. Shamir, "Differential Cryptanalysis of the Full 16-Round DES," *Advances in Cryptology, Proceedings Crypto'92, Springer-Verlag,* vol. 740, pp. 487-496, 1992.

[41] American National Standards Institute (ANSI) X9.52, "American National Standard for Triple Data Encryption Algorithm (TDEA)," 1998.

[42] X. Lai and J. L. Massey, "A Proposal for a New Block Encryption Standard," *Advances in cryptology, Eurocrypt '90, Lecture Notes in Computer Science, Springer-Verlag,* vol. 473, pp. 389 - 404, 1990.

[43] E. Biham and A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems," *Journal of Cryptology,* vol. 4, pp. 3-72, 1991.

[44] P. R. Zimmermann, *The Official PGP User's Guide*: MIT Press, 1995.

[45] W. Meier, "On the Security of the IDEA Block Cipher," *Advances in Cryptology, Eurocrypt '93 Proceedings, Lecture Notes in Computer Science, Springer-Verlag,* vol. 765, pp. 371-385, 1993.

[46] J. Borst, *et al.*, "Two Attacks on Reduced IDEA," *Advances in Cryptology, Eurocrypt '97 Proceedings, Lecture Notes in Computer Science, Springer-Verlag,* vol. 1233, pp. 1-13, 1997.

[47] E. Biham, *et al.*, "Miss-in-the-middle Attacks on IDEA and Khufru," *Fast Software Encryption, 6th International Workshop Proceedings, Lecture Notes in Computer Science, Springer-Verlag,* vol. 1636, pp. 124-138, 1999.

[48] R. Phan, "Impossible Differential Cryptanalysis of 7-Rounds Advanced Encryption Standard (AES)," *Information Processing Letters,* vol. 91, pp. 33-38, 2004.

[49] H. Mala, *et al.*, "Improved Impossible Differential Cryptanalysis of 7-Round AES-128," *Indocrypt '10, Lecture Notes in Computer Science. Springer,* vol. 6498, pp. 282-291, 2010.

[50] O. Dunkelman, *et al.*, "Improved Single-Key Attacks on 8-Round AES-192 and AES-256," *Advances in Cryptology, Asiacrypt '10, Lecture Notes in Computer Science, Springer*, vol. 6477, pp. 158-176, 2010.

[51] Federal Information Processing Standards Publication. (1980). *FIPS PUB 81, DES Modes of Operation*. Available: http://www.itl.nist.gov/fipspubs/fip81.htm

[52] National Institute of Standards and Technology. (2001). *NIST Special Publication 800-38A, Recommendation for Block Cipher Modes of Operation Methods and Techniques*. Available: http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

[53] D. R. Stinson, *Cryptography: Theory and Practice (Discrete Mathematics and Its Applications)*: Chapman & Hall/CRC (Taylor & Francis Group), 2005.

[54] D. Kahn, *The Codebreakers: The Story of Secret Writing*. New York: Macmillan Publishing Co.,, 1967.

[55] L. R. Knudsen, "The Number of Rounds in Block Ciphers," Public report, NESSIE. NES/DOC/UIB/WP3/003, 2000.

[56] M. Matsui, "Linear Cryptanalysis Methods for DES Cipher," in *Advances in Cryptology - Eurocrypt' 93 Proceedings, Lecture Notes in Computer Science*, . vol. 765, ed: T. Helleseth, Ed., Springer-Verlag, 1994, pp. 386-397.

[57] C. M. Maxfield, *FPGAs World Class Designs*: Elsevier, 2009.

[58] R. Woods, *et al.*, *FPGA-Based Implementation of Signal Processing Systems*: John Wiley & Sons Ltd, 2008.

[59] C. M. Maxfield, *The Design Warrior's Guide to FPGAs: Devices, Tools and Flows*: Elsevier, 2004.

[60] R. C. Cofer and B. Harding, *Rapid System Prototyping with FPGAs Accelerating the Design Process*: Elsevier, 2006.

[61] S. Vassiliadis and D. Soudris, *Fine- and Coarse-Grain Reconfigurable Computing*: Springer, 2007.

[62] K. Wong, *et al.*, "A single-chip FPGA Implementation of the Data Encryption Standard (DES) Algorithm," in *Global Telecommunications Conference. GLOBECOM 98. The Bridge to Global Integration. IEEE*, 1998, pp. 827-832 vol.2.

[63] C. Patterson, "High Performance DES Encryption in Virtex FPGAs Using JBits," in *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2000, pp. 113-121.

[64] T. Arich and M. Eleuldj, "Hardware Implementations of the Data Encryption Standard," in *The 14th International Conference on Microelectronics - ICM*, 2002, pp. 100-103.

[65] M. McLoone and J. V. McCanny, "High-performance FPGA Implementation of DES Using a Novel Method for Implementing the Key Schedule," *IEE Proceedings -Circuits, Devices and Systems*, vol. 150, pp. 373-8, 2003.

[66] S. Taherkhani, *et al.*, "Implementation of Non-Pipelined and Pipelined Data Encryption Standard (DES) Using Xilinx Virtex-6 FPGA Technology," in *10th International Conference on Computer and Information Technology (CIT), IEEE* 2010, pp. 1257-1262.

[67] M. McLoone and J. V. McCanny, "Rijndael FPGA Implementation Utilizing Look-Up Tables," *IEEE Workshop on Signal Processing Systems*, pp. 349-360, 2001.

[68] N. Sklavos and O. Koufopavlou, "Architectures and VLSI Implementations of the AES-Proposal Rijndael," *IEEE Transactions on Computers*, vol. 51, pp. 1454-1459, 2002.

[69] A. Hodjat and I. Verbauwhede, " A 21.54 Gbits/s Fully Pipelined AES Processor on FPGA," *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04),* pp. 308-309, 2004.

[70] N. C. Iyer, *et al.*, "High Throughput, low cost, Fully Pipelined Architecture for AES Crypto Chip," in *Annual IEEE India Conference*, 2006, pp. 1-6.

[71] H. Li, "Efficient and Flexible Architecture for AES," *IEE Proceedings -Circuits, Devices and Systems,* vol. 153, pp. 533-538, 2006.

[72] K. Thongkhome, *et al.*, "A FPGA Design of AES Core Architecture for Portable Hard Disk," in *Eighth International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 2011, pp. 223-228.

[73] B. Schneier, *et al.*, "Twofish: A 128-bit Block Cipher," in *First AES Candidate Conference (AES1)*, USA, August 20-22, 1998.

[74] X. B. Yang and S. Boussakta, "A New Development of Symmetric Key Cryptosystem," *International Conference on Communications (ICC 08),* pp. 1546-1550, 2008.

[75] X. B. Yang, *et al.*, "A New Development of Cryptosystem Using New Mersenne Number Transform," in *7th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)*, England, UK, 2010, pp. 701-705.

[76] H. M. Heys and Stafford E. Tavares, "Avalanche Characteristics of Substitution-Permutation Encryption Networks," *IEEE Transactions On Computers,* vol. 44, pp. 1131-1139, 1995.

[77] A. F. Webster and S. E. Tavares, "On the Design of S-boxes," *Advances in Cryptology, Proceedings Crypto'85, Lecture Notes in Computer Science, Springer-Verlag,* vol. 218, pp. 523-534, 1986.

[78] J. B. Kam and G. I. Davida, "Structured Design of Substitution-Permutation Encryption Networks," *IEEE Transactions on Computers*, vol. C-28, pp. 747-753, 1979.

[79] J. H. McClellan and C. M. Rader, *Number Theory in Digital Signal Processing*: Prentice-Hall, 1979.

[80] S. Boussakta and A. G. J. Holt, "Filtering Employing a New Transform," in *OCEANS '94. 'Oceans Engineering for Today's Technology and Tomorrow's Preservation.' Proceedings*, 1994, pp. I/547-I/553.

[81] R. Agarwal and C. Burrus, "Fast Convolution Using Fermat Number Transforms with Applications to Digital Filtering," *IEEE Transactions on Acoustics, Speech and Signal Processing* vol. 22, pp. 87-97, 1974.

[82] S. Boussakta and A. G. J. Holt, "Number Theoretic Transforms and their Applications in Image Processing," *Advances in Imaging and Electron Physics,* vol. 111, pp. 1-90, 1999.

[83] I. S. Reed, *et al.*, "The Fast Decoding of Reed-Solomon Codes Using Number Theoretic Transforms," Jet Propulsion Laboratory, Pasadena, Calif.1976.

[84] D. Kehil and Y. Ferdi, "Signal Encryption Using New Mersenne Number Transform," in *2010 7th International Symposium on Communication Systems Networks and Digital Signal Processing (CSNDSP), ,* England, UK, 2010, pp. 736-740.

[85] Z. Yanqun and W. Qianping, "A New Scrambling Method Based on Arnold and Fermat Number Transformation," in *International Conference on Environmental Science and Information Application Technology, ESIAT* 2009, pp. 624-628.

[86] S. Boussakta and A. G. J. Holt, "New Number Theoretic Transform," *Electronics Letters*, vol. 28, pp. 1683-1684, 1992.

[87] S. Boussakta and A. G. J. Holt, "New Transform Using the Mersenne Numbers," *IEE Proceedings -Vision, Image and Signal Processing,* vol. 142, pp. 381-388, 1995.

[88] S. Boussakta and A. G. J. Holt, "New Two Dimensional Transform," *Electronics Letters,* vol. 29, pp. 949-950, 1993.

[89] S. Boussakta, *et al.*, "3-D Vector Radix Algorithm for the 3-D New Mersenne Number Transform," *IEE Proceedings.-Vis. Image Signal Process.*, vol. 148, pp. 115-125, 2001.

[90] Omar Nibouche, *et al.*, "Pipeline Architectures for Radix-2 New Mersenne Number Transform," *IEEE Transactions on Circuits and Systems,* vol. 56, pp. 1668-1680, 2009.

[91] O. Nibouche, *et al.*, "A New Architecture for Radix-2 New Mersenne Number Transform," *IEEE ICC 2006 proceedings.*, 2006.

[92] S. Boussakta, *et al.*, "Radix-4 Decimation-in-Frequency Algorithm for the New Mersenne Number Transform," *10th IEEE International Conference on Electronics, Circuits and Systems, 2003. ICECS 2003. Proceedings of the 2003* vol. 3, pp. 1133-1136, 2003.

[93] O. Alshibami, *et al.*, "Radix-4 Algorithm for the New Mersenne Number Transform," *5th International Conference on Signal Processing Proceedings, 2000. WCCC-ICSP 2000. ,* vol. 1, pp. 54-56, 2000.

[94] O. Alshibami, *et al.*, "Split-Radix Algorithm for the New Mersenne Number Transform," *The 7th IEEE International Conference on Electronics, Circuits and Systems, 2000. ICECS 2000. ,* vol. 1, pp. 583-586, 2000.

[95] R. Agarwal and C. Burrus, "Fast Digital Convolution Using Fermat Transforms," *Southwest IEEE Conf. Rec., Houston, Tex. ,* pp. 538-543, 1973.

[96] R. C. Agarwal and C. S. Burrus, "Fast One-Dimensional Digital Convolution by Multidimensional Techniques," *IEEE Transactions on Acoustics, Speech and Signal Processing,* vol. ASSP-22, pp. 1-10, 1974.

[97] M. F. Al-Gailani and S. Boussakta, "Evaluation of One-dimensional NMNT for Security Applications," in *7th International Symposium on Communication Systems, Networks and Digital Signal Processing (CSNDSP)* England, UK, 2010, pp. 715-720.

[98] M. F. Al-Gailani, *et al.*, "Fermat Number Transform Diffusion's Analysis," in *IEEE GCC Conference and Exhibition*, Dubai, UAE, 2011, pp. 237-240.

[99] M. F. Al-Gailani and S. Boussakta, "New Mersenne Number Transform Diffusion Power Analysis " *American Journal of Engineering and Applied Sciences,* vol. 4, pp. 461-469, 2011.

[100] H. M. Heys, "A Tutorial on Linear and Differential Cryptanalysis," *Cryptologia,* vol. 26, pp. 189-221, 2002.

[101] J. Seberry, *et al.*, "Pitfalls in Designing Substitution Boxes," *Advances in Cryptology, Proceedings Crypto'94, Lecture Notes in Computer Science, Springer-Verlag,* vol. 839, pp. 383-396, 1994.

[102] T. Jakobsen and L. Knudsen, "The Interpolation Attack Against Block Ciphers," *Fast Software Encryption, 4th International Workshop Proceedings, FSE'97, Lecture Notes in Computer Science, Springer-Verlag,* vol. 1267, pp. 28-40, 1997.

[103] T. Hansen and G. L. Mullen, "Primitive Polynomials Over Finite Fields," *Mathematics of Computation,* vol. 59, pp. 639-643, 1992.

[104] R. Lidl and H. Niederreiter, *Finite Fields* 2nd ed. Cambridge: Cambridge University Press, 2008.

[105] H. M. Heys and S. E. Taveres, "Substitution-Permutation Networks Resisitant to Differential and Linear Cryptanalysis," *Journal of Cryptology*, vol. 9, pp. 1-9, 1996.

[106] L. E. BasshamIII. (2002). *The Advanced Encryption Standard Algorithm Validation Suite (AESAVS)*. Available: http://csrc.nist.gov/groups/STM/cavp/documents/aes/AESAVS.pdf

[107] H. V. Sorensen, *et al.*, "On Computing the Discrete Hartley Transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. VOL. ASSP-33, pp. 1231-1238, 1985.

[108] P. Duhamel, "Implementation of "Split-Radix" FFT Algorithms for Complex, Real, and Real-Symmetric Data," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. ASSP-34, pp. 285-295, 1986.

[109] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Boca Raton: Chapman & Hall/CRC (Taylor & Francis Group), 2008.

[110] B. Preneel, *et al.*, "Principles and Performance of Cryptographic Algorithms," *Dr. Dobb's Journal*, vol. 23, pp. 126-131, 1998.

[111] A. K. Lenstra and E. R. Verheul, "Selecting Cryptographic Key Sizes," *Journal of Cryptology*, vol. 14, pp. 255-293, 2001.

[112] P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," *Advances in Cryptology, Proceedings Crypto'96, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1109, pp. 104-113, 1996.

[113] P. C. Kocher, *et al.*, "Differential Power Analysis," *Advances in Cryptology, Proceedings Crypto'99, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1666, pp. 388-397, 1999.

[114] B. Preneel, *et al.*, "Final Report of European Project Number IST-1999-12324: New European Schemes for Signatures, Integrity and Encryption (NESSIE)," http://www.cryptonessie.org ,April 19, 2004.

[115] C. Bouillaguet, *et al.*, "Low Data Complexity Attacks on AES," Cryptology ePrint Archive, Report 2010/633, 2010.

[116] K. Sakamura, *et al.*, "A Study on the Linear Cryptanalysis of AES Cipher," *Journal of the Faculty of Environmental Science and Technology*, vol. 9, pp. 19-26, 2004.

[117] S. Lucks, "The Saturation Attack - a Bait for Twofish," *Fast Software Encryption, 8th International Workshop Proceedings, Lecture Notes in Computer Science, Springer-Verlag*, vol. 2355, pp. 1-15, 2002.

[118] E. Biham, "New Types of Cryptanalytic Attacks Using Related Keys," *Advances in Cryptology, Proceedings Eurocrypt'93, Lecture Notes in Computer Science, Springer-Verlag*, vol. 765, pp. 398-409, 1994.

[119] J. Smith, "The Design of Lucifer: A Cryptographic Device for Data Communications," *Technical report, IBM T.J. Watson Research Center, USA*, 1971.

[120] A. Biryukov and D. Wagner, "Slide Attacks," *Fast Software Encryption, 6rd International Workshop Proceedings, FSE'99, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1636, pp. 245-259, 1999.

[121] A. Biryukov and D. Wagner, "Advanced Slide Attacks," *Advances in Cryptology, Eurocrypt '00, Lecture Notes in Computer Science, Springer-Verlag*, vol. 1807, pp. 589-606, 2000.

[122] A. Biryukov, *et al.*, "New Weak-Key Classes of IDEA," *4th International Conference in Information and Communications Security (ICICS), Lecture Notes in Computer Science, Springer-Verlag*, vol. 2513, pp. 315-326, 2002.

[123] C. M. Rader., "Discrete convolutions via Mersenne transforms," in *IEEE TRANSACTIONS ON COMPUTERS* vol. C-21, ed, 1972, pp. 1269-1273.

[124] U. Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays*, 3rd ed.: Springer-Verlag Berlin Heidelberg, 2007.

[125] L. Leibowitz, "A Simplified Binary Arithmetic for the Fermat Number Transform," *IEEE Transactions on Acoustics, Speech and Signal Processing,*, vol. 24, pp. 356-359, 1976.

[126] P. Duhamel and H. Hollmann. (1984, Split-radix FFT algorithm. *IET Electrincs Letters 20(1)*, 14-16.

[127] O. Alshibami, "Development of Fast Algorithms for One and Multidimensional Transforms with Application to Digital Signal Processing and Communications," PhD, School of Electronic and Electrical Engineering, University of Leeds, Leeds, 2002.

[128] F.-X. Standaert, *et al.*, "Efficient Implementation of Rijndael Encryption in Reconfigurable Hardware: Improvements and Design Tradeoffs," *CHES 2003, LNCS*, vol. 2779, pp. 334-350, 2003.

Appendix A

NMNT Diffusion Analysis Results

The NMNT diffusion analysis results based on the probabilities (second technique) for different modulus and transform lengths are attached in this appendix.

