Implementation and Analysis of the Generalised New Mersenne Number Transforms for Encryption

Nick Rutter

Newcastle University

Newcastle upon Tyne, UK.

A thesis submitted for the degree of

Doctor of Philosophy

October 2015

Declaration

I declare that this thesis is my own work and it has not been previously submitted, either by me or by anyone else, for a degree or diploma at any educational institute, school or university. To the best of my knowledge, this thesis does not contain any previously published work, except where another person's work used has been cited and included in the list of references.

Nick Rutter

I dedicate this thesis to my family for all of their support and inspiration; in particular Yingpei Li Rutter, Lucy Li Rutter, Jackie Rutter and especially Donald Rutter (1920 — 2010), who inspired me with his invaluable involvement with the Bombe and encouraged me to pursue this area of research.

With all my love.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my first supervisor, Prof. Said Boussakta, for the opportunity to undertake this Ph.D. study and research and along with my second supervisor, Dr. Alex Bystrov with his abundance of patience, enthusiasm and motivation. For identifying key areas requiring attention within this thesis, I would especially like to express my appreciation to Prof. Ahmed Bouridane and Dr. Charalampos Tsimenidis.

I would like to thank all the other members of staff that I have had the pleasure to know and work with, particularly Dr. Martin Johnson, Dr. Patrick Degenaar and Prof. Patrick Briddon. Dr. Johnston has provided me with significant support during my time undertaking my research, including his guidance in pursuing additional areas of research and the opportunity to be involved as co-investigator in two projects: KH148843 and KH135892. Dr. Degenaar gave me the impetus to pursue research in general-purpose graphics processing unit computing from his involvement with his own research, which profoundly sparked my interest in techniques and development in this exciting area of computation. Prof. Briddon shared my enthusiasm for parallel processing and we frequently discussed ideas and techniques.

I would also like to acknowledge all those who that have supported me in any respect during the completion of the project; especially my family for their continual support.

Finally, I would like to thank the European Physical Science Research Council (EPSRC) for financing this research under grant number EP/P50502X/1.

Abstract

Encryption is very much a vast subject covering myriad techniques to conceal and safeguard data and communications. Of the techniques that are available, methodologies that incorporate the number theoretic transforms (NTTs) have gained recognition, specifically the new Mersenne number transform (NMNT). Recently, two new transforms have been introduced that extend the NMNT to a new generalised suite of transforms referred to as the generalised NMNT (GNMNT). These two new transforms are termed the odd NMNT (ONMNT) and the odd-squared NMNT (O^2NMNT).

Being based on the Mersenne numbers, the GNMNTs are extremely versatile with respect to vector lengths. The GNMNTs are also capable of being implemented using fast algorithms, employing multiple and combinational radices over one or more dimensions. Algorithms for both the decimation-in-time (DIT) and -frequency (DIF) methodologies using radix-2, radix-4 and split-radix are presented, including their respective complexity and performance analyses.

Whilst the original NMNT has seen a significant amount of research applied to it with respect to encryption, the ONMNT and O^2NMNT can utilise similar techniques that are proven to show stronger characteristics when measured using established methodologies defining diffusion. Analyses in diffusion using a small but reasonably sized vector-space with the GNMNTs will be exhaustively assessed and a comparison with the Rijndael cipher, the current advanced encryption standard (AES) algorithm, will be presented that will confirm strong diffusion characteristics.

Implementation techniques using general-purpose computing on graphics processing units (GPGPU) have been applied, which are further assessed and discussed. Focus is drawn upon the future of cryptography and in particular cryptology, as a consequence of the emergence and rapid progress of GPGPU and consumer based parallel processing.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

2FA	Two Factor Authentication
3DES	Triple DES
AES	Advanced Encryption Standard
ASCII	American Standard Code for Information Interchange
AVX	Advanced Vector Extensions
CBC	Cipher Block Chaining
CCM	Cipher Block Chaining-Message Authentication Code
CFB	Cipher Feedback
CMEA	Cellular Message Encryption Algorithm
COA	Cipher-Only Attacks
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CTR	Counter
CUDA	Compute Unified Device Architecture
DC	Differential Cryptanalysis
DCT	Discrete Cosine Transform
DES	Data Encryption Standard
DH	Diffie-Hellman Standard

DIF	Decimation-in-Frequency
DIT	Decimation-in-Time
DLP	Discrete Logarithm Problem
DSA	Digital Signature Algorithm
EB	Exabyte (10^{18})
ECB	Electronic Code Book
ECC	Elliptic Curve Cryptography
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
EFF	Electronic Frontier Foundation
EiB	Exbibyte (2^{60})
FIPS	Federal Information Processing Standards
FNT	Fermat Number Transform
GB	Gigabyte (10^9)
GF	Galois Field
GFLOPS	10^9 Floating Point Operations per Second
GiB	Gibibyte (2^{30})
GNMNT	Generalised NMNT
GPGPU	General-Purpose computing on Graphics Processing Units
GPU	Graphics Processing Unit
GSM	Global System for Mobile Communications formally Groupe Spécial Mobile
IoT	Internet of Things
IV	Initialisation Vector

KB	Kilobyte (10^3)
KiB	Kibibyte (2^{10})
KPA	Known Plaintext Attack
LAN	Local Area Network
LC	Linear Cryptanalysis
LFSR	Linear-Feedback Shift Register
MAC	Message Authentication Code
MB	Megabyte (10^6)
MiB	Mebibyte (2^{20})
MIC	Many Integrated Core
MMX	Multimedia Extensions
MNT	Mersenne Number Transform
NIST	National Institute of Standards and Technology
NMNT	New Mersenne Number Transform
NTT	Number Theoretic Transform
O^2NMNT	Odd-Squared NMNT
OFB	Output Feedback
ONMNT	Odd NMNT
OpenCL	Open Compute Language
OTP	One-Time Pad
PAN	Personal Area Network
PB	Petabyte (10^{15})
PHT	Pseudo-Hadamard Transform
PiB	Pebibyte (2^{50})

RC	Row-Column
RSA	Rivest-Shamir-Adleman Cryptosystem
SAC	Strict Avalanche Criterion
SDK	Software Development Kit
SIMD	Single-Instruction Multiple-Data
SPN	Substitution-Permutation Network
TB	Terabyte (10^{12})
TDEA	Triple Data Encryption Algorithm
TDES	Triple DES
TiB	Tebibyte (2^{40})
WAN	Wide Area Network
WHT	Walsh-Hadamard transform
XEX	XOR Encrypt XOR
XOR	Exclusive-OR
XTS	XEX Tweakable Block Cipher with Ciphertext Stealing
ZB	Zettabyte (10^{21})
ZiB	Zebibyte (2^{70})

List of Symbols

α	Generator for β
β	Kernel generator used in deriving GNMNTs
δ	Difference between vectors expressed in bits
Δ	Difference between vectors expressed as a percentage
λ	Normalisation factor, mults to adds
θ	Representation of beta quantities, $\frac{2k+1}{2}$
σ	Standard deviation
τ	Normalised standard deviation expressed in bits
ϕ	Representation of beta quantities, $\frac{2n+1}{2}$
φ	Euler's totient function
$\mathcal{D}$	Arbitrary decryption algorithm
$\mathcal{E}$	Arbitrary encryption algorithm
$\mathcal{G}$	GNMNT transform
$\mathbb{Z}$	Set of integers
$\mathbb{Z}/p\mathbb{Z}$	Field of integers modulo p
$(\mathbb{Z}/p\mathbb{Z})^*$	Multiplicative field of integers modulo p
$\#E$	Cardinality of the set E
c	Cipher vector
$\hat{c}$	Modified cipher vector
$C_{\mathcal{G}}$	Cipher vector in GNMNT domain
E	Set of elliptic curve points
GF	Galois field
Mp	Mersenne prime
k	kth element in the GNMNT domain
K	Encryption key
n	nth element in the number domain

N	Length of a vector
p	generator for Mp, $Mp = 2^p - 1$
r	Recovered plaintext vector
$\hat{r}$	Alternate recovered plaintext vector
$R_{\mathcal{G}}$	Recovered plaintext vector in GNMNT domain
t	Plaintext vector
$\hat{t}$	Alternate plaintext vector
$T_{\mathcal{G}}$	Plaintext vector in GNMNT domain
x	Vector in the number domain
$\overline{x}$	Mean value
X	Vector in the NMNT domain
X_O	Vector in the ONMNT domain
X_{O^2}	Vector in the O^2NMNT domain
$\wedge$	Bitwise AND operation
$\oplus$	Bitwise Exclisive-Or (XOR) operation
$\otimes$	Point-by-point multiplication
$\circledast$	Convolution operator
$\odot$	Convolution operator using the GNMNT
$< . >_{Mp}$	Modulo Mp

Chapter 1

Introduction

1.1 Motivation

The perpetual evolution of technology continues to erode current encryption algorithms owing to performance increases and evolving architectures. Universally, more and more devices are being invented that rely upon the ability to communicate wirelessly [1–4] to name but a few. Whilst there are methodologies in place to help protect these devices from unauthorised access, this does not imply that such methodologies and their underlying algorithms are infallible to breaches of security. In fact, the strongest algorithm will always be a victim of its weakest component, which for proven algorithms is invariably the user's choice of password. Unfortunately, there is no amount of research that is comparable to spending time instructing the user of the importance of deriving and applying passwords that are both robust and varied.

Historically, encryption was generally reserved for communications where messages were to be either sent or broadcast across an unsecured channel. In this instance, a channel can mean any medium, including transportation and radio-wave. With the large amounts of data that companies began to accrue, commercial encryption found its way into storage mediums that are accessible to one or more individuals. Subsequently, this led to consumer-based encryption, allowing the home user to encrypt files for their protection should a device be lost or stolen. With the advance of the mobile phone revolution - where both communications and storage are encrypted - the number of such online devices will soon outnumber the world's population, it can be said now that the requirement

for encryption is ubiquitous.

There are two aspects to protecting one's communication and data: security and cryptography. Whilst security follows a protocol that determines what goes where, when and how, cryptographic algorithms that encrypt and decrypt provide the methods of applying such security. An example of how the security aspect is applied could be as simple as which algorithm to use when sending a key to decrypt the data, or in writing policies that direct users how to derive new keys and how frequently they should be derived (usually always) and, importantly, ensuring that all keys are kept confidential. The encryption algorithm on the other hand, is usually publically known and often heavily analysed, both ensuring and asserting the robustness of such algorithms. The strongest algorithms have a completely transparent architecture that neither hides secrets nor provides 'back doors' that will offer a way to bypass the security and allow decryption without a valid key.

The current de facto algorithm for providing encryption and decryption is the advanced encryption algorithm (AES), which has been in used since its ratification in 2001 [5]. Since this time, despite numerous attacks and analyses directed at this algorithm, it continues to show resilience. The obvious question is why should alternatives be sought if the original AES has not yet been compromised? As previously highlighted, the effects of new technology continue to diminish current techniques, albeit at an exceedingly slow rate. Nevertheless, if a method is discovered to break the algorithms that are currently in use, a requirement to switch to backup algorithms would be precluded if no viable alternatives can be readily accessed. In addition, with the ever increasing size of data that is retained by both industry and the consumer, dependence in stronger methods that can better serve to protect this data may be required in future.

There are many types of encryption algorithms, which are selected depending on the task at hand. These types fall into two main categories: symmetric (private key) and asymmetric (public key). Symmetric based encryption algorithms are usually based upon stream or block ciphers, where speed is a major requirement without sacrificing security. Such algorithms will use the same key to decrypt as that used to encrypt. However, asymmetric based encryption algorithms are usually derived through a mathematical problem where numbers (or keys) are paired using a mathematical technique. While such techniques are extremely resilient to attacks and analysis, they are notoriously slow in contrast to their

symmetric counterparts. They do however offer unique functions such as signing and key exchange operations.

1.2 Aims and Objectives of the Thesis

The aims and objectives of this work are to first analyse work that has previously been undertaken in this subject. This analysis will dictate the direction of the research and help to inform the development of new transforms that can also provide and extend the properties of the original new Mersenne number transform (NMNT) [6, 7], with the benefits of making it more resilient to cryptanalysis. These new transforms are known as the odd-NMNT (ONMNT) and the odd-squared-NMNT (O^2NMNT), which extends the original NMNT into the generalised new Mersenne number transform (GNMNT) [8], the taxonomy of which is shown in Figure 1.1.

These transforms have a long and versatile power of two, which is unfound in other transforms like the Fermat number transform (FNT) [9] and the original Mersenne number transform (MNT) [10]. The latter transforms have short transform lengths that are rigid and inflexible in size and thus have very limited scope.

Like the NMNT, the GNMNTs can be derived using fast algorithms. Although there has been previous development with deriving fast algorithms for the new transforms of the GNMNT - particularly in the one dimensional aspect - supporting literature is extremely limited and consequently incomplete [8, 11]. Current work aims to address these shortcomings by deriving fast algorithms that include the radix-2, radix-4 and split-radix using both decimation-in-time (DIT) and -frequency (DIF) methodologies. Complexity analysis will be provided and, together with performance metrics, these algorithms will be further analysed.

In addition to the fast algorithms for applications using single-dimensional implementations, row-column implementations have been derived to provide two-dimensional implementations through inclusion of the separable algorithm. These algorithms are validated using two dimensional convolution techniques followed by demonstrations using two-dimensional encryption examples.

A major technique used in cryptanalysis is assessing the diffusion properties known as the strict avalanche criterion (SAC) [12–14]. These properties will

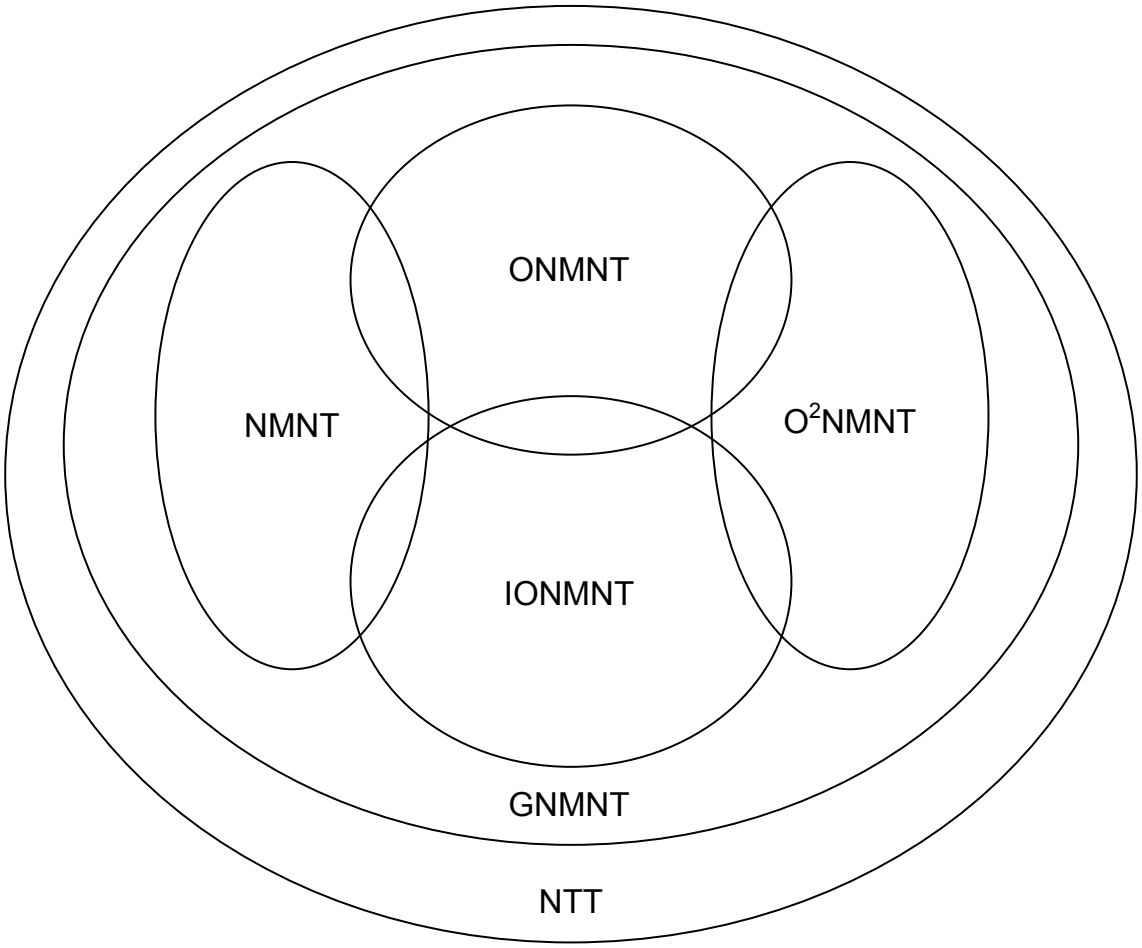


Figure 1.1: Taxonomy of the GNMNT

be performed in the first instance by doing an exhaustive test, simulating every possible vector input of a pre-determined size and applying the series of tests on each vector. Previously, this would have taken approximately 8.5 years using C and a single processor thread. However, with the advent of parallel processing techniques that are available using consumer grade graphics cards [15, 16], such tests become a very much more realistic goal in terms of a modest two days.

1.3 Contributions

There are four main areas that have been addressed within this thesis, which are:

- The new transforms collectively known as the GNMNT, including fast algorithms applied to the one-dimensional case
- Derivation of the ONMNT for two-dimensional applications
- Assessment of the diffusion as an exhaustive simulation and against a series

of vectors using the relevant aspects of the advanced encryption algorithm (AES) for comparison

- Implementations of the GNMNT using graphics processing units (GPUs) for parallel processing techniques.

1.4 Publications Arising From This Research

1. Boussakta, S.; Hamood, M.T.; **Rutter, N.**, "Generalized New Mersenne Number Transforms," *Signal Processing, IEEE Transactions on*, vol.60, no.5, pp.2640-2647, May 2012.

2. **Rutter, N.**; Boussakta, S.; Bystrov, A., "Assessment of the One-Dimensional Generalized New Mersenne Number Transform for Security Systems," *Vehicular Technology Conference (VTC Spring)*, 2013 IEEE 77th, pp.1-5, 2-5 June 2013.

1.5 Thesis Outline

This thesis consists of seven chapters, each of which is described in this section. Chapter 2 will introduce the background of cryptography. During the course of this chapter, different types of methodologies will be covered along with their appropriate applications. This chapter will briefly touch upon the NMNT, which has previously been researched for use in encryption techniques.

After this brief introduction, Chapter 3 will formally introduce the NMNT and the new GNMNT transforms that were developed in [8]. During this chapter, notes of particular interest will be presented that will be relevant in proceeding chapters.

Chapter 4 will expand upon previous work that was undertaken during the development of the fast algorithms by introducing new techniques that build from radix-2 including radix-4 and split-radix DIT and DIF derivations. This chapter will also assess the complexities and performances of each of these new derivations and compare the findings with previous methodologies.

A natural progression is to expand the fast algorithms to row-column derivations, where work that was presented in Chapter 4 will be applied and converted to

two-dimensions in using newly adapted separable algorithms for the GNMNT in Chapter 5.

Chapter 6 introduces cryptanalysis techniques and introduce the SAC. Using the SAC, the new transforms will be assessed for their suitability for inclusion within encryption algorithms. Initially, an assessment over a small but practical and exhaustive range of vectors will be undertaken. Upon comparing the results, a stripped down version of the AES will then be tested alongside the GNMNT using identical vectors. This implementation of the AES does not contain any password or round key functions so as to attain a true raw diffusion measurement.

Finally, conclusions and suggestions for further work will be presented in Chapter 7.

Chapter 2

Background

2.1 Introduction

This chapter will introduce a small cross-section of applications that rely upon encryption, during which some examples will be provided including some best practices. A breakdown of the composition of encryption system will presented, including key, cipher and mode types. The discussion includes the process of determining the AES and a brief breakdown of each cipher. This chapter will also introduce new and emerging technologies that could undermine current and future systems prior to presenting a summary.

2.2 Encryption Applications

Security is ubiquitous in todays emerging technology. Encryption algorithms can be found in many devices that are taken for granted on a day to day basis, frequently and most likely without the user's knowledge that it has ever been implemented in the first place. As an example, use of mobile phones now extends to approximately 7 billion reported users and 3.5 billion reported unique subscribers [17], representing approximately 50% of the global population [18]. A vast majority of these users will not even consider the concept of encryption, never mind whether or not their phone calls are protected or not. Additionally, we can also find encryption in wireless sensors and electronic wearable devices such as the smart watches and heart-rate monitors for example [2, 19]; such devices are a fast-growing trend and mobile smart phones are regularly used to process the signals. However, there

are also reports that these sensor signals are currently not being encrypted, which could potentially be exposing both private data and processing devices that may otherwise be unprotected and vulnerable to attacks [20].

In addition to the requirements of security within a communicational environment, there is also a requirement for the encryption of stored data, whether it is for personal or commercial usage. Within a commercial environment, the protection of documents containing strategies and financial accounts that are pertinent to the livelihood of the business is not only desirable, but may well be a requirement by law, depending on the actual type of data and particularly if it relates to client information [21–23]. This need will become more prevalent with the evolution of cloud computing, where consumer computing requirements will be virtualised and processed across the Internet, including cloud storage [24–28].

However, it is still important that local storage is adequately protected, whether it is encrypted databases, implementing hard drives with built-in encryption or even backup solutions. The importance of such is exemplified with the data storage trends that are now referred to as big data [29], where electronic data is being generated and stored at an exponential rate [30]. This trend has opened new markets and research where new types of technologies are being researched and developed to accommodate our ever expanding storage requirements. According to [31], this is growing at a rate of approximately 2.5 EB / day and where [30] believes that the digital universe will reach a size of 40 ZB by the year 2020. As we are connecting more devices to the internet [32] and using increasingly faster link speeds, the amount of data that we are moving is also increasing [33]. These trends can and will no doubt impact on the way we manage data, which should obviously be protected using encryption.

The most important aspects of encryption are to use using the appropriate scheme and degree of security for the job and to always use different keys. For example, long-term security requires strong encryption whereas short-term security should use a less secure system. The reason for this is through building cribs or clues from intercepted messages where known cipher texts had their corresponding plaintexts known. This opens up the cipher to a known plaintext attack (KPA). An example of this was when the same extremely strong encryption that was used for priority communications was also used for encrypting weather and status reports during World War II where messages were often duplicated using different daily

keys [34]. Likewise, having an important understanding on the application of keys is paramount, namely that keys should be both strong and unique, consist of many unique and special characters wherever possible and be unique to each site, message or application. Thus, should a key be compromised, damage would be limited to the site, message or application that is attributed to that key. This practice is further exemplified in [35], which goes on to emphasise that this is generally the weakest part of an encryption scheme and that users should be trained accordingly about the importance of key management.

2.3 Encryption Types

Encryption systems are generally categorised as falling into two distinct fields; symmetric (private key) and asymmetric (public key). The symmetric system uses the same key to decrypt that was used to encrypt and is always kept private, hence the reason it is also called private key encryption. Conversely, systems that are derived using asymmetric keys use two keys; a private key and a public key, which give name to these types of systems as the public key encryption schemes. Systems that were first derived used symmetric keys and so will be covered first.

2.3.1 Symmetric Keys

Systems using symmetric keys date all the way back to ancient times, where it has been shown that Julius Caesar used encryption for communications. This is known today as the Caesar Cipher [36]. It is a letter substation system where letters of the message were substituted by letters a number of places either forwards or backwards: modulo 26. This offset was the key and literature records that Julius Caesar's preference was to use a key that was three letters to the right. All that was then required to decrypt the message would be to reverse the operation by substituting the letters in the opposite direction, which in Caesars case would be three letters to the left. An example of this cipher is shown in Table 2.1.

Table 2.1: Letter Indexing for Caesar Cipher with Key = 3

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
In	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Out	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*	*A*	*B*	*C*

A major problem with this scheme, other than the existence of such a weak key, was that cryptanalysis would be easy by determining letter substitutions through letter frequencies. This was overcome with the Vigenère Cipher, where a repeating passphrase was used instead, the cipher was later broken by Babbage [37]. Whilst it was a much stronger system in comparison to the Caesar Cipher, it will nevertheless still be regarded as a toy system by todays standards. Examples of this cipher system in operation can be seen in Tables 2.2 and 2.3. Further dissemination of this encryption type will be provided further in block ciphers in Section 2.4.2.

2.3.2 Asymmetric Keys

The asymmetric encryption scheme uses two keys; a public key to encrypt, hence the moniker public key encryption, and the corresponding private key to decrypt. It was invented in 1978 by [38] where an algorithm called RSA was developed and published, based on the initial of each of the inventors surnames. A similar system was invented in 1975 but because it was developed under classified circumstances, there was no mention of it until it was declassified in 1997 [39]. Although it is not as fast as symmetric encryption, owing to the complexity of the calculations, for applications where a reasonably small encryption message, such as a key for a symmetric system that is required to be communicated, it is an ideal solution for communicating over an exposed public channel. Similar schemes have been developed that significantly increase security, such as the elliptic curve encryption system as shown in Table 2.4 from [40]. However, one of the biggest issues with this system that can also be obtained from [40] is that it is an inaccessible system to implement given that it is protected by over 130 patents.

Table 2.2: Encryption Using Vigenère Cipher

t	h	i	s		i	s		t	o	p		s	e	c	r	e	t	
P	A	S	S		W	O		R	D	P		A	S	S	W	O	R	+
i	*h*	*a*	*k*		*e*	*g*		*k*	*r*	*e*		*s*	*w*	*u*	*n*	*s*	*k*	

Table 2.3: Decryption Using Vigenère Cipher

i	*h*	*a*	*k*		*e*	*g*		*k*	*r*	*e*		*s*	*w*	*u*	*n*	*s*	*k*	
P	A	S	S		W	O		R	D	P		A	S	S	W	O	R	−
t	h	i	s		i	s		t	o	p		s	e	c	r	e	t	

2.3.2.1 Factorisation Problem

The factorisation problem is based around a large modulus whose generation is based upon two prime numbers. During the operation of constructing the modulus, two keys are created that are linked to the modulus within a finite field. This is best described by using the RSA algorithm, which demonstrates the fundamentals behind this type of encryption system. It is based on the factorisation problem where two primes are selected, p and q; the product of these primes is calculated such that

$$n = pq. \tag{2.1}$$

The modulus for all future computations within the encryption algorithm will be n. The totient is then calculated so that

$$\varphi = (p-1)(q-1). \tag{2.2}$$

A public key e is selected such that $e < n$ and the greatest common divisor of e and φ is equal to 1 such that

$$\gcd(e, \varphi) = 1. \tag{2.3}$$

The private key d is calculated as the inverse of the public key, e^{-1} modulo the totient φ as

$$d = e^{-1} \bmod \varphi. \tag{2.4}$$

Encryption is performed using

$$c = m^e \bmod n \tag{2.5}$$

and similarly, decryption is performed by

$$m = c^d \bmod n. \tag{2.6}$$

Table 2.4: NIST Recommended Key Sizes

Key Size (bits)		
Symmetric	RSA and Diffie-Hellman	Elliptic Curve
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

This algorithm is relatively simple to implement, with the main requirement being the ability to perform calculations on very large numbers based upon the size of the modulus being used. It is a fairly fast algorithm when used with small key sizes, although the security of RSA is weak in contrast to symmetric based encryption algorithms due to the size of the key required to offer comparable strength in security. For example, a symmetric key size of 256 bits would be comparable to an RSA key size of 15360 bits as previously shown from [40] in Table 2.4.

2.3.2.2 Discrete Logarithm Problem

The discrete logarithm problem (DLP) is another asymmetric method [41]. However, instead of building problems based upon factorisation it is instead based around logarithms. To demonstrate, the DLP is based upon solving the problem

$$\alpha^x \equiv \beta \bmod p \tag{2.7}$$

for x such that

$$x = \log_\alpha \beta \bmod p \tag{2.8}$$

having previously disclosed α, β and p. The problem of solving x is deemed to be very difficult. When this problem is applied over a cyclic finite group $\mathbb{Z}/p\mathbb{Z}$ over modulo p, which has been created using a generator, then there exists a multiplicative inverse that is difficult to derive as each element is the result of raising the generator to a power up to $p-2$ modulo p. For example, if $\alpha = 2$, $x = 7$ and $p = 17$ then $2^7 = 128$, which results in $2^7 \equiv 17$. The multiplicative field $(\mathbb{Z}/p\mathbb{Z})^*$ is defined as

$$(\mathbb{Z}/p\mathbb{Z})^* = \{1, 2, \ldots, p-1\}. \tag{2.9}$$

This field is computed from α^0 to α^{p-2} providing $p-1$ elements using generator α as

$$(\mathbb{Z}/p\mathbb{Z})^* = \left\{\alpha^0, \alpha^1, \ldots, \alpha^{p-2}\right\}. \tag{2.10}$$

The field is calculated up to $p-2$ because

$$\alpha^{p-1} \equiv 1 \bmod p. \tag{2.11}$$

Should, for example, $p = 17$ with $(\mathbb{Z}/p\mathbb{Z})^* = \{1, 2, \ldots, p-1\}$ and generator $\alpha = 7$ is used then

$$(\mathbb{Z}/p\mathbb{Z})^* = \left\{7^0, 7^1, 7^2, 7^3, 7^4, 7^5, 7^6, 7^7, 7^8, 7^9, 7^{10}, 7^{11}, 7^{12}, 7^{13}, 7^{14}, 7^{15}\right\}. \tag{2.12}$$

When this field is calculated over mod p, it becomes

$$(\mathbb{Z}/p\mathbb{Z})^* = \{1, 7, 15, 3, 4, 11, 9, 12, 16, 10, 2, 14, 13, 6, 8, 5\}. \tag{2.13}$$

When $x = 11$ is applied to (2.7) then $7^{11} \equiv 14 \bmod 17$. The problem then lies within solving x with knowledge of $alpha$, β and p such that

$$x = \log_\alpha \bmod p. \tag{2.14}$$

However, as p increases it becomes extremely difficult to derive x from α and β.

2.3.2.3 Diffie-Hellman Key Exchange

One of the most common applications for asymmetric cryptography is the Diffie-Hellman (DH) key exchange protocol that was first published by [42]. Again like RSA [43], a similar system was developed previously but as it was developed under classified circumstances, it was unable to be published at that time [39]. The DH scheme is usually used to exchange symmetric keys, which are generally preferred over asymmetric keys as they are typically faster than asymmetric systems. Based on the DLP, it provides a method for two parties to communicate with each other without exposing decryption keys. Both parties will decide upon a multiplicative field over a prime, $(\mathbb{Z}/p\mathbb{Z})^*$ and a generator, α. Each party will derive an ephemeral or temporary key in the range $1 < k < p-1$ and apply it to the generator as

$$K_A = \alpha^{k_a} \bmod p \text{ for } 0 < k_a < p-1 \tag{2.15}$$

and

$$K_B = \alpha^{k_b} \bmod p \text{ for } 0 < k_b < p-1. \tag{2.16}$$

Each party sends each other K_A and K_B and by applying their private keys they can both derive the same shared session key

$$(K_A)^{k_a} = S_k = (K_B)^{k_b}. \tag{2.17}$$

This works because

$$\left(\alpha^{k_b}\right)^{k_a} = \left(\alpha^{k_a}\right)^{k_b}. \tag{2.18}$$

This method of key exchange is very secure and underpins how private communications are established across the Internet, albeit in RSA form where it has been adapted to the factorisation problem.