Table A.1: NMNT diffusion for $P = 7$, $M_p = 127$, $N = 4$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $M_p = 0$	Different Modifi. modulo $M_p \neq 0$
Paired elements *i*, *i*+2 *i*-even (x = 1)	Odd	-	-	-
	Even	-	-	-
	All	50%	50%	100%
Paired elements *i*, *i*+2 *i*-odd (x = 1)	Odd	-	-	-
	Even	-	-	-
	All	50%	50%	100%
Paired elements *i*, *i*+2 *i*-mix (x = 1)	Odd	-	-	-
	Even	-	-	-
	All	25%	50%	75-100%
Paired elements *i*, *i*+1 *i*-even (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	-	-	-
Paired elements *i*, *i*+1 *i*-odd (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	-	-	-
Paired elements *i*, *i*+1 *i*-mix (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	-	-	-
Unpaired elem. Even	Single	100%	100%	100%
	Odd	-	-	-
	Even	-	-	-
Unpaired elem. Odd	Single	100%	100%	100%
	Odd	-	-	-
	Even	-	-	-
Unpaired elem. Mix	Odd	-	-	-
	Even	-	-	-
i, *i*+2, random for others	Odd	50-100%	50-100%	75-100%
	Even	-	-	-

Table A.2: NMNT diffusion for $P = 7$, $M_p = 127$, $N = 8$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $M_p = 0$	Different Modifi. modulo $M_p \neq 0$
Paired elements *i*, *i*+4 *i*-even (x = 1)	Odd	50%	25%	75%
	Even	-	-	-
	All	25%	50%	75-100%
Paired elements *i*, *i*+4 *i*-odd (x = 1)	Odd	50%	50%	100%
	Even	-	-	-
	All	25%	50%	75-100%
Paired elements *i*, *i*+4 *i*-mix (x = 1)	Odd	50%	37.5-50%	75-100%
	Even	25%	37.5-50%	62.5-100%
	All	12.5%	37.5-50%	75-100%
Paired elements *i*, *i*+2 *i*-even (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	75%	75%	100%
Paired elements *i*, *i*+2 *i*-odd (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	50%	50%	100%
Paired elements *i*, *i*+2 *i*-mix (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	62.5%	62.5-75%	75-100%
Unpaired elem. Even	Single	75%	75%	75%
	Odd	-	-	-
	Even	75%	75%	75-100%
Unpaired elem. Odd	Single	100%	100%	100%
	Odd	-	-	-
	Even	50%	50%	50-100%
Unpaired elem. Mix	Odd	-	-	-
	Even	75%	75%	75-100%
i, *i*+4, random for others	Odd	62.5-100%	75-100%	75-100%
	Even	62.5-100%	75-100%	75-100%

Table A.3: NMNT diffusion for $P = 7$, $M_p = 127$, $N = 16$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $M_p = 0$	Different Modifi. modulo $M_p \neq 0$
Paired elements *i*, *i*+8 *i*-even (x = 1)	Odd	37.5%	37.5-50%	75-100%
	Even	25- 37.5%	25-50%	62.5-100%
	All	12.5%	37.5-50%	75-100%
Paired elements *i*, *i*+8 *i*-odd (x = 1)	Odd	50%	25-50%	75-100%
	Even	25%	37.5-50%	75-100%
	All	12.5%	25-50%	75-100%
Paired elements *i*, *i*+8 *i*-mix (x = 1)	Odd	37.5-50%	25-50%	68.8-100%
	Even	12.5-43.8%	31.3-50%	75-100%
	All	6.25%	43.8-50%	87.5-100%
Paired elements *i*, *i*+4 *i*-even (x = 2)	Odd	75%	62.5%	62.5-87.5%
	Even	-	-	-
	All	62.5%	62.5-75%	75-100%
Paired elements *i*, *i*+4 *i*-odd (x = 2)	Odd	50-75%	50-75%	100%
	Even	-	-	-
	All	62.5%	62.5-75%	75-100%
Paired elements *i*, *i*+4 *i*-mix (x = 2)	Odd	75%	62.5-75%	81.3-100%
	Even	62.5%	62.5-75%	81.3-100%
	All	56.25%	56.3-75%	87.5-100%
Unpaired elem. Even	Single	87.5%	87.5%	87.5%
	Odd	87.5%	75-87.5%	75-100%
	Even	62.5-87.5%	62.5-87.5%	62.5-100%
Unpaired elem. Odd	Single	75-100%	75-100%	75-100%
	Odd	75-100%	75-87.5%	75-100%
	Even	50-75%	50-87.5%	50-100%
Unpaired elem. Mix	Odd	75-100%	75-93.8%	75-100%
	Even	62.5-93.8%	62.5-93.8%	75-100%
i, *i*+8, random for others	Odd	81.3-100%	81.3-100%	81.3-100%
	Even	75-100%	75-100%	75-100%

Table A.4: NMNT diffusion for $P = 7$, $M_p = 127$, $N = 32$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $M_p = 0$	Different Modifi. modulo $M_p \neq 0$
Paired elements *i*, *i*+16 *i*-even (x = 1)	Odd	43.8-50%	25-50%	68.8-100%
	Even	12.5- 50%	25-50%	56.3-100%
	All	6.25%	37.5-50%	87.5-100%
Paired elements *i*, *i*+16 *i*-odd (x = 1)	Odd	37.5-50%	25-50%	62.5-100%
	Even	12.5-43.8%	25-50%	50-100%
	All	6.25%	31.3-50%	81.3-100%
Paired elements *i*, *i*+16 *i*-mix (x = 1)	Odd	37.5-50%	34.4-50%	71.9-100%
	Even	18.8-50%	34.4-50%	81.3-100%
	All	3.125%	43.8-50%	90.6-100%
Paired elements *i*, *i*+8 *i*-even (x = 2)	Odd	68.75%	56.3-75%	81.3-100%
	Even	62.5-68.8%	62.5-75%	68.8-100%
	All	56.25%	62.5-75%	87.5-100%
Paired elements *i*, *i*+8 *i*-odd (x = 2)	Odd	50-75%	50-75%	62.5-100%
	Even	62.5%	62.5-75%	75-100%
	All	56.25%	62.5-75%	87.5-100%
Paired elements *i*, *i*+8 *i*-mix (x = 2)	Odd	68.8-75%	62.5-75%	84.4-100%
	Even	56.3-71.9%	62.5-75%	84.4-100%
	All	53.125%	65.6-75%	90.6-100%
Unpaired elem. Even	Single	93.8%	93.75%	93.75%
	Odd	93.8%	68.8-93.8%	68.8-100%
	Even	56.3-100%	56.3-93.8%	68.8-100%
Unpaired elem. Odd	Single	75-100%	75-100%	75-100%
	Odd	75-100%	68.8-93.8%	75-100%
	Even	50-93.8%	50-93.8%	50-100%
Unpaired elem. Mix	Odd	71.9-100%	68.8-96.9%	62.5-100%
	Even	56.3-100%	50-96.9%	68.8-100%
i, *i*+16, random for others	Odd	84.4-100%	87.5-100%	87.5-100%
	Even	84.4-100%	84.4-100%	87.5-100%

Table A.5: NMNT diffusion for $P = 7$, $M_p = 127$, $N = 64$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $M_p = 0$	Different Modifi. modulo $M_p \neq 0$
Paired elements *i*, *i*+32 *i*-even (x = 1)	Odd	43.8-50%	37.5-50%	78.1-100%
	Even	9.4- 50%	34.4-50%	75-100%
	All	3.125%	46.9-50%	93.8-100%
Paired elements *i*, *i*+32 *i*-odd (x = 1)	Odd	37.5-50%	25-50%	62.5-100%
	Even	12.5-50%	25-50%	62.5-100%
	All	3.125%	43.8-50%	90.6-100%
Paired elements *i*, *i*+32 *i*-mix (x = 1)	Odd	35.9-50%	37.5-50%	75-100%
	Even	21.9-50%	40.6-50%	84.4-100%
	All	1.56%	46.9-50%	95.3-100%
Paired elements *i*, *i*+16 *i*-even (x = 2)	Odd	65.6-75%	62.5-75%	84.4-100%
	Even	56.3-75%	62.5-75%	78.1-100%
	All	53.125%	68.8-75%	93.8-100%
Paired elements *i*, *i*+16 *i*-odd (x = 2)	Odd	50-75%	50-75%	62.5-100%
	Even	56.3-71.9%	62.5-75%	81.3-100%
	All	53.125%	68.8-75%	90.6-100%
Paired elements *i*, *i*+16 *i*-mix (x = 2)	Odd	65.6-75%	65.6-75%	84.4-100%
	Even	57.8-75%	64.1-75%	90.6-100%
	All	51.56%	71.9-75%	95.3-100%
Unpaired elem. Even	Single	96.9%	96.9%	96.9%
	Odd	90.6-100%	71.9-96.9%	75-100%
	Even	59.4-100%	62.5-96.9%	78.1-100%
Unpaired elem. Odd	Single	75-100%	75-100%	75-100%
	Odd	71.9-100%	68.8-96.9%	75-100%
	Even	50-100%	50-96.9%	62.5-100%
Unpaired elem. Mix	Odd	73.4-100%	68.8-98.4%	73.4-100%
	Even	62.5-100%	62.5-98.4%	81.3-100%
i, *i*+32, random for others	Odd	87.5-100%	90.6-100%	87.5-100%
	Even	89.1-100%	87.5-100%	90.6-100%

Table A.6: NMNT diffusion for $P = 7$, $M_p = 127$, $N = 128$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $M_p = 0$	Different Modifi. modulo $M_p \neq 0$
Paired elements *i*, *i*+64 *i*-even (x = 1)	Odd	45.3-50%	37.5-50%	82.8-100%
	Even	10.9- 50%	40.6-50%	82.8-100%
	All	1.563%	46.9-50%	95.3-100%
Paired elements *i*, *i*+64 *i*-odd (x = 1)	Odd	35.9-50%	25-50%	68.8-100%
	Even	9.4-50%	25-50%	62.5-100%
	All	1.563%	48.4-50%	95.3-100%
Paired elements *i*, *i*+64 *i*-mix (x = 1)	Odd	36.7-50%	43.8-50%	75-100%
	Even	23.4-50%	43.8-50%	92.2-100%
	All	0.781%	47.7-50%	97.7-100%
Paired elements *i*, *i*+32 *i*-even (x = 2)	Odd	67.2-75%	65.6-75%	85.9-100%
	Even	54.7 – 75%	62.575%	89.1-100%
	All	51.56%	71.9-75%	95.3-100%
Paired elements *i*, *i*+32 *i*-odd (x = 2)	Odd	50-75%	50-75%	62.5-100%
	Even	56-75%	62.5-75%	68.8-100%
	All	51.56%	71.9-75%	95.3-100%
Paired elements *i*, *i*+32 *i*-mix (x = 2)	Odd	66.4-75%	68.8-75%	86.7-100%
	Even	60.9%	68.8-75%	92.2-100%
	All	50.78%	72.7-75%	96.9-100%
Unpaired elem. Even	Single	98.4%	98.4%	98.4%
	Odd	92.2-100%	81.3-98.4%	85.9-100%
	Even	60.9-100%	73.4-98.4%	73.4-100%
Unpaired elem. Odd	Single	75-100%	75-100%	75-100%
	Odd	75-100%	68.8-98.4%	71.4-100%
	Even	50-100%	50-98.4%	81.3-100%
Unpaired elem. Mix	Odd	75-100%	75-99.2%	87.5-100%
	Even	73.4-100%	73.4-99.2%	85.9-100%
i, *i*+64, random for others	Odd	93.8-100%	93-100%	93-100%
	Even	93-100%	93-100%	93-100%

Table A.7: NMNT diffusion for $P = 17$, $M_p = 131071$, $N = 4$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $M_p = 0$	Different Modifi. modulo $M_p \neq 0$
Paired elements *i*, *i*+2 *i*-even (x = 1)	Odd	-	-	-
	Even	-	-	-
	All	50%	50%	100%
Paired elements *i*, *i*+2 *i*-odd (x = 1)	Odd	-	-	-
	Even	-	-	-
	All	50%	50%	100%
Paired elements *i*, *i*+2 *i*-mix (x = 1)	Odd	-	-	-
	Even	-	-	-
	All	25%	50%	100%
Paired elements *i*, *i*+1 *i*-even (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	-	-	-
Paired elements *i*, *i*+1 *i*-odd (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	-	-	-
Paired elements *i*, *i*+1 *i*-mix (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	-	-	-
Unpaired elem. Even	Single	100%	100%	100%
	Odd	-	-	-
	Even	-	-	-
Unpaired elem. Odd	Single	100%	100%	100%
	Odd	-	-	-
	Even	-	-	-
Unpaired elem. Mix	Odd	-	-	-
	Even	-	-	-
i, *i*+2, random for others	Odd	100%	100%	100%
	Even	-	-	-

Table A.8: NMNT diffusion for $P = 17$, $M_p = 131071$, $N = 8$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $M_p = 0$	Different Modifi. modulo $M_p \neq 0$
Paired elements *i*, *i*+4 *i*-even (x = 1)	Odd	50%	25%	75%
	Even	-	-	-
	All	25%	50%	100%
Paired elements *i*, *i*+4 *i*-odd (x = 1)	Odd	50%	50%	100%
	Even	-	-	-
	All	25%	50%	100%
Paired elements *i*, *i*+4 *i*-mix (x = 1)	Odd	50%	50%	100%
	Even	25%	37.5-50%	100%
	All	12.5%	37.5-50%	100%
Paired elements *i*, *i*+2 *i*-even (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	75%	75%	100%
Paired elements *i*, *i*+2 *i*-odd (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	50%	50%	100%
Paired elements *i*, *i*+2 *i*-mix (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	62.5%	75%	100%
Unpaired elem. Even	Single	75%	75%	75%
	Odd	-	-	-
	Even	75%	75%	100%
Unpaired elem. Odd	Single	100%	100%	100%
	Odd	-	-	-
	Even	50%	50%	100%
Unpaired elem. Mix	Odd	-	-	-
	Even	75%	75%	100%
i, *i*+4, random for others	Odd	100%	100%	100%
	Even	100%	100%	100%

Table A.9: NMNT diffusion for $P = 17$, $M_p = 131071$, $N = 16$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $M_p = 0$	Different Modifi. modulo $M_p \neq 0$
Paired elements *i*, *i*+8 *i*-even (x = 1)	Odd	37.5%	50%	87.5-100%
	Even	25- 37.5%	50%	87.5-100%
	All	12.5%	50%	100%
Paired elements *i*, *i*+8 *i*-odd (x = 1)	Odd	50%	25-50%	75-100%
	Even	25%	50%	100%
	All	12.5%	50%	100%
Paired elements *i*, *i*+8 *i*-mix (x = 1)	Odd	37.5-50%	43.8-50%	93.8-100%
	Even	12.5-43.8%	37.5-50%	93.8-100%
	All	6.25%	50%	100%
Paired elements *i*, *i*+4 *i*-even (x = 2)	Odd	75%	62.5%	87.5%
	Even	-	-	-
	All	62.5%	75%	87.5-100%
Paired elements *i*, *i*+4 *i*-odd (x = 2)	Odd	50-75%	50-75%	100%
	Even	-	-	-
	All	62.5%	75%	100%
Paired elements *i*, *i*+4 *i*-mix (x = 2)	Odd	75%	75%	100%
	Even	62.5%	75%	100%
	All	56.25%	75%	100%
Unpaired elem. Even	Single	87.5%	87.5%	87.5%
	Odd	87.5%	87.5%	100%
	Even	62.5-87.5%	62.5-87.5%	87.5-100%
Unpaired elem. Odd	Single	75-100%	75-100%	75-100%
	Odd	75-100%	87.5%	100%
	Even	50-75%	50-87.5%	100%
Unpaired elem. Mix	Odd	75-100%	75-93.8%	100%
	Even	62.5-93.8%	87.5-93.8%	93.8-100%
i, *i*+8, random for others	Odd	93.8-100%	100%	93.8-100%
	Even	93.8-100%	93.8-100%	93.8-100%

Table A.10: NMNT diffusion for $P = 17$, $M_p = 131071$, $N = 32$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $M_p = 0$	Different Modifi. modulo $M_p \neq 0$
Paired elements *i*, *i*+16 *i*-even (x = 1)	Odd	43.8-50%	50%	93.8-100%
	Even	12.5- 50%	43.8-50%	87.5-100%
	All	6.25%	50%	100%
Paired elements *i*, *i*+16 *i*-odd (x = 1)	Odd	37.5-50%	25-50%	75-100%
	Even	12.5-43.8%	43.8-50%	87.5-100%
	All	6.25%	50%	100%
Paired elements *i*, *i*+16 *i*-mix (x = 1)	Odd	37.5-50%	43.8-50%	93.8-100%
	Even	18.8-50%	43.8-50%	93.8-100%
	All	3.125%	50%	100%
Paired elements *i*, *i*+8 *i*-even (x = 2)	Odd	68.75%	75%	93.8-100%
	Even	62.5-68.8%	75%	93.8-100%
	All	56.25%	75%	93.8-100%
Paired elements *i*, *i*+8 *i*-odd (x = 2)	Odd	50-75%	50-75%	87.5-100%
	Even	62.5%	75%	100%
	All	56.25%	75%	100%
Paired elements *i*, *i*+8 *i*-mix (x = 2)	Odd	68.8-75%	71.9-75%	96.9-100%
	Even	56.3-71.9%	71.9-75%	96.9-100%
	All	53.125%	75%	96.9-100%
Unpaired elem. Even	Single	93.75%	93.75%	93.75%
	Odd	93.8-100%	81.3-93.8%	93.8-100%
	Even	56.3-100%	68.8-93.8%	93.8-100%
Unpaired elem. Odd	Single	75-100%	75-100%	75-100%
	Odd	75-100%	81.3-93.8%	87.5-100%
	Even	50-93.8%	50-93.8%	87.5-100%
Unpaired elem. Mix	Odd	75-100%	81.3-96.9%	81.3-100%
	Even	56.3-100%	81.3-96.9%	93.8-100%
i, *i*+16, random for others	Odd	96.9-100%	96.9-100%	96.9-100%
	Even	93.8-100%	96.9-100%	96.9-100%