2.3.2.4 ElGamal Encryption

In 1985, Taher ElGamal developed an encryption scheme that was based on the DLP [44]. Being based on the DLP, it also shares similarities with the DH key exchange system [42], particularly the sharing of the 'secret'. Both Alice and Bob agree upon a large prime p and generator α over the multiplicative group $(\mathbb{Z}/p\mathbb{Z})^*$. Alice then selects a random integer k_a such that $1 < k_a < p-1$ and sends to Bob the derivation of

$$K_A = \alpha^{k_a} \bmod p \tag{2.19}$$

Bob receives the value derived in (2.19) and himself derives a random key k_b from $1 < k_b < p-1$. Representing the message m in the range $0, 1, \ldots, p-1$ he then derives

$$\gamma = \alpha^{k_b} \bmod p \tag{2.20}$$

and

$$\delta = mK_A^{k_b} \bmod p \tag{2.21}$$

sending $c = (\gamma, \delta)$ to Alice. Upon receiving c, Alice uses her private key a to derive

$$\gamma^{p-1-k_a} \bmod p \tag{2.22}$$

which because

$$\gamma^{p-1-k_a} = \gamma^{-k_a} \tag{2.23}$$

this operation is equivalent to deriving $\alpha^{-k_a k_b}$. Alice can now decode the message by

$$m = \gamma^{-a}\delta \bmod p. \tag{2.24}$$

2.3.2.5 Elliptic Curve Discrete Logarithm Problem

Elliptic curve cryptography (ECC) was developed in 1985 independently by [45] and [46]. Schemes using elliptic curves are based around the elliptic curve discrete

logarithm problem (ECLDP). Similar to the LDP, the ECC uses a logarithm problem. However, it accomplishes this by using points that exist along an elliptic curve defined as

$$E: \ y^2 \equiv x^3 + ax + b \bmod p \tag{2.25}$$

where $a, b \in \mathbb{Z}/p\mathbb{Z}$ are user parameters used to define the curve over a field defined by prime $\mathbb{Z}/p\mathbb{Z}$, $p > 3$. This field contains coordinate pairs $(x, y) \in \mathbb{Z}/p\mathbb{Z}$ and an imaginary point $\mathcal{O}$, which is used to denote the point at infinity. The validity of the curve is dependent on satisfying

$$4a^3 + 27b^2 \not\equiv 0 \bmod p. \tag{2.26}$$

All the calculations for determining the coordinates of the points are carried out from G, which is provided to define G_x and G_y . This is the generator point and is used to define all of the other points. The problem is then defined as

$$Q = dG \bmod p \tag{2.27}$$

where d is selected randomly and is in the range $2 \leq d < p - 2$. Equation (2.27) then produces Q, the public key. Determining d with only knowledge of Q is therefore intractable if the field is large enough. It should be noted that technically points are not multiplied directly but instead repetitively added, for which there are two distinct operations: adding and doubling. However, these are not straightforward operations, as may be implied, as the line must first be characterised by its gradient. This can accomplished by first deriving s as

$$s = \frac{y_2 - y_1}{x_2 - x_1} \bmod p \tag{2.28}$$

if the point is to be added with another point or

$$s = \frac{3x_1^2 + a}{2y_1} \bmod p \tag{2.29}$$

if the point is to be doubled, i.e. $P + P$ or $2P$. These operations can be better visualised in Figure 2.2. Addition is shown by adding points P and Q and drawing a line that intersects these points and continues until it hits the next part of the curve. At the point where the line meets the next part of the curve, a vertical line is drawn through the $y = 0$ axis until it reaches the curve again, denoting the actual

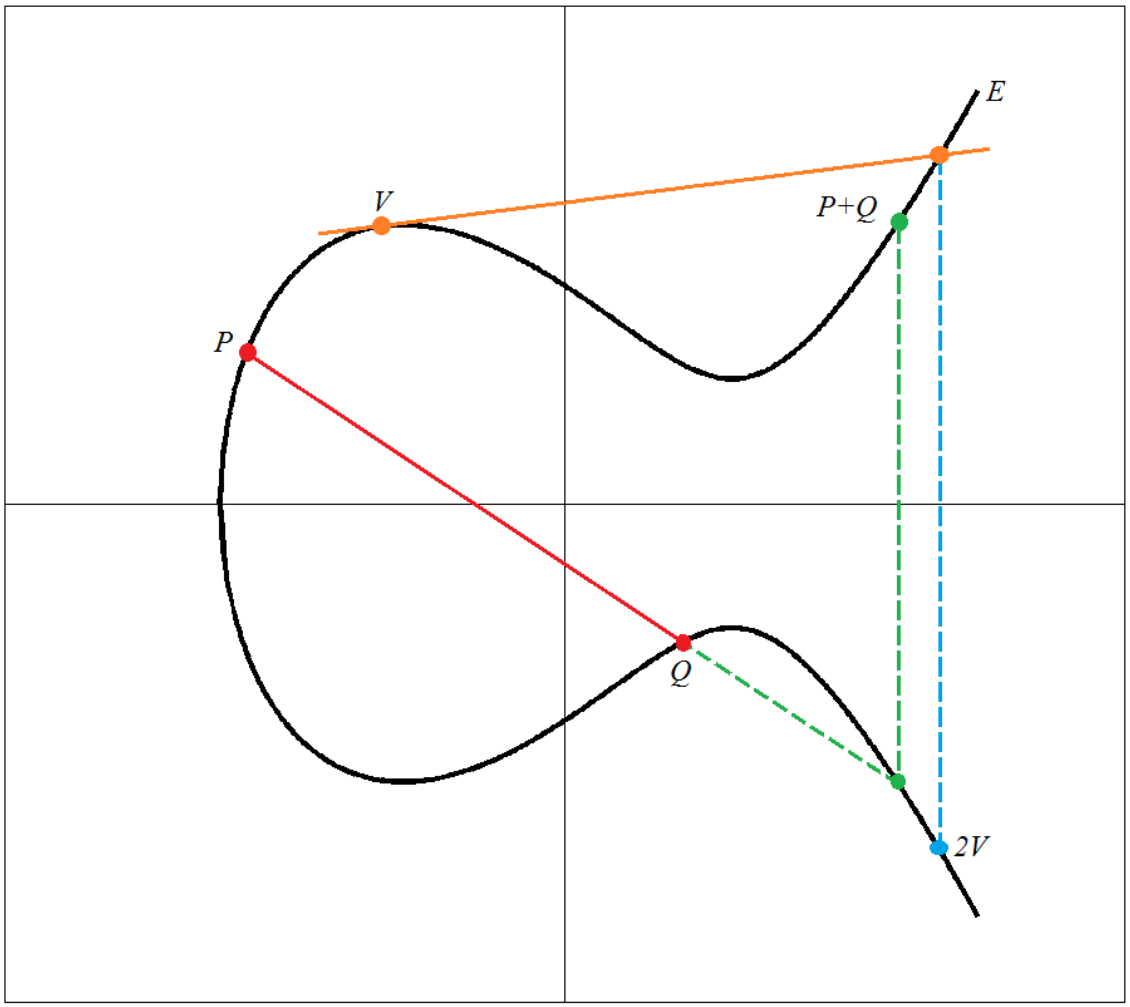


Figure 2.1: Addition and Doubling Operations in the Elliptic Curve

position of the addition. Point doubling is shown by drawing a line at a tangent to the point V and continuing until it reaches the curve once more, characterised by point E in Figure 2.2. Like the point addition, a vertical line is drawn through the $y = 0$ axis until it reaches the curve once more, indicating the point $2V$.

An important aspect of the division operations in (2.28) and (2.30) is that because the operations are performed within a field, the division should be performed by multiplying with the multiplicative inverse instead. This can easily be undertaken using the extended Euclidean algorithm or

$$x^{-1} = x^{p-2} \bmod p. \tag{2.30}$$

The line characterised by s connects the two points that are to be added and where the line intersects the curve, is the point mirrored across the x-axis. While point doubling only uses a single point, the line is derived by forming a tangent to the curve at the point, again noting where the line intersects the curve. These operations are better visualised in Figure 2.1. The new point can then be then be

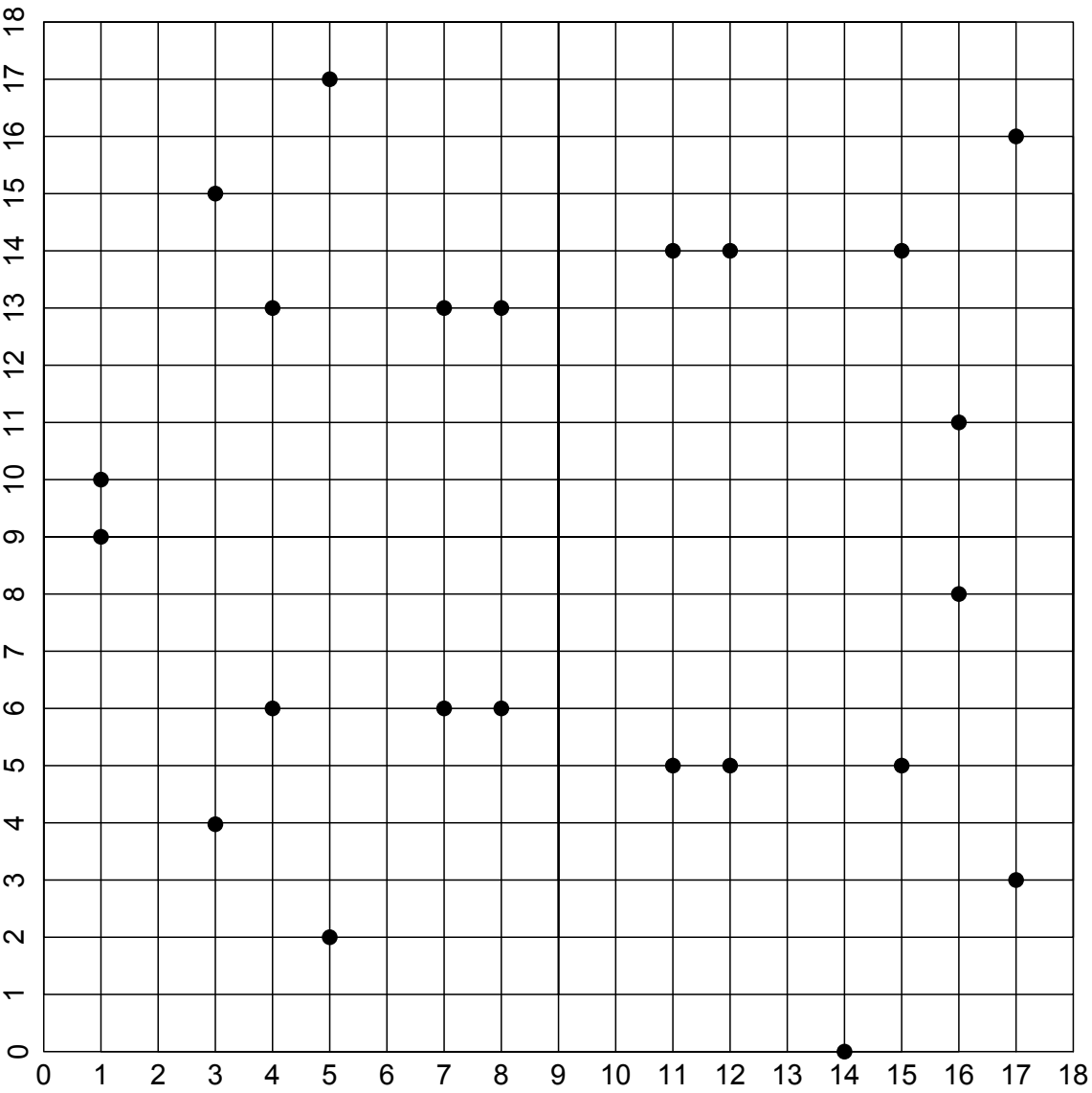


Figure 2.2: The Field of Elliptic Curve Points for $p = 19, a = 2, b = 2$

derived by

$$x_3 = s^2 - x1 - x2 \bmod p \tag{2.31}$$

and

$$y_3 = s\,(x_1 - x_3) \bmod p. \tag{2.32}$$

Subtraction can be achieved by using the reflective point and then adding the two points together such that if

$$P = (x_1, y_1) \bmod p \tag{2.33}$$

then

$$-P = (x_1, -y_1) \bmod p \tag{2.34}$$

such that

$$\begin{aligned} Q - P &= (x_2, y_2) - (x_1, y_1) \mod p \\ &= (x_2, y_2) + (x_1, -y_1) \mod p. \end{aligned} \tag{2.35}$$

The method for providing multiplication is provided by point addition and doubling, where the number of steps is equivalent to the number of bits in the prime. This is accomplished using the left to right method and is exemplified in Table 2.5. This multiplication technique is applied as shown in (2.27) where Alice and Bob both select a random private key $1 < k < g$ and performs

$$K_A = k_a G \mod p \tag{2.36}$$

and

$$K_B = k_b G \mod p \tag{2.37}$$

Sending each other their public keys, Alice and Bob apply their private keys similar to (2.17) such that

$$k_a K_B \mod p = S_k = k_b K_A \mod p. \tag{2.38}$$

Where S_k becomes the shared session key. Using curve P-192 that was defined by the National Institute of Standards and Technology (NIST) [47] shown in Table

Table 2.5: Elliptic Curve Multiplication ($13 \times P$) using Double and Add

Step	Operation		Result	Representative
1		-	P	1_2P
2a	Double	$P + P$	$2P$	10_2P
2b	Add	$2P + P$	$3P$	11_2P
3	Double	$3P + 3P$	$6P$	110_2P
4a	Double	$6P + 6P$	$12P$	1100_2P
4b	Add	$12P + P$	$13P$	1101_2P

Table 2.6: NIST Prime Elliptic Curve P-192

$p_{192} = 6277101735386680763835789423207666416083908700390324961279$
$a = 6277101735386680763835789423207666416083908700390324961276$
$b = 2455155546008943817740293915197451784769108058161191238065$
$G_x = 6020462823756886567582134805875261119166989766368846848 18$
$G_y = 1740503322936220314048575522802194103640234889273866506 41$

2.6, we can perform the following example where

$$\begin{aligned}
K_A &= k_a G \bmod p \\
&= 3546841343384271782483417883356648447359578242035282619 50 \\
&\quad \times (6020462823756886567582134805875261119166989766368846848 18, \\
&\quad\ 1740503322936220314048575522802194103640234889273866506 41) \\
&= (2933140742448625203731641248407683827818026042006007265108, \\
&\quad\ 1342160346315844524113810504268757567486830356720401950124)
\end{aligned} \tag{2.39}$$

$$\begin{aligned}
K_B &= k_b G \bmod p \\
&= 1799075742360715006265099004488067778762135129614130234431 \\
&\quad \times (6020462823756886567582134805875261119166989766368846848 18, \\
&\quad\ 1740503322936220314048575522802194103640234889273866506 41) \\
&= (4147123246579862628199761933154009809426422151451232702766, \\
&\quad\ 4754143181055516863572102753585715493405392185760408272858)
\end{aligned} \tag{2.40}$$

$$\begin{aligned}
S_k &= k_a K_B \bmod p \\
&= 3546841343384271782483417883356648447359578242035282619 50 \\
&\quad \times (4147123246579862628199761933154009809426422151451232702766, \\
&\quad\ 4754143181055516863572102753585715493405392185760408272858) \\
&= (4728039436511587306189881807373555356659618554916171973 73, \\
&\quad\ 4253469622903940143028516631910075165803201780065008295480) \\
&= k_b K_A \bmod p \\
&= 1799075742360715006265099004488067778762135129614130234431 \\
&\quad \times (2933140742448625203731641248407683827818026042006007265108, \\
&\quad\ 1342160346315844524113810504268757567486830356720401950124) \\
&= (4728039436511587306189881807373555356659618554916171973 73, \\
&\quad\ 4253469622903940143028516631910075165803201780065008295480)
\end{aligned} \tag{2.41}$$

As (2.41) shows that a common point on the curve has been established, the next step would be to take the x-coordinate and process it through a typical hashing algorithm, currently SHA-1. The resultant hash will then be used as the symmetric key.

Of particular interest are reports that have been circulating recently suggesting

that the curves offered by the NIST [47] have tainted curves and that they contain known weaknesses or backdoors [48–50]. This has not only brought about the removal of the perpetrated curve and technique, but also led to companies posting advisories and removing support for these curves from their products and applications, with the focus moving to curves that have been developed by the cryptographic community [51–54].

2.4 Cipher Types

There are two distinct methods that underpin how a cipher text is produced. These methods are stream or block ciphers. Each method has an appropriate method of use and will be discussed accordingly.

2.4.1 Stream Ciphers

Stream ciphers are derived by combining the plaintext and the key to derive a cipher text, one bit at a time. This requires combining a key stream by adding it to the plaintext modulo-2 or using the exclusive-or (XOR) operation. The main focus therefore is how the key stream is derived. The most secure way of deriving the key is via the one-time pad (OTP) where the key is pre-computed and used only once. However, deriving one-time pads must be undertaken in advance and will obviously require storage.

Another method, which is more commonly used, is the linear feedback shift register (LFSR) and applications may use one or more of these. The LFSR works by shuffling a series of bits through a register, the length of which will contribute to determining the periodicity of the register. Taps are taken at certain points of the register and these values are added together modulo-2. The resultant value is then placed at the beginning of the register when all the bits shuffle to the next position. The key would therefore be determined by the initial state of the LFSR prior to the commencement of the operation. All LFSRs are cyclic or periodic and depending on their design determines their length and whether they can utilise the full space, which would be $2^n - 1$ values, where n is the number of bits it contains. The only value that a running LFSR cannot produce is the value zero, as there would be no way otherwise to change this value.

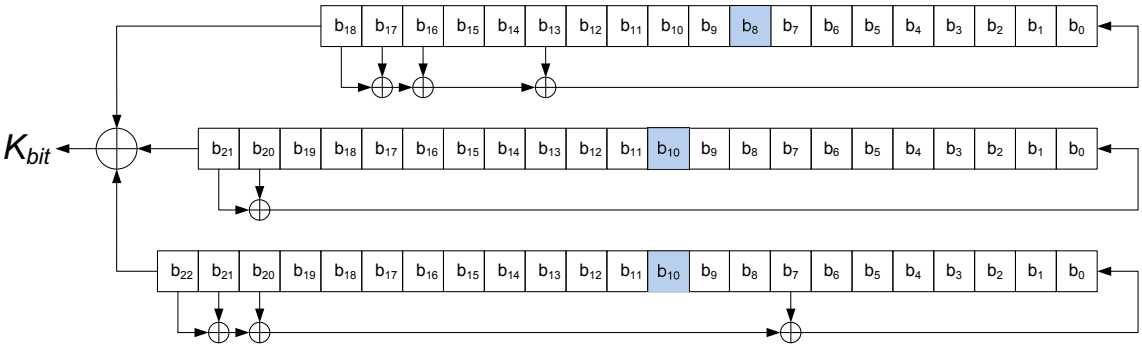


Figure 2.3: A5/1 Cipher Scheme

2.4.1.1 A5/1

The A5/1 cipher that was initially used with GSM mobile phones was designed using three different LFSRs of varying bit sizes [55]. It combines the output of three LFSRs that are built of varying sizes and configuration that are derived from the following polynomials

$$LFSR_1 = x^{19} + x^5 + x^2 + x + 1 \tag{2.42}$$

$$LFSR_2 = x^{22} + x + 1 \tag{2.43}$$

$$LFSR_3 = x^{23} + x^{15} + x^2 + x + 1. \tag{2.44}$$

The LFSRs are constructed by using the first term, the degree of the polynomial, to determine the length of the LFSR and taking the taps at positions determined by the remaining terms, applying it to the LFSR by reading left to right. Therefore (2.42) would produce an LFSR with bit taps at positions 18, 17, 16 and 13. The output from each LFSR is, as well as being used as a feedback, combined with the outputs of the other LFSRs using addition modulo-2. The resultant bit is then used

Table 2.7: LFSR Clocking Behaviour for A5/1

Clocking Bit			Movement		
$LFSR_1$	$LFSR_2$	$LFSR_3$	$LFSR_1$	$LFSR_2$	$LFSR_3$
0	0	0	1	1	1
0	0	1	1	1	0
0	1	0	1	0	1
0	1	1	0	1	1
1	0	0	0	1	1
1	0	1	1	0	1
1	1	0	1	1	0
1	1	1	1	1	1

as a key bit to apply to the plaintext stream. The configuration of this scheme is shown in Figure 2.3 where the shaded bits are identified as the clocking bits. Each LFSR is clocked according to the value of its clocking bit against the values of the clocking bits in the other LFSRs. Movement occurs for an LFSR if its clocking bit matches the clocking bit of one or more of the other LFSRs. How this would affect the movement is shown in Table 2.7.

2.4.2 Block Ciphers

Block ciphers are currently the most common type of cipher in use today. While a stream cipher encrypts the plaintext on a bit-by-bit basis, changing one of the bits can only have an impact on that one bit. However, a block cipher is encrypted as a group of bits and because of this, a change to any of the bits within the block should have an impact on the remaining bits. An example of an encryption scheme that uses this type of scheme is the data encryption standard (DES) [56].

Techniques relating to block encryption have previously been described as falling into two distinct categories; confusion and diffusion [12]. The confusion aspect is generally regarded as replacing the bytes in the plaintext such that there is little to no relationship with the key. This is either undertaken by a pre-calculated lookup table where values have maximal separable distance, or functions that have a non-linear element that will appear to sever the relationship. Diffusion is the process of using each input byte to affect and impact the other bytes in the input and corresponding cipher text. The result of this would mean the smallest change in the cipher text, or slightest alteration in the key would have a devastating impact upon the rest of the cipher text and resultant plaintext.

2.4.2.1 Substitution Boxes

The substitution box or S-box is the core of the confusion operation, proving a method to reversibly transpose incoming data. This can be in the form of a mathematical function, for example where data is swapped-out with its multiplicative inverse so that the original data can be retrieved by reversing the process. Typically, these types of operations are carried out using a Galois field (GF). In addition, they may also include processes that are non-linear so as to resist linear analysis.

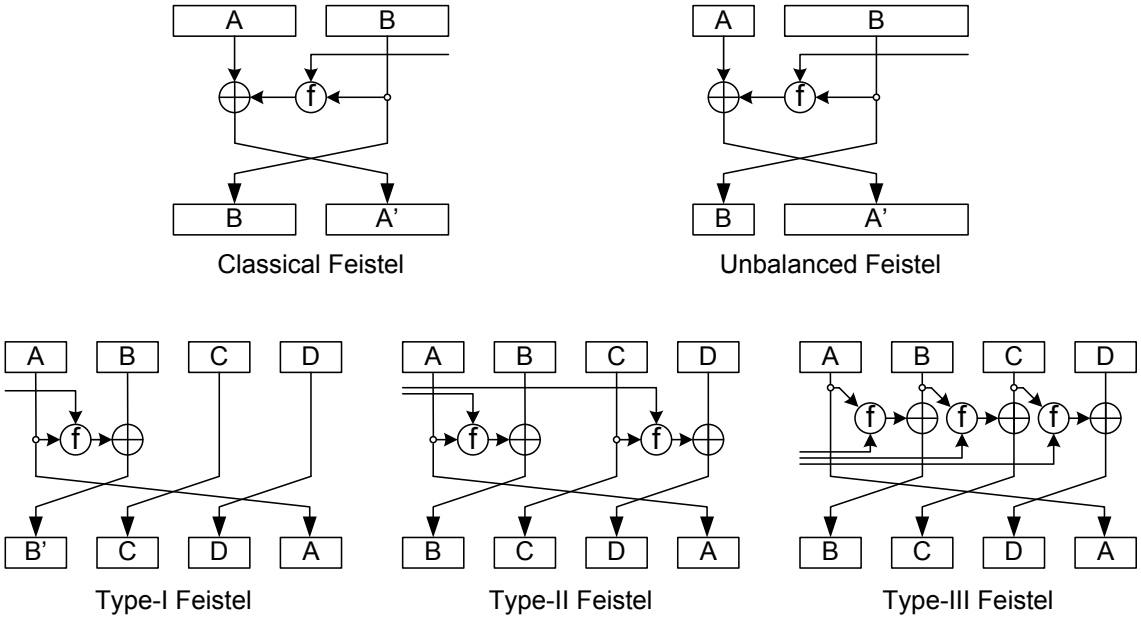


Figure 2.4: Selection of Feistel Networks

2.4.2.2 Permutation Boxes

The job of the permutation box or P-box is to mix the bytes of the cipher text around so that individual bits within each byte are spread and swapped with the bits of the other bytes. This results in changes in a single bit having a catastrophic impact on the whole of the cipher text. When used in conjunction with S-boxes over a number of stages they are together usually referred to as a substitution-permutation network (SPN) and are usually found in Feistel networks.

2.4.2.3 Feistel Networks

Feistel networks are structures that split the incoming cipher text into two or more parts that are typically applied to one or more functions at each stage depending on their classification, as generalised by [57]. The function facilitates the development of new transforms based on these structures, and can incorporate many techniques including S-boxes, Boolean functions and even transforms. The Feistel network was implemented in the DES, which was ratified for use in 1977 and uses what is now known as a classical Feistel network [56]. Other Feistel constructions include the unbalanced, type-I, type-II and type-III Feistel networks [57] and are illustrated in Figure 2.4.

2.4.2.4 Transforms

Another example that can be used as both a method of confusion and diffusion are transforms. These include the Walsh-Hadamard transform (WHT) as used in [58], the pseudo-Hadamard transform (PHT) as used in [59,60] and the number theoretic transform (NTT) as used in [61], particularly the NMNT. Typically, the NTT is a transform that is defined over a finite field modulo a prime. The NMNT uses a Mersenne prime, which while can be implemented using simple shifts and additions, serves to be a very effective operator without the need for time-consuming divide algorithms.

Recently, there is evidence of encryption systems that are derived using S-boxes and P-boxes, which are substitution and permutation boxes respectively. These methodologies are based around Shannon's ideals of message encryption to be based on confusion and diffusion [12]. The S-box, being the confusion aspect, replaces a byte within the message with another byte that is determined by the function of the box and the key. In order to resist cryptanalysis, it is important that this stage is non-linear, otherwise a linear cryptanalysis (LC) attack could potentially be applied. The importance of the P-box would be to then mix around the bits of the cipher text so that individual bits within each byte become dependent on the bits of other bytes that were also being encrypted. This helps to resist differential cryptanalysis (DC).

2.4.2.5 Data Encryption Standard

The DES was developed by IBM [62] and has since been extensively studied since its publication for use in 1977 [63]. It is based on the classical Feistel construction where the incoming code is split into two equal sized blocks and a single function element is used to process each alternate block every round as shown in Figure 2.5. Consisting of 16 rounds for each block, it is a very secure algorithm, even though it uses a key that is too short by today's standards.

This algorithm was in use for over 20 years and precedes the advanced encryption standard (AES). Curiously, whilst it uses a 64-bit block and takes a 64-bit key, it utilises only 56 bits of the key. The bits that aren't used as part of the key are deemed parity bits, although it can be seen from the design that they literally are not used.

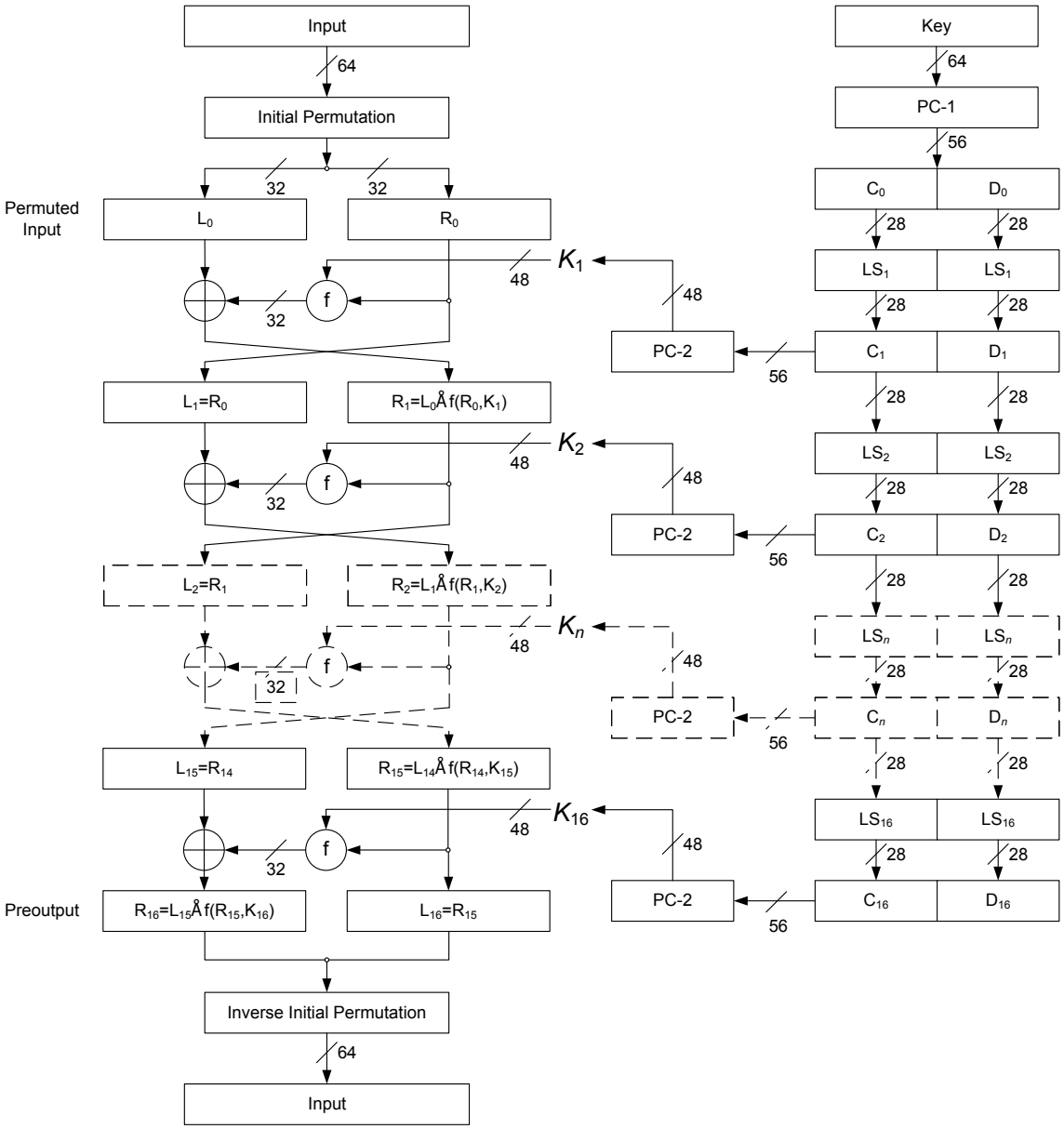


Figure 2.5: The DES Algorithm

The algorithm begins with a permutation function and ends with the inverse permutation function as shown in Figure 2.6, helping to spread the bits across the block during the encryption process and thus strengthening its diffusion properties. Following this, the algorithm then falls through 16 rounds of Feistel networks where 32 block bits are processed at any one time. The function depicted in Figure 2.7 consists of block expansion, where the 32 block bits being processed are expanded to 48 bits. The key is processed as two 28-bit halves, where 56 bits are selected from the initial key and permutated; the unused parts have no connections and are shown as shaded. A new key is derived each round and this is achieved by rotating each half and selecting a total of 48 bits from both halves via another permutation; again showing the unused parts with no connections as shaded. This round key is

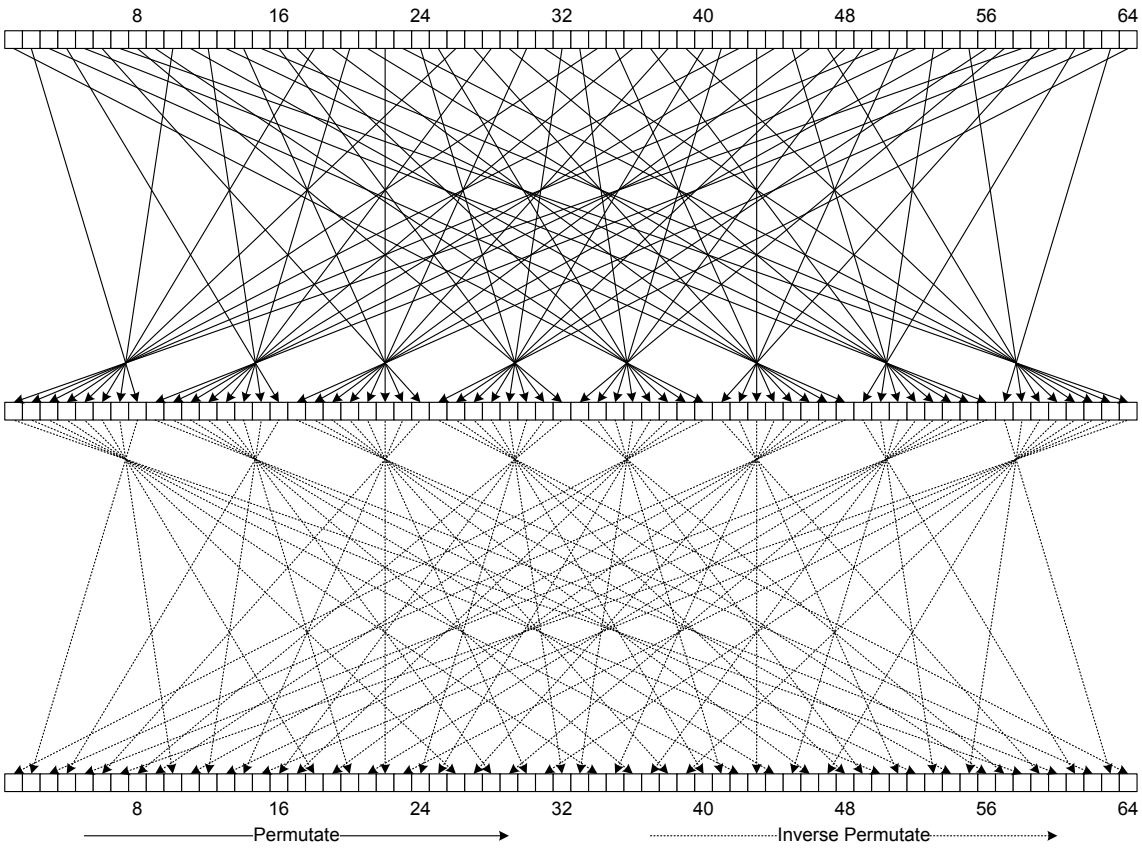


Figure 2.6: Initial and Inverse Permutate Functions

added to the expanded block text modulo-2. The intermediately processed 56-bit key is reused for repeating the rotation and permutation for each successive round serving as the round key. After the key has been applied to the block, the result is fed into eight six-input S-boxes that each have four outputs. This reduces the cipher back to 32 bits. The S-box works by using the outer-most bits of the input to select a row of a lookup table within. After selecting one of the four rows using the outer bits, the inner four bits are then used to select the column that contains the output. This is shown more clearly in Figure 2.8. The DES encryption scheme has been one of the most studied encryption systems to date and has emphasised new areas of cryptanalysis with respect to DC and LC [64–66]. However, these techniques were described in [67] where DC was previously known as a "T attack". Additionally, all of the attention that DES has received has resulted in ways to measure advances in technology and techniques in methodologies and implementations where there have been repeated attempts to defeat DES in the shortest amount of time possible using brute force attacks [68–71].