Table A.11: NMNT diffusion for $P = 17$, $M_p = 131071$, $N = 64$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $M_p = 0$	Different Modifi. modulo $M_p \neq 0$
Paired elements *i*, *i*+32 *i*-even (x = 1)	Odd	46.9-50%	43.8-50%	90.6-100%
	Even	9.4-50%	43.8-50%	90.6-100%
	All	3.125%	50%	100%
Paired elements *i*, *i*+32 *i*-odd (x = 1)	Odd	37.5-50%	25-50%	75-100%
	Even	12.5-50%	37.5-50%	87.5100%
	All	3.125%	50%	100%
Paired elements *i*, *i*+32 *i*-mix (x = 1)	Odd	37.5-50%	46.9-50%	75-100%
	Even	21.9-50%	46.9-50%	87.5-100%
	All	1.563%	50%	100%
Paired elements *i*, *i*+16 *i*-even (x = 2)	Odd	71.9-75%	71.9-75%	96.9-100%
	Even	56.3-75%	71.9-75%	90.6-100%
	All	53.125%	75%	100%
Paired elements *i*, *i*+16 *i*-odd (x = 2)	Odd	50-75%	50-75%	87.5-100%
	Even	56.3-71.9%	71.9-75%	93.8-100%
	All	53.125%	71.9-75%	100%
Paired elements *i*, *i*+16 *i*-mix (x = 2)	Odd	68.8-75%	71.9-75%	87.5-100%
	Even	59.4-75%	71.9-75%	96.9-100%
	All	51.56%	73.4-75%	98.4-100%
Unpaired elem. Even	Single	96.9%	96.9%	96.9%
	Odd	96.9-100%	84.4-96.9%	90.6-100%
	Even	59.4-100%	71.4-96.9%	90.6-100%
Unpaired elem. Odd	Single	75-100%	75-100%	75-100%
	Odd	75-100%	81.3-96.9%	93.8-100%
	Even	50-100%	50-96.9%	87.5-100%
Unpaired elem. Mix	Odd	75-100%	75-98.4%	90.6-100%
	Even	62.5-100%	87.5-98.4%	93.8-100%
i, *i*+32, random for others	Odd	98.4-100%	98.4-100%	98.4-100%
	Even	96.9-100%	96.9-100%	96.9-100%

Table A.12: NMNT diffusion for $P = 17$, $M_p = 131071$, $N = 128$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $M_p = 0$	Different Modifi. modulo $M_p \neq 0$
Paired elements *i*, *i*+64 *i*-even (x = 1)	Odd	48.4-50%	46.9-50%	95.3-100%
	Even	10.9-50%	46.9-50%	95.3-100%
	All	1.563%	48.4-50%	100%
Paired elements *i*, *i*+64 *i*-odd (x = 1)	Odd	37.5-50%	25-50%	75-100%
	Even	9.4-50%	46.9-50%	87.5-100%
	All	1.563%	50%	100%
Paired elements *i*, *i*+64 *i*-mix (x = 1)	Odd	37.5-50%	48.4-50%	98.4-100%
	Even	23.4-50%	48.4-50%	98.4-100%
	All	0.781%	50%	99.2-100%
Paired elements *i*, *i*+32 *i*-even (x = 2)	Odd	73.4-75%	71.9-75%	95.3-100%
	Even	54.7-75%	71.9-75%	95.3-100%
	All	51.56%	75%	100%
Paired elements *i*, *i*+32 *i*-odd (x = 2)	Odd	50-75%	50-75%	87.5-100%
	Even	56.3-75%	71.9-75%	93.8-100%
	All	51.56%	75%	100%
Paired elements *i*, *i*+32 *i*-mix (x = 2)	Odd	68.8-75%	73.4-75%	98.4-100%
	Even	60.9-75%	73.4-75%	98.4-100%
	All	50.78%	75%	100%
Unpaired elem. Even	Single	98.4%	98.4%	98.4-100%
	Odd	98.4-100%	87.5-98.4%	95.3-100%
	Even	60.9-100%	73.4-98.4%	95.3-100%
Unpaired elem. Odd	Single	75-100%	75-100%	75-100%
	Odd	75-100%	81.3-98.4%	93.8-100%
	Even	50-100%	50-98.4%	87.5-100%
Unpaired elem. Mix	Odd	75-100%	87.5-99.2%	95.3-100%
	Even	73.4-100%	96.9-99.2%	98.4-100%
i, *i*+64, random for others	Odd	98.4-100%	98.4-100%	98.4-100%
	Even	99.2-100%	99.2-100%	98.4-100%

Table A.13: NMNT diffusion for $P = 17$, $M_p = 131071$, $N = 256$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $M_p = 0$	Different Modifi. modulo $M_p \neq 0$
Paired elements *i*, *i*+128 *i*-even (x = 1)	Odd	49.2-50%	48.4-50%	97.7-100%
	Even	21.1-50%	48.4-50%	97.7-100%
	All	0.781%	50%	100%
Paired elements *i*, *i*+128 *i*-odd (x = 1)	Odd	37.5-50%	25-50%	75-100%
	Even	18.8-50%	43.8-50%	87.5-100%
	All	0.781%	50%	100%
Paired elements *i*, *i*+128 *i*-mix (x = 1)	Odd	37.5-50%	49.2-50%	99.2-100%
	Even	24.2-50%	49.2-50%	99.2-100%
	All	0.391%	50%	100%
Paired elements *i*, *i*+64 *i*-even (x = 2)	Odd	73.4-75%	73.4-75%	97.7-100%
	Even	55.5-75%	73.4-75%	97.7-100%
	All	50.78%	75%	100%
Paired elements *i*, *i*+64 *i*-odd (x = 2)	Odd	50-75%	50-75%	87.5-100%
	Even	54.7-75%	73.4-75%	93.8-100%
	All	50.78%	75%	100%
Paired elements *i*, *i*+64 *i*-mix (x = 2)	Odd	68.8-75%	74.2-75%	99.2-100%
	Even	61.7-75%	74.2-75%	78.8-100%
	All	50.39%	74.6-75%	100%
Unpaired elem. Even	Single	99.2%	99.2%	99.2%
	Odd	98.4-100%	87.5-99.2%	97.7-100%
	Even	71.1-100%	74.2-99.2%	97.7-100%
Unpaired elem. Odd	Single	75-100%	75-100%	75-100%
	Odd	75-100%	87.5-99.2%	95.3-100%
	Even	50-100%	50-99.2%	87.5-100%
Unpaired elem. Mix	Odd	87.5-100%	87.5-99.6%	98.4-100%
	Even	86.7-100%	98.4-99.6%	99.2-100%
i, *i*+128, random for others	Odd	99.2-100%	99.2-100%	99.2-100%
	Even	99.2-100%	99.2-100%	99.2-100%

Table A.14: NMNT diffusion for $P = 17$, $M_p = 131071$, $N = 512$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $M_p = 0$	Different Modifi. modulo $M_p \neq 0$
Paired elements *i*, *i*+256 *i*-even (x = 1)	Odd	49.2-50%	49.2-50%	98.8-100%
	Even	23.8-50%	49.2-50%	98.8-100%
	All	0.391%	50%	100%
Paired elements *i*, *i*+256 *i*-odd (x = 1)	Odd	37.5-50%	25-50%	75-100%
	Even	18.8-50%	49.2-50%	87.5-100%
	All	0.391%	50%	100%
Paired elements *i*, *i*+256 *i*-mix (x = 1)	Odd	37.5-50%	49.6-50%	99.6-100%
	Even	24.6-50%	49.6-50%	99.6-100%
	All	0.195%	50%	100%
Paired elements *i*, *i*+128 *i*-even (x = 2)	Odd	73.8-75%	74.2-75%	98.8-100%
	Even	60.5-75%	74.2-75%	98.8-100%
	All	50.39%	75%	100%
Paired elements *i*, *i*+128 *i*-odd (x = 2)	Odd	50-75%	50-75%	87.5-100%
	Even	59.4-75%	73.4-75%	93.8-100%
	All	50.39%	75%	99.8-100%
Paired elements *i*, *i*+128 *i*-mix (x = 2)	Odd	68.8-75%	74.6-75%	99.6-100%
	Even	62.1-75%	74.6-75%	99.4-100%
	All	50.195%	74.8-75%	99.8-100%
Unpaired elem. Even	Single	99.61%	99.61%	99.61%
	Odd	98.8-100%	93-99.6%	98.8-100%
	Even	74.6-100%	74.6-99.6%	98.8-100%
Unpaired elem. Odd	Single	75-100%	75-100%	75-100%
	Odd	87.5-100%	87.5-99.6%	96.9-100%
	Even	50-100%	50-99.6%	87.5-100%
Unpaired elem. Mix	Odd	96.9-100%	96.9-99.8%	99.2-100%
	Even	96.5-100%	99.2-99.8%	99.6-100%
i, *i*+256, random for others	Odd	99.6-100%	99.6-100%	99.6-100%
	Even	99.6-100%	99.6-100%	99.6-100%

Table A.15: NMNT diffusion for $P = 17$, $M_p = 131071$, $N = 1024$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $M_p = 0$	Different Modifi. modulo $M_p \neq 0$
Paired elements *i*, *i*+512 *i*-even (x = 1)	Odd	49.4-50%	49.6-50%	99.4-100%
	Even	24.4-50%	49.6-50%	99.4-100%
	All	0.195%	50%	100%
Paired elements *i*, *i*+512 *i*-odd (x = 1)	Odd	37.5-50%	25-50%	75-100%
	Even	18.8-50%	49.6-50%	87.5-100%
	All	0.195%	50%	100%
Paired elements *i*, *i*+512 *i*-mix (x = 1)	Odd	37.5-50%	49.8-50%	99.8-100%
	Even	24.8-50%	49.8-50%	99.8-100%
	All	0.0977%	50%	100%
Paired elements *i*, *i*+256 *i*-even (x = 2)	Odd	74.4-75%	74.6-75%	99.4-100%
	Even	61.9-75%	74.6-75%	99.4-100%
	All	50.2%	75%	99.8-100%
Paired elements *i*, *i*+256 *i*-odd (x = 2)	Odd	50-75%	50-75%	87.5-100%
	Even	59.4-75%	74.2-75%	93.8-100%
	All	50.2%	75%	100%
Paired elements *i*, *i*+256 *i*-mix (x = 2)	Odd	68.8-75%	74.8-75%	99.8-100%
	Even	62.3-75%	74.8-75%	99.8-100%
	All	50.1%	75%	99.9-100%
Unpaired elem. Even	Single	99.8%	99.8%	99.8%
	Odd	99.4-100%	96.7-99.8%	99.4-100%
	Even	74.8-100%	74.8-99.8%	99.4-100%
Unpaired elem. Odd	Single	75-100%	75-100%	75-100%
	Odd	96.9-100%	96.9-99.8%	99.6-100%
	Even	50-100%	50-99.8%	93.8-100%
Unpaired elem. Mix	Odd	99.1-100%	98.4-99.9%	99.6-100%
	Even	98.3-100%	99.6-99.9%	99.8-100%
i, *i*+512, random for others	Odd	99.7-100%	99.8-100%	99.8-100%
	Even	99.8-100%	99.8-100%	99.8-100%

Appendix B

FNT Diffusion Analysis Results

This appendix provides the diffusion analysis results of the FNT based on the probabilities (second technique) with different modulus and transform lengths.

Table B.1: FNT diffusion for $t = 3$, $F_t = 257$, $N = 4$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $F_t = 0$	Different Modifi. modulo $F_t \neq 0$
Paired elements *i*, *i*+2 *i*-even (x = 1)	Odd	-	-	-
	Even	-	-	-
	All	50%	50%	100%
Paired elements *i*, *i*+2 *i*-odd (x = 1)	Odd	-	-	-
	Even	-	-	-
	All	50%	50%	100%
Paired elements *i*, *i*+2 *i*-mix (x = 1)	Odd	-	-	-
	Even	-	-	-
	All	25%	25-50%	75-100%
Paired elements *i*, *i*+1 *i*-even (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	-	-	-
Paired elements *i*, *i*+1 *i*-odd (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	-	-	-
Paired elements *i*, *i*+1 *i*-mix (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	-	-	-
Unpaired elem. Even	Single	100%	100%	100%
	Odd	-	-	-
	Even	-	-	-
Unpaired elem. Odd	Single	100%	100%	100%
	Odd	-	-	-
	Even	-	-	-
Unpaired elem. Mix	Odd	-	-	-
	Even	-	-	-
i, *i*+2, random for others	Odd	50-100%	50-100%	75-100%
	Even	-	-	-

Table B.2: FNT diffusion for $t = 3$, $F_t = 257$, $N = 8$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $F_t = 0$	Different Modifi. modulo $F_t \neq 0$
Paired elements *i*, *i*+4 *i*-even (x = 1)	Odd	50%	50%	100%
	Even	-	-	-
	All	25%	25-50%	75-100%
Paired elements *i*, *i*+4 *i*-odd (x = 1)	Odd	50%	50%	100%
	Even	-	-	-
	All	25%	25-50%	75-100%
Paired elements *i*, *i*+4 *i*-mix (x = 1)	Odd	50%	37.5-50%	87.5-100%
	Even	37.5%	37.5-50%	87.5-100%
	All	12.5%	25-50%	87.5-100%
Paired elements *i*, *i*+2 *i*-even (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	75%	75%	75-100%
Paired elements *i*, *i*+2 *i*-odd (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	75%	75%	75-100%
Paired elements *i*, *i*+2 *i*-mix (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	62.5%	62.5-75%	75-100%
Unpaired elem. Even	Single	100%	100%	100%
	Odd	-	-	-
	Even	75%	75%	75-100%
Unpaired elem. Odd	Single	100%	100%	100%
	Odd	-	-	-
	Even	75%	75%	75-100%
Unpaired elem. Mix	Odd	-	-	-
	Even	87.5%	87.5%	87.5-100%
i, *i*+4, random for others	Odd	75-100%	75-100%	75-100%
	Even	75-100%	75-100%	87.5-100%

Table B.3: FNT diffusion for $t = 3$, $F_t = 257$, $N = 16$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $F_t = 0$	Different Modifi. modulo $F_t \neq 0$
Paired elements *i*, *i*+8 *i*-even (x = 1)	Odd	50%	37.5-50%	87.5-100%
	Even	25- 37.5%	25-50%	75-100%
	All	12.5%	25-50%	87.5-100%
Paired elements *i*, *i*+8 *i*-odd (x = 1)	Odd	50%	37.5-50%	75-100%
	Even	25-37.5%	25-50%	75-100%
	All	12.5%	25-50%	87.5-100%
Paired elements *i*, *i*+8 *i*-mix (x = 1)	Odd	50%	37.5-50%	87.5-100%
	Even	18.8-50%	25-50%	81.3-100%
	All	6.25%	43.8-50%	81.3-100%
Paired elements *i*, *i*+4 *i*-even (x = 2)	Odd	75%	75%	75-100%
	Even	-	-	-
	All	62.5%	62.5-75%	75-100%
Paired elements *i*, *i*+4 *i*-odd (x = 2)	Odd	75%	75%	75-100%
	Even	-	-	-
	All	62.5%	62.5-75%	75-100%
Paired elements *i*, *i*+4 *i*-mix (x = 2)	Odd	75%	68.8-75%	87.5-100%
	Even	68.75%	68.8-75%	87.5-100%
	All	56.25%	62.5-75%	87.5-100%
Unpaired elem. Even	Single	100%	100%	100%
	Odd	100%	75-87.5%	75-100%
	Even	62.5-87.5%	75-87.5%	75-100%
Unpaired elem. Odd	Single	100%	100%	100%
	Odd	100%	75-87.5%	75-100%
	Even	62.5-87.5%	75-87.5%	75-100%
Unpaired elem. Mix	Odd	100%	75-93.8%	75-100%
	Even	62.5-100%	75-93.8%	81.3-100%
i, *i*+8, random for others	Odd	87.5-100%	81.3-100%	87.5-100%
	Even	81.3-100%	81.3-100%	87.5-100%

Table B.4: FNT diffusion for $t = 3$, $F_t = 257$, $N = 32$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $F_t = 0$	Different Modifi. modulo $F_t \neq 0$
Paired elements *i*, *i*+16 *i*-even (x = 1)	Odd	50%	37.5-50%	87.5-100%
	Even	12.5- 50%	25-50%	50-100%
	All	6.25%	43.8-50%	87.5-100%
Paired elements *i*, *i*+16 *i*-odd (x = 1)	Odd	50%	31.3-50%	87.5-100%
	Even	12.5-50%	25-50%	75-100%
	All	6.25%	43.8-50%	87.5-100%
Paired elements *i*, *i*+16 *i*-mix (x = 1)	Odd	50%	37.5-50%	87.5-100%
	Even	21.9-50%	37.5-50%	87.5-100%
	All	3.125%	43.8-50%	93.8-100%
Paired elements *i*, *i*+8 *i*-even (x = 2)	Odd	75%	62.5-75%	75-100%
	Even	62.5-68.8%	62.5-75%	87.5-100%
	All	56.25%	62.5-75%	87.5-100%
Paired elements *i*, *i*+8 *i*-odd (x = 2)	Odd	75%	62.5-75%	75-100%
	Even	62.5-68.8%	62.5-75%	87.5-100%
	All	56.25%	68.75-75%	87.5-100%
Paired elements *i*, *i*+8 *i*-mix (x = 2)	Odd	75%	65.63-75%	90.6-100%
	Even	59.4-75%	65.6-75%	90.6-100%
	All	53.125%	68.75-75%	93.8-100%
Unpaired elem. Even	Single	100%	100%	100%
	Odd	100%	75-93.8%	75-100%
	Even	62.5-100%	75-93.8%	75-100%
Unpaired elem. Odd	Single	100%	100%	100%
	Odd	100%	75-93.8%	75-100%
	Even	62.5-100%	75-93.8%	75-100%
Unpaired elem. Mix	Odd	93.8-100%	75-96.9%	84.4-100%
	Even	62.5-100%	75-96.9%	81.3-100%
i, *i*+16, random for others	Odd	90.6-100%	90.6-100%	90.6-100%
	Even	87.5-100%	90.6-100%	90.6-100%

Table B.5: FNT diffusion for $t = 3$, $F_t = 257$, $N = 64$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $F_t = 0$	Different Modifi. modulo $F_t \neq 0$
Paired elements *i*, *i*+32 *i*-even (x = 1)	Odd	50%	37.5-50%	81.3-100%
	Even	12.5- 50%	25-50%	75-100%
	All	3.125%	43.8-50%	93.8-100%
Paired elements *i*, *i*+32 *i*-odd (x = 1)	Odd	50%	37.5-50%	87.5-100%
	Even	12.5-50%	25-50%	75-100%
	All	3.125%	46.9-50%	93.8-100%
Paired elements *i*, *i*+32 *i*-mix (x = 1)	Odd	46.9-50%	42.2-50%	89.1-100%
	Even	23.4-50%	37.5-50%	90.6-100%
	All	1.563%	48.4-50%	95.3-100%
Paired elements *i*, *i*+16 *i*-even (x = 2)	Odd	71.9-75%	65.6-75%	75-100%
	Even	56.3-75	62.5-75%	87.5-100%
	All	53.125%	68.8-75%	93.8-100%
Paired elements *i*, *i*+16 *i*-odd (x = 2)	Odd	71.9-75%	62.5-75%	75-100%
	Even	56.3-75	62.5-75%	87.5-100%
	All	53.125%	68.8-75%	93.8-100%
Paired elements *i*, *i*+16 *i*-mix (x = 2)	Odd	71.9-75%	68.8-75%	92.2-100%
	Even	60.9-75%	62.5-75%	89.1-100%
	All	51.56%	70.3-75%	95.3-100%
Unpaired elem. Even	Single	100%	100%	100%
	Odd	93.8-100%	75-96.9%	81.3-100%
	Even	62.5-100%	75-96.9%	75-100%
Unpaired elem. Odd	Single	100%	100%	100%
	Odd	93.8-100%	75-96.9%	75-100%
	Even	62.5-100%	75-96.9%	75-100%
Unpaired elem. Mix	Odd	95.3-100%	75-98.4%	87.5-100%
	Even	68.8-100%	75-98.4%	87.5-100%
i, *i*+32, random for others	Odd	90.6-100%	92.2-100%	90.6-100%
	Even	90.6-100%	90.6-100%	90.6-100%

Table B.6: FNT diffusion for $t = 3$, $F_t = 257$, $N = 128$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $F_t = 0$	Different Modifi. modulo $F_t \neq 0$
Paired elements *i*, *i*+64 *i*-even (x = 1)	Odd	46.9-50%	37.5-50%	87.5-100%
	Even	12.5- 50%	25-50%	75-100%
	All	1.563%	46.9-50%	96.9-100%
Paired elements *i*, *i*+64 *i*-odd (x = 1)	Odd	46.9-50%	37.5-50%	87.5-100%
	Even	12.5-50%	25-50%	75-100%
	All	1.563%	46.9-50%	96.9-100%
Paired elements *i*, *i*+64 *i*-mix (x = 1)	Odd	47.7-50%	45.3-50%	93-100%
	Even	24.2-50%	46.1-50%	94.5-100%
	All	0.781%	47.7-50%	96.9-100%
Paired elements *i*, *i*+32 *i*-even (x = 2)	Odd	71.9-75%	67.2-75%	75-100%
	Even	56.3-75%	62.5-75%	87.5-100%
	All	51.56%	71.9-75%	96.9-100%
Paired elements *i*, *i*+32 *i*-odd (x = 2)	Odd	71.9-75%	62.5-75%	75-100%
	Even	56.3-75%	62.5-75%	87.5-100%
	All	51.56%	70.3-75%	96.9-100%
Paired elements *i*, *i*+32 *i*-mix (x = 2)	Odd	71.9-75%	70.3-75%	93-100%
	Even	61.7-75%	70.3-75%	94.5-100%
	All	50.78%	73.4-75%	97.7-100%
Unpaired elem. Even	Single	100%	100%	100%
	Odd	95.3-100%	75-98.4%	87.5-100%
	Even	68.8-100%	75-98.4%	75-100%
Unpaired elem. Odd	Single	100%	100%	100%
	Odd	95.3-100%	75-98.4%	87.5-100%
	Even	68.8-100%	75-98.4%	75-100%
Unpaired elem. Mix	Odd	96.1-100%	75-99.2%	87.5-100%
	Even	74.2-100%	93-99.2%	94.5-100%
i, *i*+64, random for others	Odd	93.8-100%	93.8-100%	93.8-100%
	Even	94.5-100%	93.8-100%	94.5-100%

Table B.6: FNT diffusion for $t = 3$, $F_t = 257$, $N = 256$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $F_t = 0$	Different Modifi. modulo $F_t \neq 0$
Paired elements *i*, *i*+128 *i*-even (x = 1)	Odd	47.7-50%	37.5-50%	87.5-100%
	Even	21.9- 50%	25-50%	75-100%
	All	0.781%	48.4-50%	97.7-100%
Paired elements *i*, *i*+128 *i*-odd (x = 1)	Odd	47.7-50%	37.5-50%	87.5-100%
	Even	21.9-50%	25-50%	75-100%
	All	0.781%	48.4-50%	97.7-100%
Paired elements *i*, *i*+128 *i*-mix (x = 1)	Odd	48-50%	46.5-50%	96.5-100%
	Even	24.6-50%	43-50%	93-100%
	All	0.391%	48.8-50%	98.4-100%
Paired elements *i*, *i*+64 *i*-even (x = 2)	Odd	71.9-75%	62.5-75%	75-100%
	Even	56.3-75%	68.8-75%	87.5-100%
	All	50.78%	71.9-75%	97.7-100%
Paired elements *i*, *i*+64 *i*-odd (x = 2)	Odd	71.9-75%	68.8-75%	75-100%
	Even	56.3-75%	62.5-75%	87.5-100%
	All	50.78%	72.7-75%	96.9-100%
Paired elements *i*, *i*+64 *i*-mix (x = 2)	Odd	72.3-75%	71.5-75%	96.1-100%
	Even	62.1-75%	71.1-75%	96.1-100%
	All	50.39%	73.4-75%	98.4-100%
Unpaired elem. Even	Single	100%	100%	100%
	Odd	96.1-100%	75-99.2%	87.5-100%
	Even	74.2-100%	75-99.2%	87.5-100%
Unpaired elem. Odd	Single	100%	100%	100%
	Odd	96.1-100%	75-99.2%	87.5-100%
	Even	74.2-100%	75-99.2%	75-100%
Unpaired elem. Mix	Odd	96.9-100%	87.5-99.6%	90.6-100%
	Even	92.6-100%	95.7-99.6%	93.8-100%
i, *i*+128, random for others	Odd	96.5-100%	96.5-100%	96.5-100%
	Even	96.5-100%	96.1-100%	96.1-100%

Table B.7: FNT diffusion for $t = 4$, $F_t = 65537$, $N = 4$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $F_t = 0$	Different Modifi. modulo $F_t \neq 0$
Paired elements *i*, *i*+2 *i*-even (x = 1)	Odd	-	-	-
	Even	-	-	-
	All	50%	50%	100%
Paired elements *i*, *i*+2 *i*-odd (x = 1)	Odd	-	-	-
	Even	-	-	-
	All	50%	50%	100%
Paired elements *i*, *i*+2 *i*-mix (x = 1)	Odd	-	-	-
	Even	-	-	-
	All	25%	50%	100%
Paired elements *i*, *i*+1 *i*-even (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	-	-	-
Paired elements *i*, *i*+1 *i*-odd (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	-	-	-
Paired elements *i*, *i*+1 *i*-mix (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	-	-	-
Unpaired elem. Even	Single	100%	100%	100%
	Odd	-	-	-
	Even	-	-	-
Unpaired elem. Odd	Single	100%	100%	100%
	Odd	-	-	-
	Even	-	-	-
Unpaired elem. Mix	Odd	-	-	-
	Even	-	-	-
i, *i*+2, random for others	Odd	100%	100%	100%
	Even	-	-	-

Table B.8: FNT diffusion for $t = 4$, $F_t = 65537$, $N = 8$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $F_t = 0$	Different Modifi. modulo $F_t \neq 0$
Paired elements *i*, *i*+4 *i*-even (x = 1)	Odd	50%	50%	100%
	Even	-	-	-
	All	25%	50%	100%
Paired elements *i*, *i*+4 *i*-odd (x = 1)	Odd	50%	50%	100%
	Even	-	-	-
	All	25%	50%	100%
Paired elements *i*, *i*+4 *i*-mix (x = 1)	Odd	50%	50%	100%
	Even	37.5%	50%	100%
	All	12.5%	50%	87.5-100%
Paired elements *i*, *i*+2 *i*-even (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	75%	75%	100%
Paired elements *i*, *i*+2 *i*-odd (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	75%	75%	100%
Paired elements *i*, *i*+2 *i*-mix (x = 2)	Odd	-	-	-
	Even	-	-	-
	All	62.5%	75%	100%
Unpaired elem. Even	Single	100%	100%	100%
	Odd	-	-	-
	Even	75%	75%	100%
Unpaired elem. Odd	Single	100%	100%	100%
	Odd	-	-	-
	Even	75%	75%	100%
Unpaired elem. Mix	Odd	-	-	-
	Even	87.5%	87.5%	100%
i, *i*+4, random for others	Odd	100%	100%	100%
	Even	87.5-100%	100%	87.5-100%

Table B.9: FNT diffusion for $t = 4$, $F_t = 65537$, $N = 16$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $F_t = 0$	Different Modifi. modulo $F_t \neq 0$
Paired elements *i*, *i*+8 *i*-even (x = 1)	Odd	50%	50%	87.5-100%
	Even	25- 37.5%	50%	100%
	All	12.5%	50%	100%
Paired elements *i*, *i*+8 *i*-odd (x = 1)	Odd	50%	50%	100%
	Even	25-37.5%	50%	100%
	All	12.5%	50%	100%
Paired elements *i*, *i*+8 *i*-mix (x = 1)	Odd	50%	43.8-50%	93.8-100%
	Even	18.8-50%	43.8-50%	93.8-100%
	All	6.25%	50%	100%
Paired elements *i*, *i*+4 *i*-even (x = 2)	Odd	75%	75%	100%
	Even	-	-	-
	All	62.5%	75%	87.5-100%
Paired elements *i*, *i*+4 *i*-odd (x = 2)	Odd	75%	75%	100%
	Even	-	-	-
	All	62.5%	62.5-75%	100%
Paired elements *i*, *i*+4 *i*-mix (x = 2)	Odd	75%	75%	100%
	Even	68.75%	68.8-75%	100%
	All	56.25%	75%	93.8-100%
Unpaired elem. Even	Single	100%	100%	100%
	Odd	100%	87.5%	87.5-100%
	Even	62.5-87.5%	75-87.5%	100%
Unpaired elem. Odd	Single	100%	100%	100%
	Odd	100%	87.5%	87.5-100%
	Even	62.5-87.5%	75-87.5%	100%
Unpaired elem. Mix	Odd	100%	87.5-93.8%	87.5-100%
	Even	62.5-100%	87.5-93.8%	93.8-100%
i, *i*+8, random for others	Odd	93.8-100%	93.8-100%	93.8-100%
	Even	93.8-100%	93.8-100%	93.8-100%

Table B.10: FNT diffusion for $t = 4$, $F_t = 65537$, $N = 32$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $F_t = 0$	Different Modifi. modulo $F_t \neq 0$
Paired elements *i*, *i*+16 *i*-even (x = 1)	Odd	50%	43.8-50%	93.8-100%
	Even	12.5- 50%	43.8-50%	93.8-100%
	All	6.25%	50%	100%
Paired elements *i*, *i*+16 *i*-odd (x = 1)	Odd	50%	43.8-50%	93.8-100%
	Even	12.5-50%	43.8-50%	93.8-100%
	All	6.25%	50%	100%
Paired elements *i*, *i*+16 *i*-mix (x = 1)	Odd	50%	46.9-50%	96.9-100%
	Even	21.9-50%	46.9-50%	96.9-100%
	All	3.125%	50%	100%
Paired elements *i*, *i*+8 *i*-even (x = 2)	Odd	75%	68.8-75%	93.8-100%
	Even	62.5-68.8%	75%	100%
	All	56.25%	75%	100%
Paired elements *i*, *i*+8 *i*-odd (x = 2)	Odd	75%	68.8-75%	93.8-100%
	Even	62.5-68.8%	75%	100%
	All	56.25%	75%	100%
Paired elements *i*, *i*+8 *i*-mix (x = 2)	Odd	75%	71.9-75%	96.9-100%
	Even	59.4-75%	71.9-75%	96.9-100%
	All	53.125%	75%	96.9-100%
Unpaired elem. Even	Single	100%	100%	100%
	Odd	100%	87.5-93.8%	93.8-100%
	Even	62.5-100%	75-93.8%	93.8-100%
Unpaired elem. Odd	Single	100%	100%	100%
	Odd	100%	87.5-93.8%	93.8-100%
	Even	62.5-100%	75-93.8%	93.8-100%
Unpaired elem. Mix	Odd	100%	87.5-96.9%	87.5-100%
	Even	62.5-100%	87.5-96.9%	93.8-100%
i, *i*+16, random for others	Odd	96.9-100%	96.9-100%	96.9-100%
	Even	96.9-100%	96.9-100%	96.9-100%

Table B.11: FNT diffusion for $t = 4$, $F_t = 65537$, $N = 64$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $F_t = 0$	Different Modifi. modulo $F_t \neq 0$
Paired elements *i*, *i*+32 *i*-even (x = 1)	Odd	50%	43.8-50%	87.5-100%
	Even	12.5- 50%	43.8-50%	93.8-100%
	All	3.125%	50%	100%
Paired elements *i*, *i*+32 *i*-odd (x = 1)	Odd	50%	43.8-50%	93.8-100%
	Even	12.5-50%	43.8-50%	93.8-100%
	All	3.125%	50%	100%
Paired elements *i*, *i*+32 *i*-mix (x = 1)	Odd	50%	48.4-50%	96.9-100%
	Even	23.4-50%	48.4-50%	96.9-100%
	All	1.563%	48.4-50%	100%
Paired elements *i*, *i*+16 *i*-even (x = 2)	Odd	75%	71.9-75%	96.9-100%
	Even	56.3-75%	71.9-75%	96.9-100%
	All	53.125%	71.9-75%	100%
Paired elements *i*, *i*+16 *i*-odd (x = 2)	Odd	75%	71.9-75%	96.9-100%
	Even	56.3-75%	71.9-75%	96.9-100%
	All	53.125%	75%	96.9-100%
Paired elements *i*, *i*+16 *i*-mix (x = 2)	Odd	75%	71.9-75%	98.4-100%
	Even	60.9-75%	73.4-75%	98.4-100%
	All	51.56%	73.4-75%	98.4-100%
Unpaired elem. Even	Single	100%	100%	100%
	Odd	100%	87.5-96.9%	87.5-100%
	Even	62.5-100%	75-96.9%	93.8-100%
Unpaired elem. Odd	Single	100%	100%	100%
	Odd	100%	87.5-96.9%	93.8-100%
	Even	62.5-100%	75-96.9%	93.8-100%
Unpaired elem. Mix	Odd	98.4-100%	87.5-98.4%	93.8-100%
	Even	68.8-100%	87.5-98.4%	93.8-100%
i, *i*+32, random for others	Odd	96.9-100%	98.4-100%	96.9-100%
	Even	96.9-100%	96.9-100%	96.9-100%

Table B.12: FNT diffusion for $t = 4$, $F_t = 65537$, $N = 128$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $F_t = 0$	Different Modifi. modulo $F_t \neq 0$
Paired elements *i*, *i*+64 *i*-even (x = 1)	Odd	50%	46.9-50%	93.8-100%
	Even	12.5- 50%	43.8-50%	96.9-100%
	All	1.563%	50%	100%
Paired elements *i*, *i*+64 *i*-odd (x = 1)	Odd	50%	37.5-50%	93.8-100%
	Even	12.5-50%	43.8-50%	93.8-100%
	All	1.563%	50%	100%
Paired elements *i*, *i*+64 *i*-mix (x = 1)	Odd	48.4-50%	49.2-50%	98.4-100%
	Even	24.2-50%	49.2-50%	98.4-100%
	All	0.781%	50%	100%
Paired elements *i*, *i*+32 *i*-even (x = 2)	Odd	75%	71.9-75%	96.9-100%
	Even	56.3-75%	73.4-75%	96.9-100%
	All	51.56%	75%	100%
Paired elements *i*, *i*+32 *i*-odd (x = 2)	Odd	75%	73.4-75%	98.4-100%
	Even	56.3-75%	71.9-75%	96.9-100%
	All	51.56%	75%	100%
Paired elements *i*, *i*+32 *i*-mix (x = 2)	Odd	74.2-75%	73.4-75%	98.4-100%
	Even	61.7-75%	73.4-75%	98.4-100%
	All	50.78%	75%	100%
Unpaired elem. Even	Single	100%	100%	100%
	Odd	98.4-100%	87.5-98.4%	93.8-100%
	Even	68.8-100%	75-98.4%	87.5-100%
Unpaired elem. Odd	Single	100%	100%	100%
	Odd	98.4-100%	87.5-98.4%	96.9-100%
	Even	68.8-100%	75-98.4%	93.8-100%
Unpaired elem. Mix	Odd	98.4-100%	75-99.2%	93.8-100%
	Even	74.2-100%	97.7-99.2%	98.4-100%
i, *i*+64, random for others	Odd	98.4-100%	98.4-100%	98.4-100%
	Even	98.4-100%	98.4-100%	98.4-100%