Prior to the introduction of AES, moves were made to strengthen DES, accomplished by applying the algorithm three times in succession using three different keys. This methodology was known as the triple data encryption

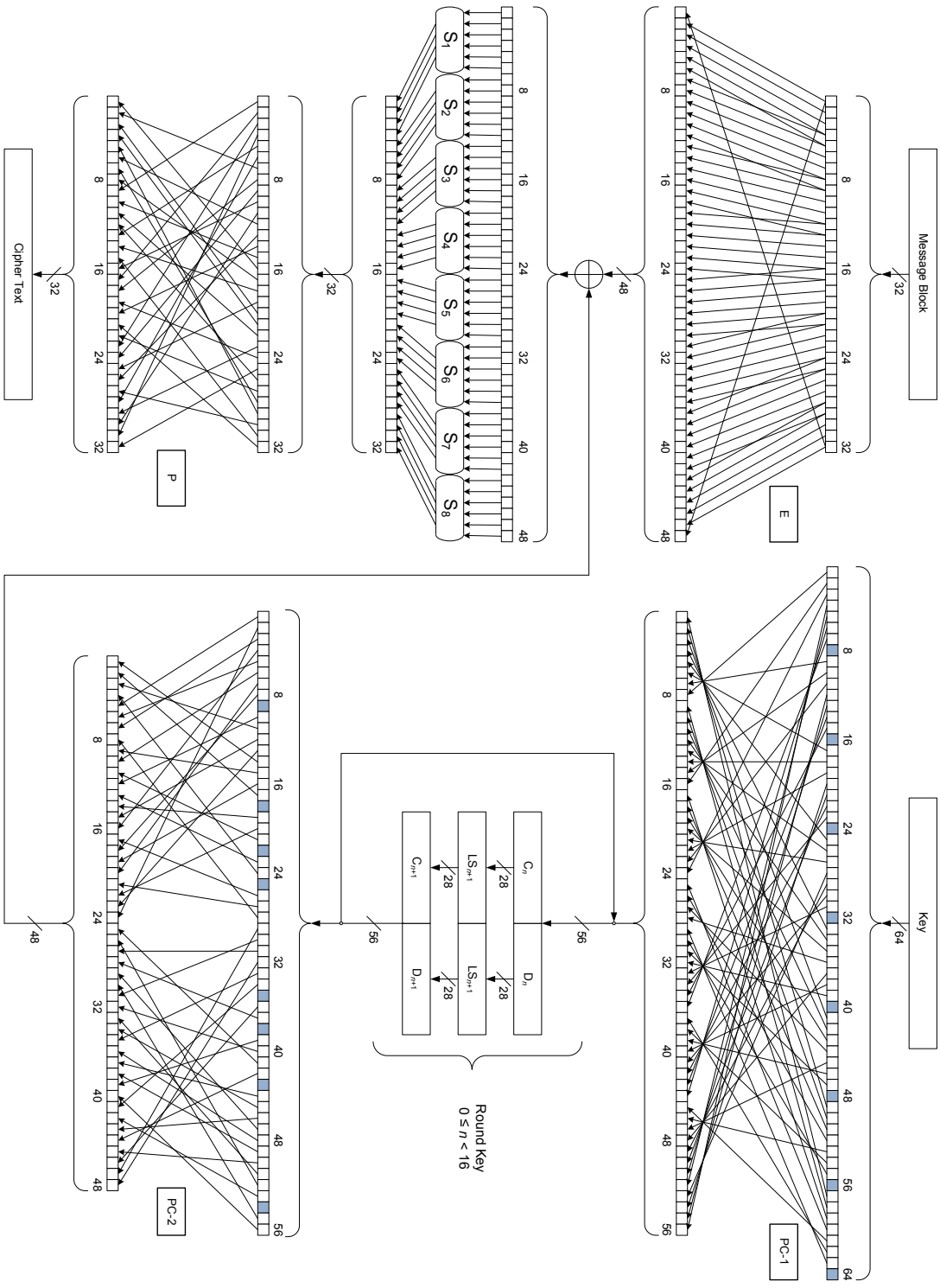


Figure 2.7: The F-Box in DES Feistel Network with Round Key

algorithm (TDEA) and triple-DES (TDES) or (3DES) and effectively increased the key size to 168 bits. This works by using three separate keys and by applying

$$c = DES\left\{E_{k_3}, DES\left[D_{k_2}, DES\left(E_{k_1}, m\right)\right]\right\} \tag{2.45}$$

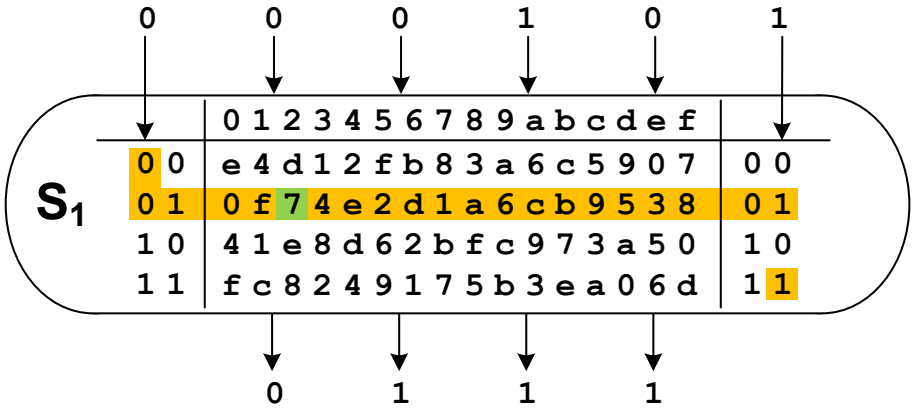


Figure 2.8: An S-Box from DES

for encryption and

$$m = DES\{D_{k_1}, DES[E_{k_2}, DES(D_{k_3}, c)]\} \tag{2.46}$$

for decryption where E_{k_n} and D_{k_n} represent the different encryption and decryption keys respectively. This makes DES a significantly stronger algorithm and remains one of the more robust algorithms developed.

2.4.2.6 Advanced Encryption Standard

While DES was coming to its end of life, the department of Federal Information Processing Standards (FIPS) announced the development of the AES [72]. The standard stated that the algorithm to be used was to be a block cipher that had a 128-bit block and variable keys sizes that include 128-, 192- and 256-bit derivations. Rather than developing a standard behind closed doors that could possibly contain any number of potential weaknesses, an open and inclusive approach was adopted in the development of the standard by inviting the current leaders in the field to submit their ideas. As a result, twenty-one algorithms were submitted and fifteen were short-listed for the first round [73]. By 2000 this number was reduced to five finalists [60, 74–78]. One month after the final conference in 2000, NIST announced that Rijndael had been selected according to the cumulative decisions by the attendees as shown in Table 2.8 [79] and it was adopted as the new encryption standard [80].

As AES has now superseded DES as the accepted encryption standard for United States commerce by NIST, it is seeing significant attention. Probably one of the biggest criticisms is that it doesnt contain enough rounds and that there

should be at least 16, 20 or 28 rounds depending on whether the key is 128-, 192- or 256-bits in size [81]. Nevertheless, it now remains the standard against which all emerging encryption systems must measure themselves.

Rijndael uses the SPN and runs over 10, 12 or 14 rounds depending whether the key size if 128-, 192- or 256-bits accordingly. It is arranged as a 4x4 block of bytes and uses an S-box developed over $GF(2^8)$ with an affine transform. Diffusion is further applied using the mix-rows and mix-columns, which entails the rotation of each row by a successive number of times followed by an invertible matrix multiplication of each column. Significantly, Rijndael has been developed as a series of byte operations rather than using full 32-bit words like other finalists. This makes the algorithm very attractive for devices with limited processing. It also allows for the majority of the calculations to be pre-processed in advance and referred to in lookup tables.

The AES algorithm will be discussed more in depth in Chapter 6 where a stripped-down version will be used without the password functionality in order to measure it against other diffusion techniques.

2.5 Mode Types

The encryption modes are techniques that are used within the encryption process to make the resultant cipher more robust. Without these modes, cipher texts would be easier to attack owing to techniques such as KPAs or known cipher-only attacks (COA). This section will present the five confidentiality modes that have been ratified as applicable for use with AES [82]. Additional modes have been ratified for use with AES depending on the purpose including: authentication [83], authenticated encryption [84], high-throughput authenticated encryption [85] and storage [86]. As well as presenting the methodologies associated with each of the

Table 2.8: Votes for AES Selection

Algorithm	Yes	No	Score
Rijndael	86	10	76
Serpent	59	7	52
Twofish	31	21	10
RC6	23	37	-14
MARS	13	84	-71

confidentiality modes, these modes are also depicted in Figure 2.9.

2.5.1 Electronic Code Book

The electronic code book (ECB) mode is the simplest of all the modes. It processes the plain text one block at a time before encrypting it and therefore each cipher block is completely independent from the others. This mode is not particularly secure as patterns can occur should identical plaintext blocks be encrypted, which will result in the production of identical cipher blocks. This is particularly prevalent with content that is uncompressed where repetition is more common. The use of ECB is strongly discouraged as it opens the underlying encryption system to KPAs. The procedure for using ECB is

$$C_j = E_K\left(P_j\right) \text{ for } j = 1, 2, \ldots, n \tag{2.47}$$

for encryption and

$$P_j = D_K\left(C_j\right) \text{ for } j = 1, 2, \ldots, n \tag{2.48}$$

for decryption.

2.5.2 Cipher Block Chaining

Cipher block chaining (CBC) mode is similar to ECB mode except that before the plaintext block is encrypted, it is added modulo-2 to the previous cipher text block. However, In the case of the first block where no previous cipher block yet exists, an Initialisation Vector (IV) is used to ensure the integrity of the first block. Both ECB and CBC modes require that the underlying encryption system is capable of decryption as well as encryption. The formula for using this mode is based on the first block being index as $j = 1$ and apply for encryption

$$\begin{aligned} C_0 &= IV \\ C_j &= E_K\left(P_j \oplus C_{j-1}\right) \text{ for } j = 1, 2, \ldots, n \end{aligned} \tag{2.49}$$

and for decryption

$$\begin{aligned} C_0 &= IV \\ P_j &= D_K\left(C_j\right) \oplus C_{j-1} \text{ for } j = 1, 2, \ldots, n. \end{aligned} \tag{2.50}$$

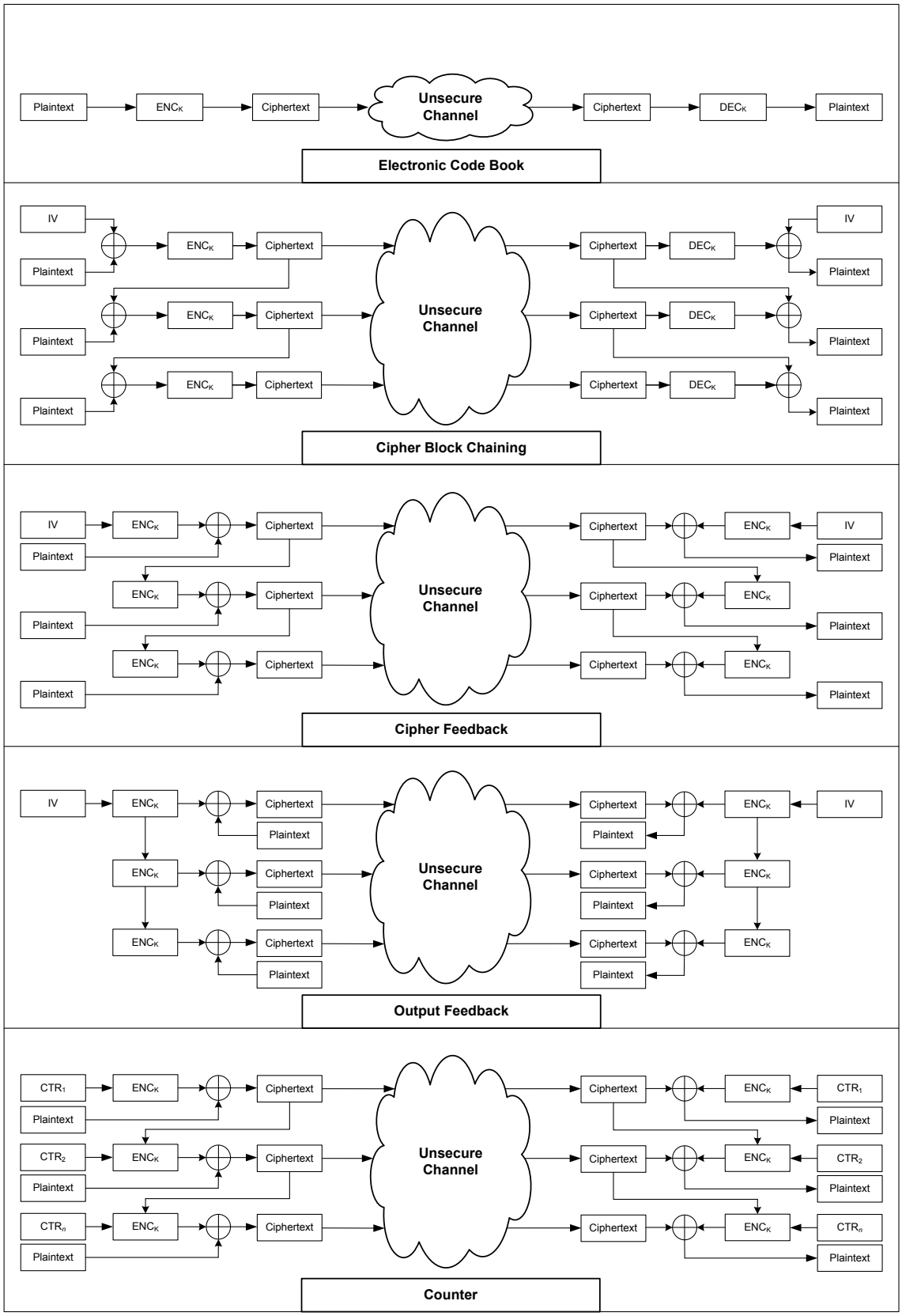


Figure 2.9: The Five Confidentiality Encryption Modes Ratified for AES

A major disadvantage with this mode is that while the ECB mode will lose a block should as little as a single bit be changed, with CBC mode the current and following blocks will be destroyed. This would make it more sensitive in noisy

channels. However, if the bit has been changed through cryptanalysis then this is obviously going to be an advantage.

2.5.3 The Initialisation Vector and Streams

The CBC mode introduced a new variable, in the IV; its role is to apply cryptographic 'salt' that is a random number to be used only once. As such, this may also be known as a 'nonce' indicating that it is a number to be used only once. When used correctly, it works as a safeguard to ensure integrity of the cipher should the user choose the same password. The remaining three confidentiality modes will also use an IV, including the CTR mode where it will use each block, incrementing each time. Protecting the IV is not necessary and it may well be transmitted with the cipher. However, if it is desired that the IV is concealed, using a public key system can help maintain confidentiality. The following three confidentiality modes are designed such that the plaintext is applied to the system after the block has been encrypted. As such, these modes can be used as steam modes where only a desired number of bits will be taken from the encrypted block and added modulo-2 to the plaintext to create the cipher. This means that these modes require that the encryption system uses the encrypting algorithm only because the decryption process is simply the repeat of the encryption process, but with the desired number of bits from the encrypted block modulo-2 to the cipher, thus producing the original plaintext. Therefore the decryption function in systems using these modes is ultimately redundant.

2.5.4 Cipher Feedback

Cipher Feedback (CFB) mode is a very similar process to CBC mode but rather than adding the plaintext and previous ciphertext (or the IV if it is the first block) modulo-2 prior to encryption, it is instead performed after encryption of the previous cipher text (or IV if first block). Encryption is performed

$$\begin{aligned} C_0 &= IV \\ C_j &= E_K\left(C_{j-1}\right) \oplus P_j \text{ for } j = 1, 2, \ldots, n. \end{aligned} \tag{2.51}$$

and decryption as

$$\begin{aligned} C_0 &= IV \\ P_j &= E_K\left(C_{j-1}\right) \oplus C_j \text{ for } j = 1, 2, \ldots, n. \end{aligned} \tag{2.52}$$

2.5.5 Output Feedback

The output feedback (OFB) mode is similar to CFB mode but with a minor distinction that the output of the encrypted block is used in conjunction with the key in subsequent encryption blocks rather than the resultant cipher. Again, it is the result of these encrypted blocks that are added modulo-2 with plaintext block. Alternatively, a selected number of bits may be used instead, which could therefore turn this block encryption system into a stream encryption system. To process this mode the following operations are used to set up the system

$$\begin{aligned} O_0 &= IV \\ O_j &= E_K\left(O_{j-1}\right) \text{ for } j = 1, 2, \ldots, n. \end{aligned} \tag{2.53}$$

then

$$C_j = P_j \oplus O_j \tag{2.54}$$

for encryption and

$$P_j = C_j \oplus O_j \tag{2.55}$$

for decryption.

2.5.6 Counter

The counter (CTR) mode is a method utilising an IV as the input to an encryption system and subsequently incrementing it at each round. Like the CFB and OFB modes, the encrypted block is added modulo-2 to the plaintext to create the cipher. There are therefore no feedback operations in this mode, which means that many ciphers can be processed in parallel. Encryption is performed

$$\begin{aligned} I &= IV \\ C_j &= E_K\left(I + j - 1\right)) \oplus P_j \text{ for } j = 1, 2, \ldots, n \end{aligned} \tag{2.56}$$

and decryption as

$$\begin{aligned} I &= IV \\ P_j &= E_K\left(I + j - 1\right)) \oplus C_j \text{ for } j = 1, 2, \ldots, n. \end{aligned} \tag{2.57}$$

2.6 Hashes

Depending on the nature of the application, there are additional types of encryption that may not necessarily perform a reversible operation in a manner that is expected. These encryption types are referred to as hashes. Hashes are one-way methods, typically containing between 128 and 512 bits and are used whenever a user sends or receives content that requires confirmation of message integrity or authentication. Typical examples of this are the message digests MD4 and MD5 [87, 88], the secure hash algorithms, SHA1 and SHA2 [89] and the newly ratified SHA3 [90], which is currently in draft format. The hash operation is synonymous to encrypting a message block by block and cumulatively combining the encrypted outputs, thus resulting in what should be an irreversible process. The security of the hash value and its underlying algorithm is the assurance that the hash value from one message cannot be duplicated by manipulating a different message. When a method is discovered to reproduce values this is called a collision and rapidly degrades the value of the algorithm.

Hashes, when used with public key cryptography, can serve as an important method of authentication as well as proving integrity. This is accomplished with the sender first deriving the hash of a message using the same previous methods. However, upon deriving the hash, it is encrypted using the senders private key. By decrypting the hash using the senders public key and verifying the hash, both message integrity and authentication of source can be proven. For example, a user may request a connection with a secure service and in doing so he sends his public key to the service. The service responds by encrypting a session key with the users public key, using previously agreed techniques relating to public key exchange and requesting that the user provides authenticating details using the session key. The user may then decrypt the session key using his matching private key and encrypting his password with the provided session key. The encrypted password is then passed through a hash function so that he can then send his user identification and hash to the service. The service will be able to reliably verify the hash by

recreating the same process using the users stored password. This adds enhanced security by avoiding the actual password from traversing an unsecured channel. The use of hashes in this manner is typically referred to as a message authentication code (MAC) and embodies the integrity and authentication operation.

2.7 General Purpose Graphics Processing Unit Processing

In 2006, NVIDIA announced the software development kit (SDK) to their proprietary compute unified device architecture (CUDA) [91], inspiring researchers and developers to push the boundaries of their developments to new architectures using graphics cards [92–94]. Meanwhile, AMD were working on their solution, close to metal, which was released in November the same year [95]. Two years later the Khronos Compute Working Group was formed [96] leading to the release of their Open Computer Language (OpenCL) [97], which although originally developed by Apple, was supported by both AMD and NVIDIA. These two architectures epitomise what we know today as general processing graphics processing unit (GPGPU) processing.

Cell processors in the Sony Playstation 3, which contain accessible compute engines for the graphics, were also discovered to be fully available to system developers [98]. In 2008, a researcher in the field of astrophysics formed a cluster, in which eight of these units were interconnected, costing less than two simulations on an available supercomputer [99]. Unfortunately, while Sony once participated and encouraged the use of their platforms for this purpose, they have since shut down this resource to developers [100], currently leaving CUDA and OpenCL as the two main contenders. However, the topic of GPGPU is currently an extremely vibrant area, with frequent publications describing new achievements; such as the ability to brute force every Windows NTLM hash derived from every possible eight-character password, based upon 52 letters (upper- and lower-case), 10 digits and 31 special characters [101]. This is a very realistic supposition as should the hacker gain access to the Windows password file, then they would be able to process some 93^8 guesses in under 6 hours to determine the containing passwords.

GPGPU processing involves using a number of graphics cards, between one

and four, within a desktop computer as a single host that can execute a users program. Kernels are developed using the appropriate development system and uploaded to the graphics card(s) along with datasets and parameters, including the number of threads that are to be executed. These kernels can be executed either synchronously, where the host program waits for the kernel to complete; or asynchronously, where an interrupt can be generated within the host program upon completion. Using graphics cards in this manner can be extremely beneficial as they are naturally highly parallel by design for their primary function to process graphical requirements; they are compact and have very low power requirements as opposed to their thoroughbred supercomputer counterparts [102]. Moreover, because these graphics cards are designed to be implemented in reasonably low-power hosts, they are massively energy efficient in terms of giga floating point operations per second (GFLOPS) per Watt, in comparison to other supercomputing topologies. Applications that exploit the potential of GPGPU are known as heterogeneous computing, owing to the significant processing workloads that can be offloaded to one or more GPUs from the CPU.

Between the rapid development of GPGPU technology and associatively low cost, the requirement for strong encryption has become a necessity, especially with the vast amount of ongoing research and development to create GPGPU-based algorithms to defeat security. Therefore, it is the balance in the critique between the security strength of encryption algorithms and the evaluation of that strength in terms of technological development.

2.8 Conclusion

This chapter has emphasised the scope and demand for secure methods of communication and storage. Through advances in technology and techniques, encryption processes are being broken and made redundant. Whilst there is a demand for emerging techniques to be processed quickly and efficiently, care and consideration must be taken into account with respect to the life span of the development of new techniques. Parallel processing has been brought to the consumer through affordable graphics cards where the advent of GPGPU processing has made a significant impact in the field of cryptanalysis. Techniques using this subject will be presented in Chapter 6 where simulations and analysis will be

undertaken using GPGPU processing.

Chapter 3

The Generalised New Mersenne Transform

3.1 Introduction

There are many different transforms that are applicable for use in signal-processing, such as the Fourier-, cosine-, sine- and wavelet-transforms to name a few. These techniques have applications that include image- and audio-processing and communications. While fundamental, these transforms are error-prone as they are built upon irrational functions that are subject to rounding and truncation errors, which therefore have limited use for applications in cryptography. A separate branch of transforms called NTTs are built over a residue field modulo a prime, such as the Fermat number (FNT), MNT and the NMNT. These particular transforms incorporate Fermat and Mersenne prime numbers respectively, which allow for extremely fast calculations using shifts and additions. As these transforms are produced using modulo operations, the results they provide are exact as they contain no irrational functions and therefore result in no rounding or truncation errors.

One of the more recently created NTTs is the GNMNT, which was developed by expanding the kernel parameters of the original NMNT, during which two new transforms were invented; ONMNT and the O^2NMNT. The GNMNT, like the NMNT is built over a field modulo a Mersenne prime and shares similar properties such as cyclic convolutions. Contrary to previous techniques such as the FNT and MNT, it has both long and versatile transform lengths that are

defined over a power of two, making it suitable for application development using fast algorithms. Like the original NMNT, it too has applications in image-, audio- and signal-processing where techniques involve one- (1D) and two-dimensional (2D) correlation and convolutions for matching and filtering, as well as applications in cryptography. This chapter will start by introducing the NMNT algorithm followed by the ONMNT and O^{2}NMNT algorithms. The derivation of the transform parameters will be briefly discussed followed by the selection of the GNMNT parameters and the taxonomy of the GNMNT will be included. The GNMNT kernel components will be discussed and how key components can influence the results will be discussed. Finally, examples of using the GNMNT to undertake tasks for encryption will be demonstrated before providing conclusions.

3.2 The NMNT

The NMNT is the original transform derived in this suite and as such, there are already significant developments in its use within signal- [6, 103–105] and image-processing [7, 106–108], cryptography [61, 109–111] for example, as well as being the topic of focus for implementations [112–115]. Its derivation is

$$X(k) = \left\langle \sum_{n=0}^{N-1} x(n)\beta\left(nk\right) \right\rangle_{Mp} \quad \text{for } k = 0, 1, 2, ..., N-1 \tag{3.1}$$

where $\langle . \rangle_{Mp}$ denotes modulo Mp, $Mp = 2^p - 1$ is a Mersenne prime for $p = 2, 3, 5, 7, 13, 17, 19, 31, ..., etc$ and N is a power of two where $N \leq 2^p$. The transform kernel β is given by

$$\beta(n) = \langle \beta_1(n) + \beta_2(n) \rangle_{Mp} \tag{3.2}$$

where

$$\beta_1(n) = \langle \mathrm{Re}(\alpha_1 + j\alpha_2)^n \rangle_{Mp}, \; \beta_2(n) = \langle \mathrm{Im}(\alpha_1 + j\alpha_2)^n \rangle_{Mp} \tag{3.3}$$

for

$$\alpha_1 = \pm \langle 2^q \rangle_{Mp}, \; \alpha_2 = \pm \langle -3^q \rangle_{Mp} \text{ and } q = 2^{p-2} \tag{3.4}$$

and Re(.) and Im(.) denote the real and imaginary parts of the enclosed terms respectively. For transform lengths equal to $\frac{N}{d}$, β_1 and β_2 can be calculated as

$$\beta_1(n) = \left\langle \mathrm{Re}\left[\left(\alpha_1 + j\alpha_2\right)^d \right]^n \right\rangle_{Mp} \tag{3.5}$$

Table 3.1: The NMNT Kernel for $N = 16$ and $Mp = 127$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	82	111	82	1	3	0	124	126	45	16	45	126	124	0	3
1	111	1	0	126	16	126	0	1	111	1	0	126	16	126	0
1	82	0	45	126	3	111	3	126	45	0	82	1	124	16	124
1	1	126	126	1	1	126	126	1	1	126	126	1	1	126	126
1	3	16	3	1	45	0	82	126	124	111	124	126	82	0	45
1	0	126	111	126	0	1	16	1	0	126	111	126	0	1	16
1	124	0	3	126	82	16	82	126	3	0	124	1	45	111	45
1	126	1	126	1	126	1	126	1	126	1	126	1	126	1	126
1	45	111	45	1	124	0	3	126	82	16	82	126	3	0	124
1	16	1	0	126	111	126	0	1	16	1	0	126	111	126	0
1	45	0	82	126	124	111	124	126	82	0	45	1	3	16	3
1	126	126	1	1	126	126	1	1	126	126	1	1	126	126	1
1	124	16	124	1	82	0	45	126	3	111	3	126	45	0	82
1	0	126	16	126	0	1	111	1	0	126	16	126	0	1	111
1	3	0	124	126	45	16	45	126	124	0	3	1	82	111	82

and

$$\beta_2(n) = \left\langle \mathrm{Im}\left[(\alpha_1 + j\alpha_2)^d\right]^n \right\rangle_{Mp} \tag{3.6}$$

where $d = \frac{2^{p+1}}{N}$ is an integer power of two and the term $(\alpha_1 + j\alpha_2)$ is of the order 2^{p+1}. The inverse NMNT is defined as

$$x(n) = \left\langle N^{-1} \sum_{k=0}^{N-1} X(k)\beta\,(nk) \right\rangle_{Mp} \quad \text{for } n = 0, 1, 2, ..., N-1. \tag{3.7}$$

It can be seen that (3.1) is the same as its inverse (3.7) except for the scale factor N^{-1}. This indicates that both the forward and inverse transforms can be implemented by the same algorithm with minimal intervention to differentiate between the two. This can easily be verified by

$$\left\langle \mathcal{G}_N \mathcal{G}_N N^{-1} \right\rangle_{Mp} = I \tag{3.8}$$

where $\mathcal{G}_N$ is an NMNT transform matrix of length N, N^{-1} is the corresponding scaling factor and I is the identity matrix. Moreover, it can be further observed that the NMNT is its own transpose because $\mathcal{G}_N = \mathcal{G}_N^T$ and therefore it is also orthogonal. An example of an NMNT transform matrix is shown in Table 3.1 where $N = 16$ and $p = 7$ thus producing $Mp = 127$. The NMNT, as well as being

Table 3.2: The ONMNT Kernel for $N = 16$ and $Mp = 127$

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
72	19	19	72	5	99	28	122	55	108	108	55	122	28	99	5
82	82	3	124	45	45	124	3	82	82	3	124	45	45	124	3
19	5	122	108	28	72	72	28	108	122	5	19	99	55	55	99
111	0	16	0	111	0	16	0	111	0	16	0	111	0	16	0
19	122	122	19	28	55	72	99	108	5	5	108	99	72	55	28
82	45	3	3	45	82	124	124	82	45	3	3	45	82	124	124
72	108	19	55	5	28	28	5	55	19	108	72	122	99	99	122
1	126	1	126	1	126	1	126	1	126	1	126	1	126	1	126
5	28	28	5	55	19	108	72	122	99	99	122	72	108	19	55
3	3	45	82	124	124	82	45	3	3	45	82	124	124	82	45
99	72	55	28	19	122	122	19	28	55	72	99	108	5	5	108
0	111	0	16	0	111	0	16	0	111	0	16	0	111	0	16
28	72	72	28	108	122	5	19	99	55	55	99	19	5	122	108
124	3	82	82	3	124	45	45	124	3	82	82	3	124	45	45
122	28	99	5	72	19	19	72	5	99	28	122	55	108	108	55

orthogonal is also symmetric, producing a kernel that looks like

$$M = \begin{bmatrix} \beta(0) & \beta(0) & \beta(0) & \beta(0) & \dots & \beta(0) \\ \beta(0) & \beta(1) & \beta(2) & \beta(3) & \dots & \beta(N-1) \\ \beta(0) & \beta(2) & \beta(4) & \beta(6) & \dots & \beta[2(N-1)] \\ \beta(0) & \beta(3) & \beta(6) & \beta(9) & \dots & \beta[3(N-1)] \\ \beta(0) & \beta(4) & \beta(8) & \beta(12) & \dots & \beta[4(N-1)] \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \beta(0) & \beta(N-1) & \beta[2(N-1)] & \beta[3(N-1)] & \dots & \beta[(N-1)(N-1)] \end{bmatrix}. \quad (3.9)$$

3.3 The ONMNT

The ONMNT is the first new transform of the GNMNT. Unlike the NMNT, it has individual forward and inverse transforms, where the inverse can easily be obtained by transposing the ONMNT matrix. This is analogous to the discrete cosine transform (DCT), where the type-III is the inverse of the type-II. The maximum transform length is slightly shorter than the NMNT and is defined as 2^{p-1}. The kernel is derived by replacing k from the NMNT with $\frac{2k+1}{2}$, which therefore defines the forward transform as

$$X_O(k) = \left\langle \sum_{n=0}^{N-1} x(n)\beta\left(\frac{2k+1}{2}n\right) \right\rangle_{Mp} \quad \text{for } k = 0, 1, 2, ..., N-1. \quad (3.10)$$

The kernel that is produced looks like

$$M = \begin{bmatrix} \beta(0) & \beta(0) & \beta(0) & \beta(0) & \dots & \beta(0) \\ \beta\left(\frac{1}{2}\right) & \beta\left(\frac{3}{2}\right) & \beta\left(\frac{5}{2}\right) & \beta\left(\frac{7}{2}\right) & \dots & \beta\left(\frac{2N-1}{2}\right) \\ \beta(1) & \beta(3) & \beta(5) & \beta(7) & \dots & \beta(2N-1) \\ \beta\left(\frac{3}{2}\right) & \beta\left(\frac{9}{2}\right) & \beta\left(\frac{15}{2}\right) & \beta\left(\frac{21}{2}\right) & \dots & \beta\left[\frac{3(2N-1)}{2}\right] \\ \beta(2) & \beta(6) & \beta(10) & \beta(14) & \dots & \beta[3(2N-1)] \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \beta\left(\frac{N-1}{2}\right) & \beta\left[\frac{3(N-1)}{2}\right] & \beta\left[\frac{5(N-1)}{2}\right] & \beta\left[\frac{7(N-1)}{2}\right] & \dots & \beta\left[\frac{(N-1)(2N-1)}{2}\right] \end{bmatrix} \tag{3.11}$$

Subsequently, the inverse ONMNT (IONMNT) is derived by replacing n from the NMNT with $\frac{2n+1}{2}$ while retaining the original k term so that the inverse transform is a transpose of the forward transform. This inverse transform of the ONMNT can formally be defined as

$$x(n) = \left\langle N^{-1} \sum_{n=0}^{N-1} X_O(k) \beta\left(\frac{2n+1}{2}k\right) \right\rangle_{Mp} \quad \text{for } n = 0, 1, 2, ..., N-1. \tag{3.12}$$

Table 3.3: The IONMNT Matrix for $N = 16$ and $Mp = 127$

1	72	82	19	111	19	82	72	1	5	3	99	0	28	124	122
1	19	82	5	0	122	45	108	126	28	3	72	111	72	3	28
1	19	3	122	16	122	3	19	1	28	45	55	0	72	82	99
1	72	124	108	0	19	3	55	126	5	82	28	16	28	82	5
1	5	45	28	111	28	45	5	1	55	124	19	0	108	3	72
1	99	45	72	0	55	82	28	126	19	124	122	111	122	124	19
1	28	124	72	16	72	124	28	1	108	82	122	0	5	45	19
1	122	3	28	0	99	124	5	126	72	45	19	16	19	45	72
1	55	82	108	111	108	82	55	1	122	3	28	0	99	124	5
1	108	82	122	0	5	45	19	126	99	3	55	111	55	3	99
1	108	3	5	16	5	3	108	1	99	45	72	0	55	82	28
1	55	124	19	0	108	3	72	126	122	82	99	16	99	82	122
1	122	45	99	111	99	45	122	1	72	124	108	0	19	3	55
1	28	45	55	0	72	82	99	126	108	124	5	111	5	124	108
1	99	124	55	16	55	124	99	1	19	82	5	0	122	45	108
1	5	3	99	0	28	124	122	126	55	45	108	16	108	45	55

As expected, this transform produces a matrix that is the transpose of the forward transform kernel. This produces a kernel that looks like

$$M^{-1} = \begin{bmatrix} \beta(0) & \beta\left(\frac{1}{2}\right) & \beta(1) & \beta\left(\frac{3}{2}\right) & \dots & \beta\left(\frac{N-1}{2}\right) \\ \beta(0) & \beta\left(\frac{3}{2}\right) & \beta(3) & \beta\left(\frac{9}{2}\right) & \dots & \beta\left[\frac{3(N-1)}{2}\right] \\ \beta(0) & \beta\left(\frac{5}{2}\right) & \beta(5) & \beta\left(\frac{15}{2}\right) & \dots & \beta\left[\frac{5(N-1)}{2}\right] \\ \beta(0) & \beta\left(\frac{7}{2}\right) & \beta(7) & \beta\left(\frac{21}{2}\right) & \dots & \beta\left[\frac{7(N-1)}{2}\right] \\ \beta(0) & \beta\left(\frac{9}{2}\right) & \beta(9) & \beta\left(\frac{27}{2}\right) & \dots & \beta\left[\frac{9(N-1)}{2}\right] \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \beta(0) & \beta\left(\frac{2N-1}{2}\right) & \beta(2N-1) & \beta\left[\frac{3(2N-1)}{2}\right] & \dots & \beta\left[\frac{(N-1)(2N-1)}{2}\right] \end{bmatrix} \tag{3.13}$$

$$\left\langle \mathcal{G}_O \mathcal{G}_O^T N^{-1} \right\rangle_{Mp} = I \tag{3.14}$$

where $\mathcal{G}_O$ is an ONMNT transform matrix of length N, $\mathcal{G}_O^T$ is the transpose of the ONMNT transform matrix, N^{-1} is the corresponding scaling factor and I is the identity matrix. Examples of the ONMNT and IONMNT transform matrices are shown in Tables 3.2 and 3.3 respectively using $N = 16$ and $p = 7$.