Table B.13: FNT diffusion for $t = 4$, $F_t = 65537$, $N = 256$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $F_t = 0$	Different Modifi. modulo $F_t \neq 0$
Paired elements *i*, *i*+128 *i*-even (x = 1)	Odd	48.4-50%	46.9-50%	93.8-100%
	Even	21.9-50%	46.9-50%	96.9-100%
	All	0.781%	49.2-50%	100%
Paired elements *i*, *i*+128 *i*-odd (x = 1)	Odd	48.4-50%	48.4-50%	98.4-100%
	Even	21.9-50%	46.9-50%	96.9-100%
	All	0.781%	50%	100%
Paired elements *i*, *i*+128 *i*-mix (x = 1)	Odd	49.2-50%	49.2-50%	99.2-100%
	Even	24.6-50%	49.2-50%	99.2-100%
	All	0.391%	50%	99.6-100%
Paired elements *i*, *i*+64 *i*-even (x = 2)	Odd	74.2-75%	73.4-75%	98.4-100%
	Even	56.3-75%	71.9-75%	96.9-100%
	All	50.78%	75%	100%
Paired elements *i*, *i*+64 *i*-odd (x = 2)	Odd	74.2-75%	73.4-75%	96.9-100%
	Even	56.3-75%	73.4-75%	96.9-100%
	All	50.78%	75%	99.2-100%
Paired elements *i*, *i*+64 *i*-mix (x = 2)	Odd	74.2-75%	74.2-75%	99.2-100%
	Even	62.1-75%	73.8-75%	99.2-100%
	All	50.39%	74.6-75%	99.6-100%
Unpaired elem. Even	Single	100%	100%	100%
	Odd	98.4-100%	87.5-99.2%	96.9-100%
	Even	74.2-100%	75-99.2%	98.4-100%
Unpaired elem. Odd	Single	100%	100%	100%
	Odd	98.4-100%	87.5-99.2%	96.9-100%
	Even	74.2-100%	75-99.2%	98.4-100%
Unpaired elem. Mix	Odd	98.4-100%	93.8-99.6%	98.4-100%
	Even	93.4-100%	98.8-99.6%	99.2-100%
i, *i*+128, random for others	Odd	99.2-100%	99.2-100%	99.2-100%
	Even	99.2-100%	99.2-100%	99.2-100%

Table B.14: FNT diffusion for $t = 4$, $F_t = 65537$, $N = 512$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $F_t = 0$	Different Modifi. modulo $F_t \neq 0$
Paired elements *i*, *i*+256 *i*-even (x = 1)	Odd	49.2-50%	46.9-50%	93.8-100%
	Even	24.6-50%	49.2-50%	99.2-100%
	All	0.391%	50%	100%
Paired elements *i*, *i*+256 *i*-odd (x = 1)	Odd	49.2-50%	48.4-50%	96.9-100%
	Even	24.6-50%	49.2-50%	98.8-100%
	All	0.391%	50%	99.6-100%
Paired elements *i*, *i*+256 *i*-mix (x = 1)	Odd	49.6-50%	49.6-50%	99.6-100%
	Even	24.8-50%	49.6-50%	99.4-100%
	All	0.195%	50%	99.8-100%
Paired elements *i*, *i*+128 *i*-even (x = 2)	Odd	74.2-75%	73.4-75%	98.4-100%
	Even	60.9-75%	73.4-75%	98.4-100%
	All	50.391%	75%	100%
Paired elements *i*, *i*+128 *i*-odd (x = 2)	Odd	74.2-75%	71.9-75%	96.9-100%
	Even	60.9-75%	73.4-75%	98.4-100%
	All	50.391%	75%	99.6-100%
Paired elements *i*, *i*+128 *i*-mix (x = 2)	Odd	74.6-75%	74.6-75%	99.6-100%
	Even	62.3-75%	74.6-75%	99.6-100%
	All	50.195%	74.8-75%	99.8-100%
Unpaired elem. Even	Single	100%	100%	100%
	Odd	98.4-100%	93.8-99.6%	98.4-100%
	Even	75-100%	75-99.6%	99.2-100%
Unpaired elem. Odd	Single	100%	100%	100%
	Odd	98.4-100%	93.8-99.6%	98.4-100%
	Even	75-100%	75-99.6%	99.2-100%
Unpaired elem. Mix	Odd	98.4-100%	96.9-99.8%	98.4-100%
	Even	98.2-100%	99.2-99.8%	99.6-100%
i, *i*+256, random for others	Odd	99.4-100%	99.6-100%	99.4-100%
	Even	99.6-100%	99.4-100%	99.6-100%

Table B.15: FNT diffusion for $t = 4$, $F_t = 65537$, $N = 1024$

Location	No. Modified elements	Same Modification	Different Modifi. modulo $F_t = 0$	Different Modifi. modulo $F_t \neq 0$
Paired elements *i*, *i*+512 *i*-even (x = 1)	Odd	49.2-50%	49.2-50%	98.4-100%
	Even	24.8-50%	49.6-50%	99.6-100%
	All	0.195%	50%	100%
Paired elements *i*, *i*+512 *i*-odd (x = 1)	Odd	49.2-50%	49.2-50%	98.4-100%
	Even	24.8-50%	49.6-50%	99.6-100%
	All	0.195%	50%	100%
Paired elements *i*, *i*+512 *i*-mix (x = 1)	Odd	49.8-50%	49.8-50%	99.8-100%
	Even	24.9-50%	49.8-50%	99.8-100%
	All	0.098%	50%	100%
Paired elements *i*, *i*+256 *i*-even (x = 2)	Odd	74.6-75%	74.2-75%	98.4-100%
	Even	62.3-75%	74.6-75%	99.4-100%
	All	50.195%	74.8-75%	100%
Paired elements *i*, *i*+256 *i*-odd (x = 2)	Odd	74.6-75%	73.4-75%	98.4-100%
	Even	62.3-75%	74.6-75%	99.6-100%
	All	50.195%	75%	100%
Paired elements *i*, *i*+256 *i*-mix (x = 2)	Odd	74.8-75%	74.7-75%	99.8-100%
	Even	62.4-75%	74.8-75%	99.7-100%
	All	50.098%	75%	100%
Unpaired elem. Even	Single	100%	100%	100%
	Odd	98.4-100%	98.4-99.8%	99.2-100%
	Even	75-100%	75-99.8%	99.6-100%
Unpaired elem. Odd	Single	100%	100%	100%
	Odd	98.4-100%	96.9-99.8%	99.2-100%
	Even	75-100%	75-99.8%	99.6-100%
Unpaired elem. Mix	Odd	99.6-100%	99.2-99.9%	99.6-100%
	Even	99.4-100%	99.6-99.9%	99.8-100%
i, *i*+512, random for others	Odd	99.7-100%	99.8-100%	99.8-100%
	Even	99.8-100%	99.8-100%	99.8-100%

Appendix C

Test Vectors Using NMNT

The following are the three vector tests, namely: the known answer test with its two parts, the variable text and the variable key; the multi-block message test; and the Monte Carlo test. These tests are run on the algorithm based on the NMNT in order to verify its correctness.

1. The Known Answer Tests (KAT)

a. Variable Text

Kc	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PT	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	52	104	96	49	102	66	37	120	23	8	4	38	104	91	100	20
PT	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	66	87	59	38	14	103	38	50	105	80	1	79	94	126	97	35
PT	112	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	80	29	122	35	21	1	26	6	83	122	93	35	61	120	120	110
PT	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	44	4	50	114	18	13	120	114	29	122	35	50	107	19	43	103
PT	124	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	117	52	0	31	68	33	94	81	22	99	44	72	77	45	78	72
PT	126	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	106	108	77	46	44	71	73	4	109	77	70	5	25	79	121	85
PT	126	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	15	122	74	91	71	97	62	100	16	117	6	14	122	94	98	70
PT	126	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	106	69	110	4	99	90	110	46	68	56	118	82	79	112	81	111
PT	126	112	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	120	29	113	19	8	109	13	54	22	26	85	72	59	121	68	94
PT	126	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	70	103	28	36	97	93	22	108	74	82	54	14	7	124	93	13
PT	126	124	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	106	119	122	97	91	37	61	115	74	16	20	33	61	115	15	14
PT	126	126	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	25	2	115	45	33	39	75	124	121	47	91	82	105	86	111	102
PT	126	126	64	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	89	47	103	24	28	80	95	80	92	124	18	70	87	76	91	99
PT	126	126	96	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	55	70	43	9	21	99	46	48	58	7	0	33	44	107	118	107
PT	126	126	112	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	39	124	110	60	57	99	57	71	93	51	99	99	120	62	71	82
PT	126	126	120	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	100	38	107	34	93	81	10	93	33	40	18	63	19	13	23	30
PT	126	126	124	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	96	19	87	38	52	46	63	65	112	57	110	88	63	123	33	82
PT	126	126	126	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	27	7	19	14	116	15	49	71	114	111	18	116	5	11	16	97
PT	126	126	126	64	0	0	0	0	0	0	0	0	0	0	0	0
CT	117	10	9	67	24	111	98	117	84	111	28	6	6	94	26	80
PT	126	126	126	96	0	0	0	0	0	0	0	0	0	0	0	0
CT	102	70	47	122	28	104	108	117	116	111	22	98	71	15	9	50
PT	126	126	126	112	0	0	0	0	0	0	0	0	0	0	0	0
CT	58	19	32	59	20	72	106	91	45	25	111	27	75	69	36	80
PT	126	126	126	120	0	0	0	0	0	0	0	0	0	0	0	0
CT	14	64	103	34	39	89	100	55	80	16	104	99	122	16	7	61

PT	126	126	126	124	0	0	0	0	0	0	0	0	0	0	0	0
CT	26	100	124	48	3	75	32	102	75	37	81	72	27	28	89	111
PT	126	126	126	126	0	0	0	0	0	0	0	0	0	0	0	0
CT	103	118	26	125	101	43	117	102	88	71	14	112	39	58	0	21
PT	126	126	126	126	64	0	0	0	0	0	0	0	0	0	0	0
CT	31	70	104	21	48	10	110	87	76	99	35	60	47	26	94	21
PT	126	126	126	126	96	0	0	0	0	0	0	0	0	0	0	0
CT	19	88	6	112	84	68	38	85	88	49	32	65	49	116	110	43
PT	126	126	126	126	112	0	0	0	0	0	0	0	0	0	0	0
CT	86	72	43	95	6	126	104	74	103	120	60	85	84	89	17	122
PT	126	126	126	126	120	0	0	0	0	0	0	0	0	0	0	0
CT	19	11	4	4	31	87	113	36	84	6	82	103	111	36	74	103
PT	126	126	126	126	124	0	0	0	0	0	0	0	0	0	0	0
CT	3	44	117	104	15	69	118	87	82	44	93	80	49	119	59	74
PT	126	126	126	126	126	0	0	0	0	0	0	0	0	0	0	0
CT	41	104	11	52	13	100	122	86	56	114	14	16	108	73	75	15
PT	126	126	126	126	126	64	0	0	0	0	0	0	0	0	0	0
CT	36	106	82	71	20	102	47	107	46	72	97	17	116	115	108	84
PT	126	126	126	126	126	96	0	0	0	0	0	0	0	0	0	0
CT	98	112	37	52	69	114	122	78	7	22	40	96	76	71	54	123
PT	126	126	126	126	126	112	0	0	0	0	0	0	0	0	0	0
CT	23	60	121	36	103	45	81	39	125	45	19	34	80	83	106	24
PT	126	126	126	126	126	120	0	0	0	0	0	0	0	0	0	0
CT	43	116	81	88	64	21	92	43	118	24	19	10	39	88	22	38
PT	126	126	126	126	126	124	0	0	0	0	0	0	0	0	0	0
CT	4	97	60	37	37	88	48	17	56	36	41	96	54	118	73	71
PT	126	126	126	126	126	126	0	0	0	0	0	0	0	0	0	0
CT	118	122	107	20	126	50	73	100	82	103	59	27	98	10	68	123
PT	126	126	126	126	126	126	64	0	0	0	0	0	0	0	0	0
CT	112	72	33	69	50	7	11	102	20	117	63	41	109	6	2	89
PT	126	126	126	126	126	126	96	0	0	0	0	0	0	0	0	0
CT	53	73	123	38	9	36	126	33	104	27	91	83	3	116	52	81
PT	126	126	126	126	126	126	112	0	0	0	0	0	0	0	0	0
CT	18	75	69	50	55	94	2	3	15	84	10	98	114	22	49	28
PT	126	126	126	126	126	126	120	0	0	0	0	0	0	0	0	0
CT	95	27	48	14	81	105	35	15	116	53	105	78	102	57	29	102
PT	126	126	126	126	126	126	124	0	0	0	0	0	0	0	0	0
CT	119	110	26	52	56	118	23	69	89	90	6	50	20	82	77	29
PT	126	126	126	126	126	126	126	0	0	0	0	0	0	0	0	0
CT	53	35	5	122	49	66	10	111	40	74	76	42	35	114	14	13
PT	126	126	126	126	126	126	126	64	0	0	0	0	0	0	0	0
CT	124	87	106	44	41	19	85	89	6	122	91	26	70	25	34	46
PT	126	126	126	126	126	126	126	96	0	0	0	0	0	0	0	0
CT	31	34	104	13	89	120	30	59	38	104	82	83	109	85	69	121
PT	126	126	126	126	126	126	126	112	0	0	0	0	0	0	0	0
CT	10	92	55	88	69	57	67	85	20	42	69	38	68	6	9	46
PT	126	126	126	126	126	126	126	120	0	0	0	0	0	0	0	0
CT	105	118	1	99	82	112	6	20	126	83	33	27	14	56	45	80
PT	126	126	126	126	126	126	126	124	0	0	0	0	0	0	0	0
CT	32	29	44	12	15	53	39	55	23	118	113	82	18	18	48	95

PT	126	126	126	126	126	126	126	126	0	0	0	0	0	0	0	0
CT	55	2	72	9	50	15	3	100	76	8	38	121	20	84	109	81
PT	126	126	126	126	126	126	126	126	64	0	0	0	0	0	0	0
CT	121	73	59	43	81	39	107	12	94	36	79	99	122	17	5	97
PT	126	126	126	126	126	126	126	126	96	0	0	0	0	0	0	0
CT	68	114	96	32	99	33	69	9	117	73	116	48	84	82	27	85
PT	126	126	126	126	126	126	126	126	112	0	0	0	0	0	0	0
CT	76	11	3	23	126	70	62	83	30	102	14	102	84	20	113	75
PT	126	126	126	126	126	126	126	126	120	0	0	0	0	0	0	0
CT	107	100	45	123	119	98	11	107	36	116	47	25	5	19	88	61
PT	126	126	126	126	126	126	126	126	124	0	0	0	0	0	0	0
CT	78	40	104	84	10	20	115	39	89	62	43	18	6	57	65	122
PT	126	126	126	126	126	126	126	126	126	0	0	0	0	0	0	0
CT	101	27	114	78	22	50	82	42	12	42	61	75	43	126	14	113
PT	126	126	126	126	126	126	126	126	126	64	0	0	0	0	0	0
CT	57	107	81	90	10	49	42	65	98	46	29	27	89	113	31	50
PT	126	126	126	126	126	126	126	126	126	96	0	0	0	0	0	0
CT	36	9	66	112	24	109	93	67	113	57	89	109	86	82	70	11
PT	126	126	126	126	126	126	126	126	126	112	0	0	0	0	0	0
CT	14	71	104	35	86	11	93	81	63	72	82	30	111	12	51	112
PT	126	126	126	126	126	126	126	126	126	120	0	0	0	0	0	0
CT	71	113	53	102	68	30	112	73	35	125	63	67	105	14	13	112
PT	126	126	126	126	126	126	126	126	126	124	0	0	0	0	0	0
CT	17	15	44	115	123	41	27	48	2	45	51	49	48	58	77	1
PT	126	126	126	126	126	126	126	126	126	126	0	0	0	0	0	0
CT	57	95	6	92	92	111	51	28	4	60	110	1	112	88	1	46
PT	126	126	126	126	126	126	126	126	126	126	64	0	0	0	0	0
CT	62	7	14	10	35	43	122	85	73	1	44	13	14	120	30	50
PT	126	126	126	126	126	126	126	126	126	126	96	0	0	0	0	0
CT	20	125	116	124	103	103	36	98	27	34	101	11	119	26	121	93
PT	126	126	126	126	126	126	126	126	126	126	112	0	0	0	0	0
CT	60	125	103	85	109	90	117	102	5	26	10	29	58	72	12	120
PT	126	126	126	126	126	126	126	126	126	126	120	0	0	0	0	0
CT	0	52	22	73	112	83	43	106	113	18	3	92	126	77	24	58
PT	126	126	126	126	126	126	126	126	126	126	124	0	0	0	0	0
CT	126	37	68	92	40	26	103	70	29	51	0	68	52	71	104	45
PT	126	126	126	126	126	126	126	126	126	126	126	0	0	0	0	0
CT	32	114	26	117	110	18	35	4	33	109	114	55	125	11	125	98
PT	126	126	126	126	126	126	126	126	126	126	126	64	0	0	0	0
CT	45	2	57	102	55	58	102	97	11	28	122	102	9	16	47	116
PT	126	126	126	126	126	126	126	126	126	126	126	96	0	0	0	0
CT	3	49	40	15	48	33	113	98	91	8	11	56	47	12	109	88
PT	126	126	126	126	126	126	126	126	126	126	126	112	0	0	0	0
CT	54	117	76	109	88	81	111	103	4	55	2	31	122	30	78	24
PT	126	126	126	126	126	126	126	126	126	126	126	120	0	0	0	0
CT	55	13	14	77	82	102	6	89	104	103	26	66	29	19	46	36
PT	126	126	126	126	126	126	126	126	126	126	126	124	0	0	0	0
CT	50	90	65	14	0	78	80	94	17	38	44	103	116	64	86	12
PT	126	126	126	126	126	126	126	126	126	126	126	126	0	0	0	0
CT	62	15	68	79	32	101	126	17	52	33	66	25	101	77	110	0