3.4 Odd-Squared-NMNT (O^{2}NMNT)

The O^{2}NMNT is the third and final new transform of the GNMNT. The O^{2}NMNT is unique to the GNMNT in that its construction contains no trivial elements that are present in the NMNT, ONMNT and the IONMNT. The details and the implications of these characteristics will be discussed further in Section 3.8.1. The derivation of the O^{2}NMNT is

$$X_{O^2}(k) = \left\langle \sum_{n=0}^{N-1} x(n) \beta \left[\frac{(2n+1)(2k+1)}{4} \right] \right\rangle_{Mp} \quad \text{for } k = 0, 1, 2, ..., N-1 \tag{3.15}$$

and the inverse is defined as

$$x(n) = \left\langle N^{-1} \sum_{k=0}^{N-1} X_{O^2}(k) \beta \left[\frac{(2n+1)(2k+1)}{4} \right] \right\rangle_{Mp} \quad \text{for } n = 0, 1, 2, ..., N-1. \tag{3.16}$$

Like the NMNT, the O^{2}NMNT shares a similar advantage in that the inverse transform is the same as the forward transform, meaning that as well as being orthogonal, their kernels are also symmetrical. In addition to k being replaced by $\frac{2k+1}{2}$, so n is also replaced by $\frac{2n+1}{2}$, which when combined produces $\frac{(2k+1)(2n+1)}{4}$. An example of the O^{2}NMNT kernel is provided in Table 3.4 where $N = 16$ and $p = 7$.

Table 3.4: The O²NMNTKernel for $N = 16$ and $Mp = 127$

15	56	106	105	105	106	56	15	83	26	14	36	91	113	101	44
56	105	15	14	113	112	22	71	101	36	83	106	106	83	36	101
106	15	91	71	71	91	15	106	14	44	22	101	26	105	83	113
105	14	71	44	83	56	113	22	91	106	26	112	112	26	106	91
105	113	71	83	83	71	113	105	91	21	26	15	112	101	106	36
106	112	91	56	71	36	15	21	14	83	22	26	26	22	83	14
56	22	15	113	113	15	22	56	101	91	83	21	106	44	36	26
15	71	106	22	105	21	56	112	83	101	14	91	91	14	101	83
83	101	14	91	91	14	101	83	112	56	21	105	22	106	71	15
26	36	44	106	21	83	91	101	56	22	15	113	113	15	22	56
14	83	22	26	26	22	83	14	21	15	36	71	56	91	112	106
36	106	101	112	15	26	21	91	105	113	71	83	83	71	113	105
91	106	26	112	112	26	106	91	22	113	56	83	44	71	14	105
113	83	105	26	101	22	44	14	106	15	91	71	71	91	15	106
101	36	83	106	106	83	36	101	71	22	112	113	14	15	105	56
44	101	113	91	36	14	26	83	15	56	106	105	105	106	56	15

Table 3.5: Values of α_1 and α_2 According to p for NMNT

p	α_1	α_2
3	4	2
5	8	20
7	16	88
13	128	181
17	512	87260
19	1024	385302
31	65536	1268011823

As the O²NMNTis symmetrical, like the NMNT, the forward and inverse kernels are derived as

$$M = \begin{bmatrix} \beta\left(\frac{1}{4}\right) & \beta\left(\frac{3}{4}\right) & \beta\left(\frac{5}{4}\right) & \beta\left(\frac{7}{4}\right) & \cdots & \beta\left(\frac{2N-1}{4}\right) \\ \beta\left(\frac{3}{4}\right) & \beta\left(\frac{9}{4}\right) & \beta\left(\frac{15}{4}\right) & \beta\left(\frac{21}{4}\right) & \cdots & \beta\left[\frac{3(2N-1)}{4}\right] \\ \beta\left(\frac{5}{4}\right) & \beta\left(\frac{15}{4}\right) & \beta\left(\frac{25}{4}\right) & \beta\left(\frac{35}{4}\right) & \cdots & \beta\left[\frac{5(2N-1)}{4}\right] \\ \beta\left(\frac{7}{4}\right) & \beta\left(\frac{21}{4}\right) & \beta\left(\frac{35}{4}\right) & \beta\left(\frac{49}{4}\right) & \cdots & \beta\left[\frac{7(2N-1)}{4}\right] \\ \beta\left(\frac{9}{4}\right) & \beta\left(\frac{27}{4}\right) & \beta\left(\frac{45}{4}\right) & \beta\left(\frac{63}{4}\right) & \cdots & \beta\left[\frac{9(2N-1)}{4}\right] \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \beta\left(\frac{2N-1}{4}\right) & \beta\left[\frac{3(2N-1)}{4}\right] & \beta\left[\frac{5(2N-1)}{4}\right] & \beta\left[\frac{7(2N-1)}{4}\right] & \cdots & \beta\left[\frac{(2N-1)(2N-1)}{4}\right] \end{bmatrix} \tag{3.17}$$

Table 3.6: Values of α_1 and α_2 According to p and N for NMNT

	$p=3$				$p=5$				$p=7$			
N	α_1	$\overline{\alpha_1}$	α_2	$\overline{\alpha_2}$	α_1	$\overline{\alpha_1}$	α_2	$\overline{\alpha_2}$	α_1	$\overline{\alpha_1}$	α_2	$\overline{\alpha_2}$
2	6	-1	0	-7	30	-1	0	-31	126	-1	0	-127
4	0	-7	6	-1	0	-31	30	-1	0	-127	1	-126
8	5	-2	2	-5	27	-4	4	-27	119	-8	119	-8
16	·	·	·	·	18	-13	7	-24	106	-21	103	-24
32	·	·	·	·	5	-26	10	-21	102	-25	97	-30
64	·	·	·	·	·	·	·	·	49	-78	93	-34
128	·	·	·	·	·	·	·	·	5	-122	22	-105
256	·	·	·	·	·	·	·	·	·	·	·	·
512	·	·	·	·	·	·	·	·	·	·	·	·
1024	·	·	·	·	·	·	·	·	·	·	·	·
2048	·	·	·	·	·	·	·	·	·	·	·	·
4096	·	·	·	·	·	·	·	·	·	·	·	·
8192	·	·	·	·	·	·	·	·	·	·	·	·
16384	·	·	·	·	·	·	·	·	·	·	·	·
32768	·	·	·	·	·	·	·	·	·	·	·	·
65536	·	·	·	·	·	·	·	·	·	·	·	·
131072	·	·	·	·	·	·	·	·	·	·	·	·
262144	·	·	·	·	·	·	·	·	·	·	·	·
524288	·	·	·	·	·	·	·	·	·	·	·	·
1048576	·	·	·	·	·	·	·	·	·	·	·	·
2097152	·	·	·	·	·	·	·	·	·	·	·	·
4194304	·	·	·	·	·	·	·	·	·	·	·	·
8388608	·	·	·	·	·	·	·	·	·	·	·	·
16777216	·	·	·	·	·	·	·	·	·	·	·	·
33554432	·	·	·	·	·	·	·	·	·	·	·	·
67108864	·	·	·	·	·	·	·	·	·	·	·	·
134217728	·	·	·	·	·	·	·	·	·	·	·	·
268435456	·	·	·	·	·	·	·	·	·	·	·	·
536870912	·	·	·	·	·	·	·	·	·	·	·	·
1073741824	·	·	·	·	·	·	·	·	·	·	·	·
2147483648	·	·	·	·	·	·	·	·	·	·	·	·

	$p=13$				$p=17$			
N	α_1	$\overline{\alpha_1}$	α_2	$\overline{\alpha_2}$	α_1	$\overline{\alpha_1}$	α_2	$\overline{\alpha_2}$
2	8190	-1	0	-8191	131070	-1	0	-131071
4	0	-8191	8190	-1	0	-131071	131070	-1
8	64	-8127	8127	-64	256	-130815	130815	-256
16	6456	-1735	7379	-812	55712	-75359	104646	-26425
32	5114	-3077	647	-7544	125656	-5415	70500	-60571
64	2498	-5693	3389	-4802	69447	-61624	29296	-101775
128	4634	-3557	2338	-5853	77543	-53528	19726	-111345
256	336	-7855	1198	-6993	94715	-36356	57903	-73168
512	5152	-3039	4412	-3779	90069	-41002	83147	-47924
1024	253	-7938	2016	-6175	94892	-36179	42772	-88299
2048	4801	-3390	6567	-1624	125429	-5642	41255	-89816
4096	49	-8142	4664	-3527	72359	-58712	108667	-22404
8192	5	-8186	5381	-2810	20129	-110942	45131	-85940
16384	·	·	·	·	93280	-37791	36519	-94552
32768	·	·	·	·	4801	-126270	61881	-69190
65536	·	·	·	·	49	-131022	31393	-99678
131072	·	·	·	·	5	-131066	94889	-36182
262144	·	·	·	·	·	·	·	·
524288	·	·	·	·	·	·	·	·
1048576	·	·	·	·	·	·	·	·
2097152	·	·	·	·	·	·	·	·
4194304	·	·	·	·	·	·	·	·
8388608	·	·	·	·	·	·	·	·
16777216	·	·	·	·	·	·	·	·
33554432	·	·	·	·	·	·	·	·
67108864	·	·	·	·	·	·	·	·
134217728	·	·	·	·	·	·	·	·
268435456	·	·	·	·	·	·	·	·
536870912	·	·	·	·	·	·	·	·
1073741824	·	·	·	·	·	·	·	·
2147483648	·	·	·	·	·	·	·	·

	$p=19$				$p=31$			
N	α_1	$\overline{\alpha_1}$	α_2	$\overline{\alpha_2}$	α_1	$\overline{\alpha_1}$	α_2	$\overline{\alpha_2}$
2	524286	-1	0	-524287	2147483646	-1	0	-2147483647
4	0	-524287	1	-524286	0	-2147483647	1	-2147483646
8	512	-523775	512	-523775	32768	-2147450879	32768	-2147450879
16	361783	-162504	481674	-42613	1556715293	-590768354	978592373	-1168891274
32	326972	-197315	586	-523701	1241207368	-906276279	1179735656	-967747991
64	143019	-381268	88897	-435390	1641940819	-505542828	26164677	-2121318970
128	337203	-187084	518064	-6223	206059115	-1941424532	1935040570	-212443077
256	439100	-85187	63598	-460689	236104903	-1911378744	1577470940	-570012707
512	431498	-92789	331707	-192580	430821412	-1716662235	1152650470	-994833177
1024	141009	-383278	49967	-474320	2118812954	-28670693	1778905319	-368578328
2048	156575	-367712	75021	-449266	1007165928	-1140317719	96763266	-2050720381
4096	206207	-318080	416887	-107400	1860715661	-286767986	513820856	-1633662791
8192	342493	-181794	291292	-232995	503566987	-1643916660	691000908	-1456482739
16384	100120	-424167	341961	-182326	1733763612	-413720035	1375762489	-771721158
32768	204661	-319626	284580	-239707	1556415982	-591067665	1095657803	-1051825844
65536	486232	-38055	463113	-61174	2011288684	-136194963	1058967642	-1088516005
131072	4801	-519486	16811	-507476	455787664	-1691695983	555296581	-1592187066
262144	49	-524238	465610	-58677	758674321	-1388809326	160114785	-1987368862
524288	5	-524282	46561	-477726	763747008	-1383736639	1297503800	-849979847
1048576	·	·	·	·	1714604740	-432878907	1889070037	-258413610
2097152	·	·	·	·	1930613611	-216870036	768607679	-1378875968
4194304	·	·	·	·	1281838047	-865645600	186019763	-1961463884
8388608	·	·	·	·	1705978478	-441505169	781338124	-1366145523
16777216	·	·	·	·	92950461	-2054533186	1574605965	-572877682
33554432	·	·	·	·	329019877	-1818463770	1113832336	-1033651311
67108864	·	·	·	·	1975567928	-171915719	27121372	-2120362275
134217728	·	·	·	·	948821812	-1198661835	1003062899	-1144420748
268435456	·	·	·	·	46099201	-2101384446	1943377242	-204106405
536870912	·	·	·	·	4801	-2147478846	1683836729	-463646918
1073741824	·	·	·	·	49	-2147483598	2142752556	-4731091
2147483648	·	·	·	·	5	-2147483642	643771985	-1503711662

Table 3.7: Values of α_1 and α_2 According to p and N for ONMNT

N	$p=3$			$p=5$			$p=7$			$p=13$			$p=17$			$p=19$			$p=31$		
	α_1	$\overline{\alpha_1}$	α_2	$\overline{\alpha_2}$	α_1	$\overline{\alpha_1}$	α_2	$\overline{\alpha_2}$	α_1	$\overline{\alpha_1}$	α_2	$\overline{\alpha_2}$	α_1	$\overline{\alpha_1}$	α_2	$\overline{\alpha_2}$	α_1	$\overline{\alpha_1}$	α_2	$\overline{\alpha_2}$
2	0	-7	6	-1	0	-31	30	-1	0	-127	1	-126	0	-8191	8190	-1	0	-131071	131070	-1	0	-524287	1	-524286	0	-2147483647	1	-2147483646
4	5	-2	2	-5	27	-4	4	-27	119	-8	119	-8	64	-8127	8127	-64	256	-130815	130815	-256	512	-523775	512	-523775	32768	-2147450879	32768	-2147450879
8	.	.	.	.	18	-13	7	-24	106	-21	103	-24	6456	-1735	7379	-812	55712	-75359	104646	-26425	361783	-162504	481674	-42613	1556715293	-590768354	978592373	-1168891274
16	.	.	.	.	5	-26	10	-21	102	-25	97	-30	5114	-3077	647	-7544	125656	-5415	70500	-60571	326972	-197315	586	-523701	1241207368	-906276279	1179735656	-967747991
32	.	.	.	.	.	.	.	.	49	-78	93	-34	2498	-5693	3389	-4802	69447	-61624	29296	-101775	143019	-381268	88897	-435390	1641940819	-505542828	26164677	-2121318970
64	.	.	.	.	.	.	.	.	5	-122	22	-105	4634	-3557	2338	-5853	77543	-53528	19726	-111345	337203	-187084	518064	-6223	206059115	-1941424532	193504570	-212443077
128	.	.	.	.	.	.	.	.	.	.	.	.	336	-7855	1198	-6993	94715	-36356	57903	-73168	439100	-85187	63598	-460689	236104903	-1911378744	1577470940	-570012707
256	.	.	.	.	.	.	.	.	.	.	.	.	5152	-3039	4412	-3779	90069	-41002	83147	-47924	431498	-92789	331707	-192580	430821412	-1716662235	1152650470	-994833177
512	.	.	.	.	.	.	.	.	.	.	.	.	253	-7938	2016	-6175	94892	-36179	42772	-88299	141009	-383278	49967	-474320	2118812954	-28670693	1778905319	-368578328
1024	.	.	.	.	.	.	.	.	.	.	.	.	4801	-3390	6567	-1624	125429	-5642	41255	-89816	156575	-367712	75021	-449266	1007165928	-1140317719	96763266	-2050720381
2048	.	.	.	.	.	.	.	.	.	.	.	.	49	-8142	4664	-3527	72359	-58712	108667	-22404	206207	-318080	416887	-107400	1860715661	-286767986	513820856	-1633662791
4096	.	.	.	.	.	.	.	.	.	.	.	.	5	-8186	5381	-2810	20129	-110942	45131	-85940	342493	-181794	291292	-232995	503566987	-1643916660	691000908	-1456482739
8192	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	93280	-37791	36519	-94552	100120	-424167	341961	-182326	1733763612	-413720035	1375762489	-771721158
16384	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	4801	-126270	61881	-69190	204661	-319626	284580	-239707	1556415982	-591067665	1095657803	-1051825844
32768	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	49	-131022	31393	-99678	486232	-38055	463113	-61174	2011288684	-136194963	1058967642	-1088516005
65536	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	5	-131066	94889	-36182	4801	-519486	16811	-507476	455787664	-1691695983	555296581	-1592187066
131072	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	49	-524238	465610	-58677	758674321	-1388809326	160114785	-1987368862
262144	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	5	-524282	46561	-477726	763747008	-1383736639	1297503800	-849979847
524288	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1714604740	-432878907	1889070037	-258413610
1048576	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1930613611	-216870036	768607679	-1378875968
2097152	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1281838047	-865645600	186019763	-1961463884
4194304	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1705978478	-441505169	781338124	-1366145523
8388608	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	92950461	-2054533186	1574605965	-572877682
16777216	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	329019877	-1818463770	1113832336	-1033651311
33554432	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1975567928	-171915719	27121372	-2120362275
67108864	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	948821812	-1198661835	1003062899	-1144420748
134217728	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	46099201	-2101384446	1943377242	-204106405
268435456	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	4801	-2147478846	1683836729	-463646918
536870912	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	49	-2147483598	2142752556	-4731091
1073741824	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	5	-2147483642	643771985	-1503711662