PT	126	126	126	126	126	126	126	126	126	126	126	126	64	0	0	0
CT	39	110	80	104	110	13	39	124	1	5	45	123	47	66	112	102
PT	126	126	126	126	126	126	126	126	126	126	126	126	96	0	0	0
CT	98	125	43	11	118	102	51	74	19	19	104	89	27	5	24	88
PT	126	126	126	126	126	126	126	126	126	126	126	126	112	0	0	0
CT	1	55	12	97	33	42	106	1	14	56	65	28	19	3	67	17
PT	126	126	126	126	126	126	126	126	126	126	126	126	120	0	0	0
CT	20	118	104	114	123	106	101	30	72	83	86	97	83	112	28	64
PT	126	126	126	126	126	126	126	126	126	126	126	126	124	0	0	0
CT	25	5	36	23	58	82	77	46	19	38	106	76	65	42	3	64
PT	126	126	126	126	126	126	126	126	126	126	126	126	126	0	0	0
CT	27	29	27	99	8	66	125	106	82	18	121	21	103	80	62	125
PT	126	126	126	126	126	126	126	126	126	126	126	126	126	64	0	0
CT	93	89	54	63	43	1	51	15	82	89	10	123	118	123	69	2
PT	126	126	126	126	126	126	126	126	126	126	126	126	126	96	0	0
CT	81	13	6	17	15	62	103	0	106	74	84	67	73	100	47	1
PT	126	126	126	126	126	126	126	126	126	126	126	126	126	112	0	0
CT	110	55	70	73	77	58	55	12	74	77	116	95	53	95	30	54
PT	126	126	126	126	126	126	126	126	126	126	126	126	126	120	0	0
CT	73	79	58	91	24	35	83	58	97	67	110	125	64	109	62	34
PT	126	126	126	126	126	126	126	126	126	126	126	126	126	124	0	0
CT	20	47	35	19	75	112	121	74	109	106	0	44	43	114	38	1
PT	126	126	126	126	126	126	126	126	126	126	126	126	126	126	0	0
CT	61	114	20	12	40	29	84	78	42	76	122	26	69	101	114	41
PT	126	126	126	126	126	126	126	126	126	126	126	126	126	126	64	0
CT	67	122	87	6	65	72	42	108	111	38	67	32	87	1	17	7
PT	126	126	126	126	126	126	126	126	126	126	126	126	126	126	96	0
CT	2	59	14	22	93	64	61	19	81	58	83	43	0	107	101	93
PT	126	126	126	126	126	126	126	126	126	126	126	126	126	126	112	0
CT	69	11	97	89	70	63	38	31	49	28	29	115	45	70	90	6
PT	126	126	126	126	126	126	126	126	126	126	126	126	126	126	120	0
CT	44	93	115	91	61	56	27	120	125	7	107	58	112	113	97	126
PT	126	126	126	126	126	126	126	126	126	126	126	126	126	126	124	0
CT	62	112	68	124	65	30	43	125	27	38	20	101	101	88	17	117
PT	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	0
CT	4	59	3	125	109	24	112	57	7	80	0	120	60	26	40	16
PT	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	64
CT	64	2	21	36	123	48	64	93	124	45	23	116	17	93	4	80
PT	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	96
CT	33	10	47	27	120	26	79	98	100	81	36	118	66	99	2	18
PT	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	112
CT	112	63	96	106	115	35	117	74	91	84	47	120	46	28	120	120
PT	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	120
CT	102	78	63	66	47	7	27	31	62	10	76	6	81	40	107	119
PT	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	124
CT	40	60	115	119	11	109	118	54	4	89	12	73	32	24	41	50
PT	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126
CT	40	86	125	47	110	0	38	27	91	97	32	54	39	75	45	57

b. Variable Key

PT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Kc	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	114	120	13	82	8	8	59	107	19	13	40	18	46	71	2	3
Kc	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	55	49	36	51	75	92	88	89	81	96	78	106	100	36	101	53
Kc	112	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	53	57	38	69	116	66	50	119	94	71	50	47	31	53	62	57
Kc	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	32	31	29	74	20	9	31	121	81	112	21	102	11	76	57	100
Kc	124	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	114	95	101	119	56	113	37	61	92	19	74	40	114	13	96	72
Kc	126	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	96	102	21	20	74	53	91	125	111	115	64	23	126	107	15	68
Kc	126	64	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	92	73	35	70	22	93	58	42	0	98	116	106	78	84	97	106
Kc	126	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	97	13	21	13	0	122	99	21	56	110	102	77	33	25	64	23
Kc	126	112	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	78	21	28	95	26	104	8	96	8	26	46	64	86	52	57	65
Kc	126	120	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	115	118	90	90	42	113	89	31	41	6	26	123	65	59	71	9
Kc	126	124	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	47	84	46	83	19	7	93	52	33	90	16	40	82	92	110	79
Kc	126	126	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	97	78	12	75	2	68	86	59	32	13	105	87	51	117	122	31
Kc	126	126	64	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	99	45	7	102	68	15	100	80	31	52	49	116	89	38	7	96
Kc	126	126	96	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	111	66	54	35	120	107	121	62	117	57	53	111	67	95	11	29
Kc	126	126	112	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	49	59	77	106	56	16	116	50	61	1	108	44	58	13	86	38
Kc	126	126	120	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	40	23	23	100	19	107	113	125	34	63	84	67	84	84	70	101
Kc	126	126	124	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	101	90	57	68	113	11	42	115	105	110	62	9	124	103	62	107
Kc	126	126	126	0	0	0	0	0	0	0	0	0	0	0	0	0
CT	4	64	100	101	38	110	113	36	51	67	120	67	3	22	57	114
Kc	126	126	126	64	0	0	0	0	0	0	0	0	0	0	0	0
CT	83	106	109	48	51	52	63	81	97	22	42	93	109	96	76	78
Kc	126	126	126	96	0	0	0	0	0	0	0	0	0	0	0	0
CT	92	16	61	41	40	46	105	114	95	73	44	104	60	57	31	124
Kc	126	126	126	112	0	0	0	0	0	0	0	0	0	0	0	0
CT	71	96	55	28	94	18	97	101	15	120	120	97	103	78	42	81
Kc	126	126	126	120	0	0	0	0	0	0	0	0	0	0	0	0
CT	59	16	78	85	107	63	59	4	116	102	3	87	49	32	80	0
Kc	126	126	126	124	0	0	0	0	0	0	0	0	0	0	0	0
CT	122	112	126	31	74	58	32	105	39	13	28	14	105	35	122	51

Kc	126	126	126	126	0	0	0	0	0	0	0	0	0	0	0	0
CT	99	70	78	15	112	93	35	50	103	87	10	113	60	52	2	71
Kc	126	126	126	126	64	0	0	0	0	0	0	0	0	0	0	0
CT	57	34	71	15	70	29	45	45	48	125	63	108	50	65	119	74
Kc	126	126	126	126	96	0	0	0	0	0	0	0	0	0	0	0
CT	72	65	74	92	85	38	54	20	105	50	122	126	89	35	7	80
Kc	126	126	126	126	112	0	0	0	0	0	0	0	0	0	0	0
CT	0	33	124	84	89	104	34	4	23	103	67	14	69	86	98	28
Kc	126	126	126	126	120	0	0	0	0	0	0	0	0	0	0	0
CT	69	90	4	55	78	43	123	104	118	113	113	46	115	56	80	42
Kc	126	126	126	126	124	0	0	0	0	0	0	0	0	0	0	0
CT	63	5	115	32	73	3	63	97	119	118	69	34	67	75	41	37
Kc	126	126	126	126	126	0	0	0	0	0	0	0	0	0	0	0
CT	113	49	66	30	83	3	59	77	119	30	53	100	22	123	60	55
Kc	126	126	126	126	126	64	0	0	0	0	0	0	0	0	0	0
CT	106	61	41	11	86	14	121	53	36	113	83	56	87	88	126	119
Kc	126	126	126	126	126	96	0	0	0	0	0	0	0	0	0	0
CT	65	100	22	86	47	19	66	118	115	105	36	2	22	58	121	126
Kc	126	126	126	126	126	112	0	0	0	0	0	0	0	0	0	0
CT	75	105	13	110	46	24	58	99	67	79	99	9	50	115	104	111
Kc	126	126	126	126	126	120	0	0	0	0	0	0	0	0	0	0
CT	30	61	121	123	1	58	36	25	92	111	46	117	15	101	1	108
Kc	126	126	126	126	126	124	0	0	0	0	0	0	0	0	0	0
CT	105	27	76	65	65	27	48	25	89	104	30	40	69	32	31	38
Kc	126	126	126	126	126	126	0	0	0	0	0	0	0	0	0	0
CT	6	100	4	24	94	91	107	3	110	65	83	93	38	0	116	91
Kc	126	126	126	126	126	126	64	0	0	0	0	0	0	0	0	0
CT	36	112	82	1	113	82	99	38	9	44	27	47	8	120	82	104
Kc	126	126	126	126	126	126	96	0	0	0	0	0	0	0	0	0
CT	15	77	92	45	14	69	75	28	46	24	43	45	115	37	56	64
Kc	126	126	126	126	126	126	112	0	0	0	0	0	0	0	0	0
CT	21	93	83	46	2	46	11	41	46	36	78	103	113	69	35	62
Kc	126	126	126	126	126	126	120	0	0	0	0	0	0	0	0	0
CT	10	35	114	65	87	32	80	15	62	45	50	85	62	5	17	52
Kc	126	126	126	126	126	126	124	0	0	0	0	0	0	0	0	0
CT	50	55	18	103	108	83	47	42	96	82	58	124	106	75	69	44
Kc	126	126	126	126	126	126	126	0	0	0	0	0	0	0	0	0
CT	88	58	23	125	35	117	126	75	63	0	65	9	90	105	75	29
Kc	126	126	126	126	126	126	126	64	0	0	0	0	0	0	0	0
CT	26	47	49	59	54	80	9	120	33	83	118	41	46	68	126	13
Kc	126	126	126	126	126	126	126	96	0	0	0	0	0	0	0	0
CT	77	126	5	113	117	112	126	118	52	63	108	94	124	57	61	86
Kc	126	126	126	126	126	126	126	112	0	0	0	0	0	0	0	0
CT	64	84	71	25	45	69	63	69	2	41	34	18	95	46	9	77
Kc	126	126	126	126	126	126	126	120	0	0	0	0	0	0	0	0
CT	109	16	24	84	83	65	62	51	51	81	6	45	72	112	37	119
Kc	126	126	126	126	126	126	126	124	0	0	0	0	0	0	0	0
CT	119	5	46	106	92	75	3	8	76	30	90	101	125	71	117	125
Kc	126	126	126	126	126	126	126	126	0	0	0	0	0	0	0	0
CT	80	103	119	44	105	88	99	7	98	12	45	119	1	50	35	76

Kc	126	126	126	126	126	126	126	126	64	0	0	0	0	0	0	0
CT	44	45	57	101	91	92	70	2	89	116	64	6	81	27	50	98
Kc	126	126	126	126	126	126	126	126	96	0	0	0	0	0	0	0
CT	17	123	7	5	87	14	107	92	121	91	54	12	13	118	113	123
Kc	126	126	126	126	126	126	126	126	112	0	0	0	0	0	0	0
CT	32	67	11	28	11	63	79	121	99	45	15	98	15	88	50	33
Kc	126	126	126	126	126	126	126	126	120	0	0	0	0	0	0	0
CT	42	22	47	13	118	41	63	31	47	29	9	114	86	60	16	53
Kc	126	126	126	126	126	126	126	126	124	0	0	0	0	0	0	0
CT	21	23	32	76	115	98	28	77	1	89	18	63	66	10	76	121
Kc	126	126	126	126	126	126	126	126	126	0	0	0	0	0	0	0
CT	11	122	41	94	1	28	125	93	88	50	99	42	80	99	59	78
Kc	126	126	126	126	126	126	126	126	126	64	0	0	0	0	0	0
CT	42	81	86	49	50	6	12	78	93	0	107	32	51	82	15	13
Kc	126	126	126	126	126	126	126	126	126	96	0	0	0	0	0	0
CT	117	64	87	79	18	50	120	15	15	82	17	89	98	23	41	73
Kc	126	126	126	126	126	126	126	126	126	112	0	0	0	0	0	0
CT	8	54	7	1	125	121	1	30	39	8	49	48	55	28	43	41
Kc	126	126	126	126	126	126	126	126	126	120	0	0	0	0	0	0
CT	20	64	32	107	88	80	71	96	6	103	117	19	89	0	97	63
Kc	126	126	126	126	126	126	126	126	126	124	0	0	0	0	0	0
CT	37	27	123	110	29	82	77	57	17	51	61	30	81	1	10	3
Kc	126	126	126	126	126	126	126	126	126	126	0	0	0	0	0	0
CT	9	119	101	32	17	27	59	100	23	3	22	16	8	44	24	84
Kc	126	126	126	126	126	126	126	126	126	126	64	0	0	0	0	0
CT	43	68	113	2	62	83	44	20	29	54	91	100	122	102	33	21
Kc	126	126	126	126	126	126	126	126	126	126	96	0	0	0	0	0
CT	35	122	50	80	50	80	43	94	3	117	69	52	24	109	1	41
Kc	126	126	126	126	126	126	126	126	126	126	112	0	0	0	0	0
CT	9	35	89	91	16	116	97	15	30	113	61	79	84	16	53	79
Kc	126	126	126	126	126	126	126	126	126	126	120	0	0	0	0	0
CT	34	74	105	95	81	17	84	54	5	12	41	37	118	23	14	31
Kc	126	126	126	126	126	126	126	126	126	126	124	0	0	0	0	0
CT	71	36	0	122	2	17	125	4	82	44	38	102	70	62	110	118
Kc	126	126	126	126	126	126	126	126	126	126	126	0	0	0	0	0
CT	20	3	112	40	93	40	47	81	85	88	103	93	108	52	120	49
Kc	126	126	126	126	126	126	126	126	126	126	126	64	0	0	0	0
CT	39	26	121	60	50	0	13	71	2	22	76	68	4	89	24	72
Kc	126	126	126	126	126	126	126	126	126	126	126	96	0	0	0	0
CT	57	93	91	46	38	81	68	7	24	117	13	71	123	33	29	69
Kc	126	126	126	126	126	126	126	126	126	126	126	112	0	0	0	0
CT	24	8	42	26	85	69	29	99	43	63	66	78	72	63	82	73
Kc	126	126	126	126	126	126	126	126	126	126	126	120	0	0	0	0
CT	15	109	65	16	11	82	24	102	7	78	40	40	66	49	22	29
Kc	126	126	126	126	126	126	126	126	126	126	126	124	0	0	0	0
CT	111	29	94	93	71	47	106	25	17	119	98	29	119	22	90	8
Kc	126	126	126	126	126	126	126	126	126	126	126	126	0	0	0	0
CT	101	65	124	16	67	109	22	102	81	92	66	115	32	31	64	80
Kc	126	126	126	126	126	126	126	126	126	126	126	126	64	0	0	0
CT	112	26	113	115	9	7	47	74	3	24	121	62	86	6	89	102

Kc	126	126	126	126	126	126	126	126	126	126	126	126	96	0	0	0
CT	117	17	19	113	105	102	112	100	17	73	9	79	120	100	29	11
Kc	126	126	126	126	126	126	126	126	126	126	126	126	112	0	0	0
CT	85	14	65	100	60	23	27	113	55	19	33	12	78	20	26	59
Kc	126	126	126	126	126	126	126	126	126	126	126	126	120	0	0	0
CT	30	66	63	65	107	6	10	80	107	40	115	116	18	62	36	94
Kc	126	126	126	126	126	126	126	126	126	126	126	126	124	0	0	0
CT	90	36	18	95	92	52	116	34	125	91	73	16	124	25	106	78
Kc	126	126	126	126	126	126	126	126	126	126	126	126	126	0	0	0
CT	80	18	91	41	52	6	10	67	109	108	65	39	118	103	92	89
Kc	126	126	126	126	126	126	126	126	126	126	126	126	126	64	0	0
CT	122	56	98	71	45	66	4	99	45	36	47	20	122	83	100	67
Kc	126	126	126	126	126	126	126	126	126	126	126	126	126	96	0	0
CT	14	62	56	112	27	7	112	31	17	61	54	109	30	49	40	30
Kc	126	126	126	126	126	126	126	126	126	126	126	126	126	112	0	0
CT	25	76	21	93	51	39	64	27	96	83	102	0	11	125	99	20
Kc	126	126	126	126	126	126	126	126	126	126	126	126	126	120	0	0
CT	29	12	9	116	28	106	60	20	26	51	10	36	121	84	45	24
Kc	126	126	126	126	126	126	126	126	126	126	126	126	126	124	0	0
CT	7	82	32	33	56	120	38	101	47	85	103	15	84	115	35	52
Kc	126	126	126	126	126	126	126	126	126	126	126	126	126	126	0	0
CT	15	117	55	71	68	120	33	5	91	61	65	40	60	95	123	1
Kc	126	126	126	126	126	126	126	126	126	126	126	126	126	126	64	0
CT	17	16	60	21	29	15	59	114	85	50	71	99	93	22	44	121
Kc	126	126	126	126	126	126	126	126	126	126	126	126	126	126	96	0
CT	104	104	123	126	35	63	22	98	126	125	73	107	9	42	76	41
Kc	126	126	126	126	126	126	126	126	126	126	126	126	126	126	112	0
CT	24	65	55	43	53	109	92	125	112	124	111	105	17	69	100	23
Kc	126	126	126	126	126	126	126	126	126	126	126	126	126	126	120	0
CT	98	122	62	107	97	82	87	35	106	0	115	39	38	36	109	19
Kc	126	126	126	126	126	126	126	126	126	126	126	126	126	126	124	0
CT	109	31	39	60	29	111	37	77	87	34	85	32	96	68	83	30
Kc	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	0
CT	50	36	116	55	62	8	0	56	6	119	117	67	61	72	78	65
Kc	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	64
CT	36	28	122	98	30	41	12	119	31	38	30	5	44	2	33	108
Kc	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	96
CT	95	59	81	26	36	77	126	7	18	103	24	90	1	27	10	55
Kc	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	112
CT	64	48	17	47	26	89	65	121	29	33	39	41	28	18	45	56
Kc	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	120
CT	91	28	89	5	96	96	105	69	83	70	5	88	22	3	64	120
Kc	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	124
CT	66	52	30	104	126	44	87	47	43	68	19	18	20	47	121	28
Kc	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126	126
CT	90	24	107	64	105	96	7	95	45	20	67	16	111	35	12	0

2. **The Multi-block Message Test (MMT)**

Kc	45	118	71	35	98	63	121	56	72	51	49	104	37	64	89	93
1 Block																
PT	49	48	49	119	119	67	117	47	89	50	35	123	110	77	47	94
CT	14	123	66	69	99	84	34	54	35	37	17	116	16	85	14	10
2 Blocks																
PT	93	67	72	53	37	87	73	68	83	67	125	64	78	83	77	50
	47	45	73	39	35	48	110	65	62	109	49	50	104	100	97	78
CT	24	6	81	84	41	31	45	31	55	62	1	90	35	35	2	89
	1	26	43	69	76	90	109	62	23	0	98	85	93	25	7	118
3 Blocks																
PT	77	80	80	123	44	74	110	77	120	37	75	78	120	112	117	115
	65	41	98	119	53	67	120	81	42	80	114	54	122	84	68	41
	62	78	71	101	49	78	58	47	83	63	68	39	59	87	86	67
CT	48	50	13	116	87	81	60	49	57	21	35	111	114	19	50	81
	114	15	99	6	101	30	42	92	15	120	29	73	42	96	25	92
	124	109	11	16	85	40	30	59	17	36	39	11	0	73	61	10
4 Blocks																
PT	53	89	94	84	40	66	78	77	91	60	55	46	46	96	120	124
	106	46	58	114	68	78	47	107	87	96	61	64	96	41	57	108
	62	35	61	99	71	68	71	104	37	59	80	107	77	91	97	44
	49	68	80	75	116	95	113	89	101	117	95	66	70	97	81	71
CT	28	43	1	116	110	99	113	60	8	51	15	3	116	13	17	24
	31	2	7	74	86	16	120	111	85	23	45	27	109	101	95	98
	9	105	37	24	91	109	94	58	45	86	31	51	57	10	24	2
	58	11	6	93	73	93	124	64	6	50	115	125	105	92	27	77
5 Blocks																
PT	52	125	96	58	48	126	94	55	86	89	117	64	52	114	115	47
	102	114	121	80	44	87	109	36	44	67	62	36	60	66	83	107
	93	43	90	55	96	54	87	97	97	98	83	84	94	125	116	103
	77	92	101	52	79	71	87	57	67	103	46	94	119	45	103	65
	109	84	59	78	111	82	35	121	36	122	82	65	86	77	83	123
CT	126	35	81	88	74	15	38	54	61	98	104	30	83	13	94	25
	45	37	54	46	60	18	103	54	104	27	42	114	13	66	86	58
	54	23	84	61	29	3	115	111	84	58	40	37	61	106	10	75
	4	11	39	49	44	48	8	93	74	23	97	27	90	35	55	113
	6	29	113	19	35	32	2	9	15	109	35	44	96	25	69	105

6 Blocks

PT	124	99	124	91	59	58	62	68	82	114	100	85	125	62	82	105
	110	78	114	53	105	114	111	59	68	49	81	85	91	94	59	89
	124	122	77	47	55	115	97	41	82	34	47	42	99	70	99	124
	40	115	105	75	46	89	119	97	46	57	97	119	116	37	120	72
	105	106	102	42	57	77	51	40	34	83	75	119	115	89	118	85
	116	76	67	96	81	99	84	88	84	117	113	53	86	94	90	102
CT	47	6	14	46	4	123	20	51	30	88	92	45	23	86	26	10
	61	2	72	35	78	20	116	7	81	19	9	39	126	41	3	56
	63	47	2	15	71	26	57	36	76	124	120	96	65	118	30	71
	19	7	3	41	57	92	112	15	73	82	38	122	122	24	20	122
	10	17	72	103	104	42	49	40	76	4	33	97	50	63	117	63
	9	50	38	57	38	22	8	28	85	39	6	16	6	107	18	40

7 Blocks

PT	44	112	57	88	77	85	87	46	111	49	89	44	72	81	87	60
	120	63	103	125	119	66	92	104	60	77	70	67	109	35	47	33
	79	103	36	54	109	102	101	69	101	97	40	100	116	121	38	86
	109	41	54	69	87	75	116	126	124	92	62	43	94	106	80	60
	55	118	64	120	87	92	70	35	90	34	44	50	66	95	54	107
	34	126	36	102	57	83	78	52	53	95	44	52	38	104	48	70
	94	125	71	106	52	117	92	102	113	124	79	118	92	80	88	80
CT	42	110	95	65	122	110	122	53	95	88	74	59	34	105	42	79
	72	36	79	73	30	90	14	87	106	89	28	67	80	24	119	84
	74	111	90	105	91	71	56	123	34	108	88	60	0	2	54	9
	89	74	20	51	67	21	54	114	81	18	120	20	77	61	64	108
	56	13	75	105	76	121	86	26	87	112	54	17	72	80	35	36
	20	106	117	24	112	5	12	13	75	69	89	126	1	80	30	77
	27	63	105	52	27	81	108	55	46	40	15	61	55	66	75	46

8 Blocks

PT	126	103	91	95	102	94	93	122	52	124	120	52	90	67	68	96
	107	56	89	118	120	37	90	74	99	72	52	33	111	36	71	116
	36	40	99	108	54	100	62	122	46	101	88	116	37	49	115	121
	102	50	81	55	38	78	85	63	115	63	47	91	83	33	105	93
	61	35	84	121	43	51	86	40	80	83	90	33	88	115	114	72
	42	100	46	117	53	102	80	48	72	87	88	41	60	36	87	41
	60	40	71	100	77	62	74	95	96	66	119	118	44	98	117	60
	78	112	106	75	121	33	35	79	94	79	33	50	50	90	73	54
CT	85	69	52	69	108	119	17	50	16	23	83	29	33	44	115	105
	16	49	121	32	93	39	33	81	52	67	59	97	103	111	89	111
	67	48	70	47	14	112	34	51	81	85	92	61	65	126	123	72
	84	117	0	121	117	0	23	30	59	120	77	117	122	32	6	88
	12	52	106	53	68	58	70	104	8	27	13	93	6	96	5	22
	52	75	83	13	125	117	102	27	57	117	58	45	116	33	126	43
	24	62	22	78	3	45	88	9	1	34	73	68	70	120	17	95
	87	40	75	119	60	2	101	66	66	25	98	73	102	34	74	46

9 Blocks

PT	59	64	40	88	70	45	106	106	75	121	48	89	41	105	47	106
	39	55	58	43	109	71	46	51	116	67	58	48	49	99	60	49
	117	117	72	79	83	75	58	52	69	89	106	83	100	55	51	98
	108	99	58	69	60	113	116	118	66	105	95	100	55	71	120	124
	43	44	58	82	62	68	51	55	112	125	121	108	115	68	84	59
	64	59	110	54	73	76	69	86	42	51	108	36	93	112	54	82
	98	77	75	112	60	50	85	82	74	75	115	125	108	82	115	63
	69	109	105	48	109	95	36	74	88	41	89	73	121	106	98	85
	118	105	124	39	82	108	36	105	125	108	119	105	84	74	49	39
CT	56	23	66	12	0	66	106	47	35	51	98	29	92	100	72	92
	61	3	113	12	63	94	10	72	99	80	74	7	65	30	98	96
	86	85	17	123	63	101	101	4	123	36	75	25	70	37	40	17
	35	11	116	60	26	107	101	99	57	18	25	78	3	94	81	59
	26	75	83	98	31	54	82	70	49	59	13	29	21	30	113	98
	4	69	5	62	17	103	104	12	102	94	121	32	25	30	31	76
	57	21	49	117	71	75	25	25	54	74	53	56	126	44	104	105
	83	107	117	109	52	66	29	59	76	84	70	75	55	124	90	105
	88	34	52	77	0	69	104	80	51	72	63	102	50	104	5	106

10 Blocks

PT	106	88	79	96	69	38	69	94	76	49	57	109	51	85	40	75
	95	51	126	109	87	97	94	41	124	111	73	112	99	33	66	100
	117	100	68	126	78	119	38	73	58	61	72	111	100	44	43	105
	74	94	41	77	63	39	41	69	107	102	94	107	49	51	121	98
	65	122	124	42	74	121	96	72	70	76	62	72	89	112	85	35
	43	106	85	100	72	57	53	68	101	48	49	107	98	58	90	87
	54	71	107	55	83	110	36	78	110	125	116	35	43	74	68	75
	47	57	104	110	116	37	71	120	112	37	58	76	118	48	83	41
	52	100	114	34	83	54	95	118	64	61	114	56	86	52	84	82
	90	95	60	60	35	54	52	104	99	57	75	63	37	119	113	64
CT	85	94	104	71	56	26	42	38	55	13	32	91	97	73	33	42
	23	51	85	37	57	46	71	40	64	126	87	16	41	59	82	61
	121	120	18	12	101	87	60	73	47	71	14	80	98	28	48	110
	89	118	35	92	31	87	69	32	92	50	20	71	94	14	69	92
	12	45	38	67	4	61	3	49	52	57	67	80	118	99	117	57
	120	72	64	47	70	30	120	109	86	117	44	101	116	29	75	24
	80	15	57	49	76	32	119	42	43	82	18	120	104	44	41	108
	113	4	33	65	99	6	126	28	118	17	116	93	122	121	37	19
	8	34	13	26	93	78	83	79	119	16	124	84	74	117	76	55
	87	82	58	108	36	101	35	113	96	91	66	28	110	14	31	8

3. The Monte Carlo Test (MCT)

Count = 1																
Kc	78	88	64	55	85	64	68	101	115	119	59	70	65	77	55	79
PT	74	58	110	97	48	69	103	100	95	83	90	40	126	86	37	48
CT	110	21	88	67	42	123	115	37	51	4	122	56	110	93	2	59
Count = 2																
Kc	32	77	24	116	127	59	55	64	64	115	65	126	47	16	53	116
PT	110	21	88	67	42	123	115	37	51	4	122	56	110	93	2	59
CT	12	55	15	73	114	1	72	42	121	65	109	100	38	22	88	7
Count = 3																
Kc	44	122	23	61	13	58	127	106	57	50	44	26	9	6	109	115
PT	12	55	15	73	114	1	72	42	121	65	109	100	38	22	88	7
CT	61	28	61	28	0	10	61	58	83	64	6	98	0	12	29	5
Count = 4																
Kc	17	102	42	33	13	48	66	80	106	114	42	120	9	10	112	118
PT	61	28	61	28	0	10	61	58	83	64	6	98	0	12	29	5
CT	76	109	26	11	77	31	11	63	27	122	0	32	106	126	83	73
Count = 5																
Kc	93	11	48	42	64	47	73	111	113	8	42	88	99	116	35	63
PT	76	109	26	11	77	31	11	63	27	122	0	32	106	126	83	73
CT	75	20	105	18	126	38	106	89	4	92	19	108	72	80	123	53
Count = 6																
Kc	22	31	89	56	62	9	35	54	117	84	57	52	43	36	88	10
PT	75	20	105	18	126	38	106	89	4	92	19	108	72	80	123	53
CT	108	66	16	67	92	36	58	21	61	82	37	61	0	64	76	13
Count = 7																
Kc	122	93	73	123	98	45	25	35	72	6	28	9	43	100	20	7
PT	108	66	16	67	92	36	58	21	61	82	37	61	0	64	76	13
CT	45	52	81	109	35	51	8	100	111	98	61	22	28	101	95	19
Count = 8																
Kc	87	105	24	22	65	30	17	71	39	100	33	31	55	1	75	20
PT	45	52	81	109	35	51	8	100	111	98	61	22	28	101	95	19
CT	7	58	88	8	119	17	87	46	107	120	60	113	81	107	13	70
Count = 9																
Kc	80	83	64	30	54	15	70	105	76	28	29	110	102	106	70	82
PT	7	58	88	8	119	17	87	46	107	120	60	113	81	107	13	70
CT	41	115	62	3	59	7	17	81	43	29	65	14	123	45	18	78
Count = 10																
Kc	121	32	126	29	13	8	87	56	103	1	92	96	29	71	84	28
PT	41	115	62	3	59	7	17	81	43	29	65	14	123	45	18	78
CT	58	31	56	11	65	7	100	66	33	31	126	116	95	63	59	118
Count = 11																
Kc	67	63	70	22	76	15	51	122	70	30	34	20	66	120	111	106
PT	58	31	56	11	65	7	100	66	33	31	126	116	95	63	59	118
CT	29	121	43	77	63	62	83	69	26	11	40	79	126	95	72	13
Count = 12																
Kc	94	70	109	91	115	49	96	63	92	21	10	91	60	39	39	103
PT	29	121	43	77	63	62	83	69	26	11	40	79	126	95	72	13
CT	92	75	53	19	72	7	78	47	67	30	98	121	13	92	83	25

Count = 13																
Kc	2	13	88	72	59	54	46	16	31	11	104	34	49	123	116	126
PT	92	75	53	19	72	7	78	47	67	30	98	121	13	92	83	25
CT	47	115	44	12	119	2	71	2	32	52	49	2	50	112	34	108
Count = 14																
Kc	45	126	116	68	76	52	105	18	63	63	89	32	3	11	86	18
PT	47	115	44	12	119	2	71	2	32	52	49	2	50	112	34	108
CT	0	8	54	82	57	36	29	43	12	77	103	108	113	122	115	14
Count = 15																
Kc	45	118	66	22	117	16	116	57	51	114	62	76	114	113	37	28
PT	0	8	54	82	57	36	29	43	12	77	103	108	113	122	115	14
CT	88	50	101	100	122	82	38	37	17	73	17	13	49	28	12	73
Count = 16																
Kc	117	68	39	114	15	66	82	28	34	59	47	65	67	109	41	85
PT	88	50	101	100	122	82	38	37	17	73	17	13	49	28	12	73
CT	22	107	117	79	86	83	57	71	7	100	120	84	94	94	74	27
Count = 17																
Kc	99	47	82	61	89	17	107	91	37	95	87	21	29	51	99	78
PT	22	107	117	79	86	83	57	71	7	100	120	84	94	94	74	27
CT	75	20	23	52	12	13	104	89	81	97	23	5	80	88	31	5
Count = 18																
Kc	40	59	69	9	85	28	3	2	116	62	64	16	77	107	124	75
PT	75	20	23	52	12	13	104	89	81	97	23	5	80	88	31	5
CT	34	60	101	8	77	7	10	83	14	95	54	62	90	34	21	44
Count = 19																
Kc	10	7	32	1	24	27	9	81	122	97	118	46	23	73	105	103
PT	34	60	101	8	77	7	10	83	14	95	54	62	90	34	21	44
CT	28	19	36	105	65	79	110	64	114	119	47	11	4	57	76	1
Count = 20																
Kc	22	20	4	104	89	84	103	17	8	22	89	37	19	112	37	102
PT	28	19	36	105	65	79	110	64	114	119	47	11	4	57	76	1
CT	79	79	22	75	77	107	32	120	25	78	120	68	35	50	100	12
Count = 21																
Kc	89	91	18	35	20	63	71	105	17	88	33	97	48	66	65	106
PT	79	79	22	75	77	107	32	120	25	78	120	68	35	50	100	12
CT	28	46	78	106	94	98	18	34	116	59	1	80	28	97	86	93
Count = 22																
Kc	69	117	92	73	74	93	85	75	101	99	32	49	44	35	23	55
PT	28	46	78	106	94	98	18	34	116	59	1	80	28	97	86	93
CT	15	49	111	12	17	110	52	126	92	29	105	87	75	8	42	31
Count = 23																
Kc	74	68	51	69	91	51	97	53	57	126	73	102	103	43	61	40
PT	15	49	111	12	17	110	52	126	92	29	105	87	75	8	42	31
CT	88	115	23	87	113	89	87	99	97	63	64	33	60	13	29	109
Count = 24																
Kc	18	55	36	18	42	106	54	86	88	65	9	71	91	38	32	69
PT	88	115	23	87	113	89	87	99	97	63	64	33	60	13	29	109
CT	84	73	107	10	40	69	115	83	74	2	29	80	105	57	35	55

Count = 25																
Kc	70	126	79	24	2	47	69	5	18	67	20	23	50	31	3	114
PT	84	73	107	10	40	69	115	83	74	2	29	80	105	57	35	55
CT	0	82	104	41	115	51	34	21	119	97	89	84	28	121	76	102
Count = 26																
Kc	70	44	39	49	113	28	103	16	101	34	77	67	46	102	79	20
PT	0	82	104	41	115	51	34	21	119	97	89	84	28	121	76	102
CT	56	34	97	86	84	21	110	0	15	98	79	53	125	8	1	6
Count = 27																
Kc	126	14	70	103	37	9	9	16	106	64	2	118	83	110	78	18
PT	56	34	97	86	84	21	110	0	15	98	79	53	125	8	1	6
CT	11	99	99	96	14	111	16	68	20	91	94	116	0	122	99	103
Count = 28																
Kc	117	109	37	7	43	102	25	84	126	27	92	2	83	20	45	117
PT	11	99	99	96	14	111	16	68	20	91	94	116	0	122	99	103
CT	56	27	72	102	118	31	4	15	77	109	2	17	113	10	115	79
Count = 29																
Kc	77	118	109	97	93	121	29	91	51	118	94	19	34	30	94	58
PT	56	27	72	102	118	31	4	15	77	109	2	17	113	10	115	79
CT	40	32	7	37	97	71	117	101	74	17	82	124	1	70	28	17
Count = 30																
Kc	101	86	106	68	60	62	104	62	121	103	12	111	35	88	66	43
PT	40	32	7	37	97	71	117	101	74	17	82	124	1	70	28	17
CT	25	97	74	77	88	120	53	39	19	125	119	20	69	114	42	65
Count = 31																
Kc	124	55	32	9	100	70	93	25	106	26	123	123	102	42	104	106
PT	25	97	74	77	88	120	53	39	19	125	119	20	69	114	42	65
CT	17	44	120	109	94	31	59	15	89	19	112	117	88	86	59	82
Count = 32																
Kc	109	27	88	100	58	89	102	22	51	9	11	14	62	124	83	56
PT	17	44	120	109	94	31	59	15	89	19	112	117	88	86	59	82
CT	77	64	44	87	24	110	15	110	94	22	70	24	15	14	32	25
Count = 33																
Kc	32	91	116	51	34	55	105	120	109	31	77	22	49	114	115	33
PT	77	64	44	87	24	110	15	110	94	22	70	24	15	14	32	25
CT	3	118	79	0	41	120	93	82	29	12	62	54	86	11	17	45
Count = 34																
Kc	35	45	59	51	11	79	52	42	112	19	115	32	103	121	98	12
PT	3	118	79	0	41	120	93	82	29	12	62	54	86	11	17	45
CT	21	61	79	65	36	18	94	105	80	46	34	97	79	124	69	78
Count = 35																
Kc	54	16	116	114	47	93	106	67	32	61	81	65	40	5	39	66
PT	21	61	79	65	36	18	94	105	80	46	34	97	79	124	69	78
CT	122	55	58	106	115	68	12	124	58	69	79	64	55	124	22	45
Count = 36																
Kc	76	39	78	24	92	25	102	63	26	120	30	1	31	121	49	111
PT	122	55	58	106	115	68	12	124	58	69	79	64	55	124	22	45
CT	116	92	20	121	114	14	107	110	76	22	38	51	28	122	110	113

Count = 37																
Kc	56	123	90	97	46	23	13	81	86	110	56	50	3	3	95	30
PT	116	92	20	121	114	14	107	110	76	22	38	51	28	122	110	113
CT	33	5	0	55	23	119	62	98	71	119	91	117	126	67	88	77
Count = 38																
Kc	25	126	90	86	57	96	51	51	17	25	99	71	125	64	7	83
PT	33	5	0	55	23	119	62	98	71	119	91	117	126	67	88	77
CT	50	91	69	56	77	46	3	73	125	46	21	79	121	99	114	122
Count = 39																
Kc	43	37	31	110	116	78	48	122	108	55	118	8	4	35	117	41
PT	50	91	69	56	77	46	3	73	125	46	21	79	121	99	114	122
CT	63	84	52	11	58	105	72	81	76	74	116	13	85	113	116	107
Count = 40																
Kc	20	113	43	101	78	39	120	43	32	125	2	5	81	82	1	66
PT	63	84	52	11	58	105	72	81	76	74	116	13	85	113	116	107
CT	111	103	98	15	111	52	80	88	67	84	34	21	115	21	72	3
Count = 41																
Kc	123	22	73	106	33	19	40	115	99	41	32	16	34	71	73	65
PT	111	103	98	15	111	52	80	88	67	84	34	21	115	21	72	3
CT	31	82	92	24	83	74	37	59	35	90	111	41	10	55	14	107
Count = 42																
Kc	100	68	21	114	114	89	13	72	64	115	79	57	40	112	71	42
PT	31	82	92	24	83	74	37	59	35	90	111	41	10	55	14	107
CT	42	40	70	47	84	86	44	119	107	26	26	113	126	19	115	30
Count = 43																
Kc	78	108	83	93	38	15	33	63	43	105	85	72	86	99	52	52
PT	42	40	70	47	84	86	44	119	107	26	26	113	126	19	115	30
CT	91	19	90	63	106	122	53	77	8	21	124	43	100	95	28	13
Count = 44																
Kc	21	127	9	98	76	117	20	114	35	124	41	99	50	60	40	57
PT	91	19	90	63	106	122	53	77	8	21	124	43	100	95	28	13
CT	113	22	65	115	79	80	114	9	108	41	113	67	72	32	5	113
Count = 45																
Kc	100	105	72	17	3	37	102	123	79	85	88	32	122	28	45	72
PT	113	22	65	115	79	80	114	9	108	41	113	67	72	32	5	113
CT	110	69	16	33	80	34	15	110	81	121	64	37	59	125	77	71
Count = 46																
Kc	10	44	88	48	83	7	105	21	30	44	24	5	65	97	96	15
PT	110	69	16	33	80	34	15	110	81	121	64	37	59	125	77	71
CT	87	114	10	31	8	122	121	26	17	44	36	31	118	25	5	32
Count = 47																
Kc	93	94	82	47	91	125	16	15	15	0	60	26	55	120	101	47
PT	87	114	10	31	8	122	121	26	17	44	36	31	118	25	5	32
CT	0	96	2	5	4	88	85	86	76	33	64	79	49	12	125	94
Count = 48																
Kc	93	62	80	42	95	37	69	89	67	33	124	85	6	116	24	113
PT	0	96	2	5	4	88	85	86	76	33	64	79	49	12	125	94
CT	33	73	113	125	126	44	102	95	54	47	44	71	110	15	89	56

Count = 49																
Kc	124	119	33	87	33	9	35	6	117	14	80	18	104	123	65	73
PT	33	73	113	125	126	44	102	95	54	47	44	71	110	15	89	56
CT	37	95	57	38	98	112	32	111	124	95	33	32	65	113	71	22
Count = 50																
Kc	89	40	24	113	67	121	3	105	9	81	113	50	41	10	6	95
PT	37	95	57	38	98	112	32	111	124	95	33	32	65	113	71	22
CT	93	102	92	89	46	72	41	20	38	13	113	13	58	33	51	16
Count = 51																
Kc	4	78	68	40	109	49	42	125	47	92	0	63	19	43	53	79
PT	93	102	92	89	46	72	41	20	38	13	113	13	58	33	51	16
CT	39	44	66	86	33	38	32	50	44	49	10	75	87	62	110	110
Count = 52																
Kc	35	98	6	126	76	23	10	79	3	109	10	116	68	21	91	33
PT	39	44	66	86	33	38	32	50	44	49	10	75	87	62	110	110
CT	85	72	17	74	64	40	42	104	107	53	36	13	26	21	91	85
Count = 53																
Kc	118	42	23	52	12	63	32	39	104	88	46	121	94	0	0	116
PT	85	72	17	74	64	40	42	104	107	53	36	13	26	21	91	85
CT	41	0	79	51	17	84	25	53	13	17	84	100	64	57	78	72
Count = 54																
Kc	95	42	88	7	29	107	57	18	101	73	122	29	30	57	78	60
PT	41	0	79	51	17	84	25	53	13	17	84	100	64	57	78	72
CT	85	97	99	83	70	76	5	40	44	48	90	19	90	106	24	78
Count = 55																
Kc	10	75	59	84	91	39	60	58	73	121	32	14	68	83	86	114
PT	85	97	99	83	70	76	5	40	44	48	90	19	90	106	24	78
CT	22	119	14	94	68	85	84	22	99	80	123	100	119	9	34	33
Count = 56																
Kc	28	60	53	10	31	114	104	44	42	41	91	106	51	90	116	83
PT	22	119	14	94	68	85	84	22	99	80	123	100	119	9	34	33
CT	125	118	17	87	102	24	102	55	97	37	97	18	29	58	105	16
Count = 57																
Kc	97	74	36	93	121	106	14	27	75	12	58	120	46	96	29	67
PT	125	118	17	87	102	24	102	55	97	37	97	18	29	58	105	16
CT	119	39	55	10	92	45	118	73	85	72	125	60	21	72	23	17
Count = 58																
Kc	22	109	19	87	37	71	120	82	30	68	71	68	59	40	10	82
PT	119	39	55	10	92	45	118	73	85	72	125	60	21	72	23	17
CT	6	107	110	2	123	29	111	19	66	55	17	16	59	50	97	52
Count = 59																
Kc	16	6	125	85	94	90	23	65	92	115	86	84	0	26	107	102
PT	6	107	110	2	123	29	111	19	66	55	17	16	59	50	97	52
CT	109	80	60	117	64	51	11	65	29	7	46	32	99	69	71	111
Count = 60																
Kc	125	86	65	32	30	105	28	0	65	116	120	116	99	95	44	9
PT	109	80	60	117	64	51	11	65	29	7	46	32	99	69	71	111
CT	99	28	126	0	79	118	91	26	64	65	14	46	105	121	92	24

Count = 61																
Kc	30	74	63	32	81	31	71	26	1	53	118	90	10	38	112	17
PT	99	28	126	0	79	118	91	26	64	65	14	46	105	121	92	24
CT	24	113	23	122	99	102	86	91	119	53	117	45	10	13	106	125
Count = 62																
Kc	6	59	40	90	50	121	17	65	118	0	3	119	0	43	26	108
PT	24	113	23	122	99	102	86	91	119	53	117	45	10	13	106	125
CT	57	118	107	86	62	116	96	10	59	57	122	26	25	77	83	57
Count = 63																
Kc	63	77	67	12	12	13	113	75	77	57	121	109	25	102	73	85
PT	57	118	107	86	62	116	96	10	59	57	122	26	25	77	83	57
CT	108	124	29	48	44	74	79	21	109	122	69	109	72	106	54	40
Count = 64																
Kc	83	49	94	60	32	71	62	94	32	67	60	0	81	12	127	125
PT	108	124	29	48	44	74	79	21	109	122	69	109	72	106	54	40
CT	20	82	81	110	72	62	40	15	85	59	98	99	75	20	51	31
Count = 65																
Kc	71	99	15	82	104	121	22	81	117	120	94	99	26	24	76	98
PT	20	82	81	110	72	62	40	15	85	59	98	99	75	20	51	31
CT	23	103	49	96	74	55	108	79	65	101	19	52	49	102	59	19
Count = 66																
Kc	80	4	62	50	34	78	122	30	52	29	77	87	43	126	119	113
PT	23	103	49	96	74	55	108	79	65	101	19	52	49	102	59	19
CT	22	101	40	74	119	116	89	68	15	28	17	88	47	21	105	109
Count = 67																
Kc	70	97	22	120	85	58	35	90	59	1	92	15	4	107	30	28
PT	22	101	40	74	119	116	89	68	15	28	17	88	47	21	105	109
CT	11	70	123	92	32	25	5	79	110	51	124	12	1	85	97	14
Count = 68																
Kc	77	39	109	36	117	35	38	21	85	50	32	3	5	62	127	18
PT	11	70	123	92	32	25	5	79	110	51	124	12	1	85	97	14
CT	89	94	17	56	57	7	77	113	62	41	126	100	95	72	21	26
Count = 69																
Kc	20	121	124	28	76	36	107	100	107	27	94	103	90	118	106	8
PT	89	94	17	56	57	7	77	113	62	41	126	100	95	72	21	26
CT	14	34	47	98	53	64	61	34	75	80	42	7	94	62	53	29
Count = 70																
Kc	26	91	83	126	121	100	86	70	32	75	116	96	4	72	95	21
PT	14	34	47	98	53	64	61	34	75	80	42	7	94	62	53	29
CT	72	67	115	16	120	42	99	41	62	93	47	57	32	103	106	13
Count = 71																
Kc	82	24	32	110	1	78	53	111	30	22	91	89	36	47	53	24
PT	72	67	115	16	120	42	99	41	62	93	47	57	32	103	106	13
CT	18	2	78	28	6	95	111	23	18	75	52	63	94	18	25	30
Count = 72																
Kc	64	26	110	114	7	17	90	120	12	93	111	102	122	61	44	6
PT	18	2	78	28	6	95	111	23	18	75	52	63	94	18	25	30
CT	104	54	86	63	32	41	76	126	46	81	20	119	60	49	55	2

Count = 73																
Kc	40	44	56	77	39	56	22	6	34	12	123	17	70	12	27	4
PT	104	54	86	63	32	41	76	126	46	81	20	119	60	49	55	2
CT	25	73	62	113	0	0	67	117	117	86	98	17	30	94	89	88
Count = 74																
Kc	49	101	6	60	39	56	85	115	87	90	25	0	88	82	66	92
PT	25	73	62	113	0	0	67	117	117	86	98	17	30	94	89	88
CT	99	60	1	30	76	79	84	12	32	35	34	93	91	93	51	35
Count = 75																
Kc	82	89	7	34	107	119	1	127	119	121	59	93	3	15	113	127
PT	99	60	1	30	76	79	84	12	32	35	34	93	91	93	51	35
CT	77	40	29	37	20	6	59	97	53	24	41	78	89	107	80	113
Count = 76																
Kc	31	113	26	7	127	113	58	30	66	97	18	19	90	100	33	14
PT	77	40	29	37	20	6	59	97	53	24	41	78	89	107	80	113
CT	94	64	122	10	37	118	65	100	32	105	22	61	88	98	57	70
Count = 77																
Kc	65	49	96	13	90	7	123	122	98	8	4	46	2	6	24	72
PT	94	64	122	10	37	118	65	100	32	105	22	61	88	98	57	70
CT	18	122	22	19	85	89	37	88	106	111	96	58	110	87	22	62
Count = 78																
Kc	83	75	118	30	15	94	94	34	8	103	100	20	108	81	14	118
PT	18	122	22	19	85	89	37	88	106	111	96	58	110	87	22	62
CT	12	71	105	49	34	124	99	94	21	4	13	85	97	80	101	16
Count = 79																
Kc	95	12	31	47	45	34	61	124	29	99	105	65	13	1	107	102
PT	12	71	105	49	34	124	99	94	21	4	13	85	97	80	101	16
CT	9	124	63	126	56	125	70	95	28	4	88	115	17	8	76	120
Count = 80																
Kc	86	112	32	81	21	95	123	35	1	103	49	50	28	9	39	30
PT	9	124	63	126	56	125	70	95	28	4	88	115	17	8	76	120
CT	43	67	9	26	88	60	112	95	28	73	16	1	2	82	65	42
Count = 81																
Kc	125	51	41	75	77	99	11	124	29	46	33	51	30	91	102	52
PT	43	67	9	26	88	60	112	95	28	73	16	1	2	82	65	42
CT	12	77	16	11	3	56	53	108	2	84	75	4	43	53	6	119
Count = 82																
Kc	113	126	57	64	78	91	62	16	31	122	106	55	53	110	96	67
PT	12	77	16	11	3	56	53	108	2	84	75	4	43	53	6	119
CT	33	118	59	90	106	25	13	52	105	23	25	115	94	28	40	54
Count = 83																
Kc	80	8	2	26	36	66	51	36	118	109	115	68	107	114	72	117
PT	33	118	59	90	106	25	13	52	105	23	25	115	94	28	40	54
CT	113	4	56	59	74	115	5	52	83	81	21	100	39	24	68	20
Count = 84																
Kc	33	12	58	33	110	49	54	16	37	60	102	32	76	106	12	97
PT	113	4	56	59	74	115	5	52	83	81	21	100	39	24	68	20
CT	57	68	96	14	22	27	23	66	63	67	105	34	123	109	74	101

Count = 85																
Kc	24	72	90	47	120	42	33	82	26	127	15	2	55	7	70	4
PT	57	68	96	14	22	27	23	66	63	67	105	34	123	109	74	101
CT	59	79	44	47	71	84	41	60	20	111	32	6	70	56	98	40
Count = 86																
Kc	35	7	118	0	63	126	8	110	14	16	47	4	113	63	36	44
PT	59	79	44	47	71	84	41	60	20	111	32	6	70	56	98	40
CT	39	55	57	84	88	59	101	41	65	5	37	17	2	85	55	67
Count = 87																
Kc	4	48	79	84	103	69	109	71	79	21	10	21	115	106	19	111
PT	39	55	57	84	88	59	101	41	65	5	37	17	2	85	55	67
CT	84	112	40	54	52	66	99	56	113	4	78	15	15	45	100	57
Count = 88																
Kc	80	64	103	98	83	7	14	127	62	17	68	26	124	71	119	86
PT	84	112	40	54	52	66	99	56	113	4	78	15	15	45	100	57
CT	7	73	114	100	66	73	79	125	17	77	78	90	43	70	22	39
Count = 89																
Kc	87	9	21	6	17	78	65	2	47	92	10	64	87	1	97	113
PT	7	73	114	100	66	73	79	125	17	77	78	90	43	70	22	39
CT	9	54	45	30	94	77	46	33	83	42	61	11	32	98	3	25
Count = 90																
Kc	94	63	56	24	79	3	111	35	124	118	55	75	119	99	98	104
PT	9	54	45	30	94	77	46	33	83	42	61	11	32	98	3	25
CT	120	16	53	41	123	54	36	48	33	56	91	8	78	109	19	70
Count = 91																
Kc	38	47	13	49	52	53	75	19	93	78	108	67	57	14	113	46
PT	120	16	53	41	123	54	36	48	33	56	91	8	78	109	19	70
CT	88	57	10	95	121	102	44	74	30	91	65	44	107	72	51	12
Count = 92																
Kc	126	22	7	110	77	83	103	89	67	21	45	111	82	70	66	34
PT	88	57	10	95	121	102	44	74	30	91	65	44	107	72	51	12
CT	71	119	40	34	86	123	9	51	82	12	88	70	38	56	92	35
Count = 93																
Kc	57	97	47	76	27	40	110	106	17	25	117	41	116	126	30	1
PT	71	119	40	34	86	123	9	51	82	12	88	70	38	56	92	35
CT	12	81	4	112	104	37	31	122	81	125	58	71	66	120	104	5
Count = 94																
Kc	53	48	43	60	115	13	113	16	64	100	79	110	54	6	118	4
PT	12	81	4	112	104	37	31	122	81	125	58	71	66	120	104	5
CT	33	85	120	123	120	15	34	78	35	13	33	85	82	23	113	4
Count = 95																
Kc	20	101	83	71	11	2	83	94	99	105	110	59	100	17	7	0
PT	33	85	120	123	120	15	34	78	35	13	33	85	82	23	113	4
CT	6	1	4	21	15	102	120	9	121	126	35	12	52	116	15	94
Count = 96																
Kc	18	100	87	82	4	100	43	87	26	23	77	55	80	101	8	94
PT	6	1	4	21	15	102	120	9	121	126	35	12	52	116	15	94
CT	66	103	33	70	121	41	7	36	103	5	49	53	29	69	105	9

Count = 97																
Kc	80	3	118	20	125	77	44	115	125	18	124	2	77	32	97	87
PT	66	103	33	70	121	41	7	36	103	5	49	53	29	69	105	9
CT	50	5	9	10	124	100	115	62	87	60	34	58	43	102	103	55
Count = 98																
Kc	98	6	127	30	1	41	95	77	42	46	94	56	102	70	6	96
PT	50	5	9	10	124	100	115	62	87	60	34	58	43	102	103	55
CT	79	44	50	63	21	95	39	99	68	0	30	14	79	7	48	81
Count = 99																
Kc	45	42	77	33	20	118	120	46	110	46	64	54	41	65	54	49
PT	79	44	50	63	21	95	39	99	68	0	30	14	79	7	48	81
CT	37	120	33	110	54	35	0	83	25	126	114	32	12	68	61	18
Count = 100																
Kc	8	82	108	79	34	85	120	125	119	80	50	22	37	5	11	35
PT	37	120	33	110	54	35	0	83	25	126	114	32	12	68	61	18
CT	80	46	55	104	108	89	100	73	74	57	72	40	120	3	27	84