Table 3.8: Values of α_1 and α_2 According to p and N for O^{2}NMNT

	$p=3$				$p=5$				$p=7$				$p=13$			
N	α_1	$\overline{\alpha_1}$	α_2	$\overline{\alpha_2}$	α_1	$\overline{\alpha_1}$	α_2	$\overline{\alpha_2}$	α_1	$\overline{\alpha_1}$	α_2	$\overline{\alpha_2}$	α_1	$\overline{\alpha_1}$	α_2	$\overline{\alpha_2}$
2	5	-2	2	-5	27	-4	4	-27	119	-8	119	-8	64	-8127	8127	-64
4	·	·	·	·	18	-13	7	-24	106	-21	103	-24	6456	-1735	7379	-812
8	·	·	·	·	5	-26	10	-21	102	-25	97	-30	5114	-3077	647	-7544
16	·	·	·	·	·	·	·	·	49	-78	93	-34	2498	-5693	3389	-4802
32	·	·	·	·	·	·	·	·	5	-122	22	-105	4634	-3557	2338	-5853
64	·	·	·	·	·	·	·	·	·	·	·	·	336	-7855	1198	-6993
128	·	·	·	·	·	·	·	·	·	·	·	·	5152	-3039	4412	-3779
256	·	·	·	·	·	·	·	·	·	·	·	·	253	-7938	2016	-6175
512	·	·	·	·	·	·	·	·	·	·	·	·	4801	-3390	6567	-1624
1024	·	·	·	·	·	·	·	·	·	·	·	·	49	-8142	4664	-3527
2048	·	·	·	·	·	·	·	·	·	·	·	·	5	-8186	5381	-2810
4096	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
8192	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
16384	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
32768	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
65536	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
131072	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
262144	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
524288	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
1048576	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
2097152	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
4194304	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
8388608	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
16777216	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
33554432	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
67108864	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
134217728	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
268435456	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
536870912	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·

	$p=17$				$p=19$				$p=31$			
N	α_1	$\overline{\alpha_1}$	α_2	$\overline{\alpha_2}$	α_1	$\overline{\alpha_1}$	α_2	$\overline{\alpha_2}$	α_1	$\overline{\alpha_1}$	α_2	$\overline{\alpha_2}$
2	256	-130815	130815	-256	512	-523775	512	-523775	32768	-2147450879	32768	-2147450879
4	55712	-75359	104646	-26425	361783	-162504	481674	-42613	1556715293	-590768354	978592373	-1168891274
8	125656	-5415	70500	-60571	326972	-197315	586	-523701	1241207368	-906276279	1179735656	-967747991
16	69447	-61624	29296	-101775	143019	-381268	88897	-435390	1641940819	-505542828	26164677	-2121318970
32	77543	-53528	19726	-111345	337203	-187084	518064	-6223	206059115	-1941424532	1935040570	-212443077
64	94715	-36356	57903	-73168	439100	-85187	63598	-460689	236104903	-1911378744	1577470940	-570012707
128	90069	-41002	83147	-47924	431498	-92789	331707	-192580	430821412	-1716662235	1152650470	-994833177
256	94892	-36179	42772	-88299	141009	-383278	49967	-474320	2118812954	-28670693	1778905319	-368578328
512	125429	-5642	41255	-89816	156575	-367712	75021	-449266	1007165928	-1140317719	96763266	-2050720381
1024	72359	-58712	108667	-22404	206207	-318080	416887	-107400	1860715661	-286767986	513820856	-1633662791
2048	20129	-110942	45131	-85940	342493	-181794	291292	-232995	503566987	-1643916660	691000908	-1456482739
4096	93280	-37791	36519	-94552	100120	-424167	341961	-182326	1733763612	-413720035	1375762489	-771721158
8192	4801	-126270	61881	-69190	204661	-319626	284580	-239707	1556415982	-591067665	1095657803	-1051825844
16384	49	-131022	31393	-99678	486232	-38055	463113	-61174	2011288684	-136194963	1058967642	-1088516005
32768	5	-131066	94889	-36182	4801	-519486	16811	-507476	455787664	-1691695983	555296581	-1592187066
65536	·	·	·	·	49	-524238	465610	-58677	758674321	-1388809326	160114785	-1987368862
131072	·	·	·	·	5	-524282	46561	-477726	763747008	-1383736639	1297503800	-849979847
262144	·	·	·	·	·	·	·	·	1714604740	-432878907	1889070037	-258413610
524288	·	·	·	·	·	·	·	·	1930613611	-216870036	768607679	-1378875968
1048576	·	·	·	·	·	·	·	·	1281838047	-865645600	186019763	-1961463884
2097152	·	·	·	·	·	·	·	·	1705978478	-441505169	781338124	-1366145523
4194304	·	·	·	·	·	·	·	·	92950461	-2054533186	1574605965	-572877682
8388608	·	·	·	·	·	·	·	·	329019877	-1818463770	1113832336	-1033651311
16777216	·	·	·	·	·	·	·	·	1975567928	-171915719	27121372	-2120362275
33554432	·	·	·	·	·	·	·	·	948821812	-1198661835	1003062899	-1144420748
67108864	·	·	·	·	·	·	·	·	46099201	-2101384446	1943377242	-204106405
134217728	·	·	·	·	·	·	·	·	4801	-2147478846	1683836729	-463646918
268435456	·	·	·	·	·	·	·	·	49	-2147483598	2142752556	-4731091
536870912	·	·	·	·	·	·	·	·	5	-2147483642	643771985	-1503711662

Table 3.9: GNMNT Transform Selection

$\mathcal{G}$	n_0	k_0
NMNT	0	0
ONMNT	0	1
IONMNT	1	0
O^{2}NMNT	1	1

3.5 Derivation of Transform Parameters

This section outlays the taxonomy of the GNMNT and demonstrates how the transform parameters are derived for the NMNT, ONMNT and O^{2}NMNT, including consideration of their respective maximal transform lengths. For all transforms of the GNMNT, the initial values of α_1 and α_2 are calculated by selecting the appropriate value of q according to p for (3.4). The values that are subsequently obtained with respect to p are shown in Table 3.5.

With appropriate starting values selected for of α_1 and α_2, they both require adjustment according to the transform length to produce the initial values of β_1 and β_2. This is calculated as shown in (3.5) and (3.6) accordingly. The first notable difference between the GNMNTs is through the derivation of N according to the length $\frac{N}{d}$ for d is a power of two, which is 2^{p+1} for NMNT, 2^p for ONMNT and 2^{p-1} for O^{2}NMNT. A comprehensive list of final values of α_1 and α_2 according to the field derived by p and the transform length N for the NMNT, ONMNT and O^{2}NMNTis shown in Tables 3.6, 3.7 and 3.8 respectively. The taxonomy of the GNMNT that was first shown in Figure 1.1 illustrates how all of the GNMNT transforms are interlinked with each other. These transforms are generalised as

$$X(k) = \left\langle \sum_{n=0}^{N-1} x\left(n\right) \beta \left[\frac{\left(2n + n_0\right)\left(2k + k_0\right)}{4} \right] \right\rangle_{Mp} \quad \text{for } k = 0, 1, ..., N-1 \tag{3.18}$$

and

$$x(n) = \left\langle N^{-1} \sum_{k=0}^{N-1} X\left(k\right) \beta \left[\frac{\left(2n + n_0\right)\left(2k + k_0\right)}{4} \right] \right\rangle_{Mp} \quad \text{for } n = 0, 1, ..., N-1 \tag{3.19}$$

where the variables n_0 and k_0 are selected to denote which transform is to be used from the GNMNT according to the configuration shown in Table 3.9, which is described in more detail in [11]. All transforms are orthogonal and when $n_0 = k_0$ when either the NMNT or O^{2}NMNT are selected then the transforms are also

symmetrical.

3.6 Cyclic Convolution of the GNMNT

Convolution is widely used in signal processing and image processing. There are three types of convolution that have been defined, which are linear convolution, cyclic convolution and the skew-cyclic convolution [116, 117]. When the input sequence is $x(n)$ and the impulse response is $h(n)$, which has the same length N, the length of the linear convolution output $y_{LC}(n)$ is $2N-1$ [116, 117]. The output of the linear convolution is shown as

$$\begin{aligned} y_{LC}(n) &= x(n) \circledast h(n) \\ &= \sum_{k=0}^{N-1} x(k)\, h(n-k) \\ &= \sum_{k=0}^{N-1} x(n-k)\, h(k) \end{aligned} \tag{3.20}$$

where $\circledast$ denotes convolution and the length of the cyclic convolution output $y_{CC}(n)$ is N shown as

$$\begin{aligned} y_{CC}(n) &= x(n) \circledast h(n) \\ &= \sum_{k=0}^{N-1} x(k)\, h(n-k \bmod N) \\ &= \sum_{k=0}^{N-1} x(n-k \bmod N)\, h(k) \\ &= \sum_{k=0}^{n} x(k) h(n-k) + \sum_{k=n+1}^{N-1} x(k) h(n-k+N). \end{aligned} \tag{3.21}$$

Finally, the skew-cyclic convolution output $y_{SCC}(n)$ is shown as

$$\begin{aligned} y_{SCC}(n) &= x(n) \circledast h(n) \\ &= \sum_{k=0}^{n} x(k)\, h(n-k) - \sum_{k=n+1}^{N-1} x(k)\, h(n-k+N). \end{aligned} \tag{3.22}$$

3.6.1 Cyclic convolution of NMNT

As shown in (3.20) and (3.21), the output lengths are different. However, the cyclic convolution can be derived from the linear convolution by padding with zeros so that it can reach the required length of $2N-1$. The cyclic convolution can

therefore be calculated by

$$\begin{aligned}\mathcal{G}_N\left[y_{CC}(n)\right] &= \mathcal{G}_N\left[x\left(n\right) \circledast h\left(n\right)\right] \\ &= \left\langle \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} x(n-m)h\left(m\right)\beta\left(nk\right) \right\rangle_{Mp} \\ &= \left\langle \sum_{m=0}^{N-1} h\left(m\right)\sum_{s=0}^{N-1} x(s)\beta\left(sk+mk\right) \right\rangle_{Mp}\end{aligned} \tag{3.23}$$

where $s = m - n$, and $\mathcal{G}_N$ denotes the NMNT transform. According to the NMNT definition, $\beta(sk + mk)$ term in (3.23) can be simplified as

$$\begin{aligned}&\left\langle \sum_{m=0}^{N-1} h\left(m\right)\sum_{s=0}^{N-1} x(s)\left[\beta_1\left(mk\right)\beta\left(sk\right) + \beta_2\left(mk\right)\beta\left(-sk\right)\right] \right\rangle_{Mp} \\ &\quad= \left\langle \sum_{m=0}^{N-1} h\left(m\right)\beta_1\left(mk\right)\sum_{s=0}^{N-1} x(s)\beta\left(sk\right) \right. \\ &\qquad\left. + \sum_{m=0}^{N-1} h\left(m\right)\beta_2\left(mk\right)\sum_{s=0}^{N-1} x(s)\beta\left(-sk\right) \right\rangle_{Mp}.\end{aligned} \tag{3.24}$$

Due to the relationships between $\beta_1(n)$ and $\beta(n)$, $\beta_2(n)$ and $\beta(n)$ where

$$\beta_1(n) = \left\langle \frac{1}{2}\left[\beta(n) + \beta(-n)\right] \right\rangle_{Mp} \tag{3.25}$$

and

$$\beta_2(n) = \left\langle \frac{1}{2}\left[\beta(n) - \beta(-n)\right] \right\rangle_{Mp} \tag{3.26}$$

then applying (3.25) and (3.26) to (3.24) produces

$$\begin{aligned}&\left\langle \sum_{m=0}^{N-1} h\left(m\right)\beta_1\left(mk\right)\sum_{s=0}^{N-1} x(s)\beta\left(sk\right) + \sum_{m=0}^{N-1} h\left(m\right)\beta_2\left(mk\right)\sum_{s=0}^{N-1} x(s)\beta\left(-sk\right) \right\rangle_{Mp} \\ &= \left\langle \sum_{m=0}^{N-1} h\left(m\right)\left[\frac{\beta\left(mk\right)+\beta\left(-mk\right)}{2}\right]X\left(k\right) \right. \\ &\quad\left. + \sum_{m=0}^{N-1} h\left(m\right)\left[\frac{\beta\left(mk\right)-\beta\left(-mk\right)}{2}\right]X\left(N-k\right) \right\rangle_{Mp}\end{aligned} \tag{3.27}$$

where $X(k)$ and $X(N - k)$ are defined by NMNT property, as well as $H(k)$ and $H(N - k)$, which are

$$\left\langle \sum_{s=0}^{N-1} x(s)\beta\left(sk\right) \right\rangle_{Mp} = X\left(k\right), \tag{3.28}$$

$$\begin{aligned}\left\langle \sum_{s=0}^{N-1} x(s)\beta\left(-sk\right) \right\rangle_{Mp} &= X\left(-k\right) \\ &= X\left(N-k\right)\end{aligned} \tag{3.29}$$

and

$$\left\langle \sum_{m=0}^{N-1} h\left(m\right)\beta\left(mk\right) \right\rangle_{Mp} = H\left(k\right), \tag{3.30}$$

$$\begin{aligned}\left\langle \sum_{m=0}^{N-1} h\left(m\right)\beta\left(-mk\right) \right\rangle_{Mp} &= H\left(-k\right) \\ &= H\left(N-k\right).\end{aligned} \tag{3.31}$$

Therefore, applying (3.30) and (3.31) into (3.27) produces

$$\begin{aligned}&\left\langle \frac{H\left(k\right)+H\left(-k\right)}{2}X\left(k\right)+\frac{H\left(k\right)-H\left(-k\right)}{2}X\left(N-k\right) \right\rangle_{Mp} \\ &= \left\langle \left\{ \left[H\left(k\right)+H\left(N-k\right)\right]X\left(k\right)+\left[H\left(k\right)-H\left(N-k\right)\right]X\left(N-k\right) \right\} 2^{p-1} \right\rangle_{Mp} \\ &= Y\left(k\right).\end{aligned} \tag{3.32}$$

where

$$\frac{1}{2} = \left\langle\, 2^{p-1} \right\rangle_{Mp}. \tag{3.33}$$

Splitting $H\left(k\right)$ into even and odd parts as

$$\begin{aligned}H^{ev}(k) &= \left\langle \frac{H\left(k\right)+H\left(-k\right)}{2} \right\rangle_{Mp} \\ &= \left\langle \left[H\left(k\right)+H\left(-k\right)\right]2^{p-1} \right\rangle_{Mp}\end{aligned} \tag{3.34}$$

and

$$\begin{aligned}H^{od}(k) &= \left\langle \frac{H\left(k\right)-H\left(-k\right)}{2} \right\rangle_{Mp} \\ &= \left\langle \left[H\left(k\right)-H\left(-k\right)\right]2^{p-1} \right\rangle_{Mp}\end{aligned} \tag{3.35}$$

Then applying (3.34) and (3.35) to (3.32) produces the desired equation in order to derive the cyclic convolution using the NMNT as

$$\begin{aligned}\mathcal{G}_N\left[y_{CC}(n)\right] &= \mathcal{G}_N\left[x\left(n\right) \circledast h\left(n\right)\right] \\ &= H^{ev}\left(k\right)X\left(k\right)+H^{od}\left(k\right)X\left(N-k\right) \\ &= Y\left(k\right).\end{aligned} \tag{3.36}$$

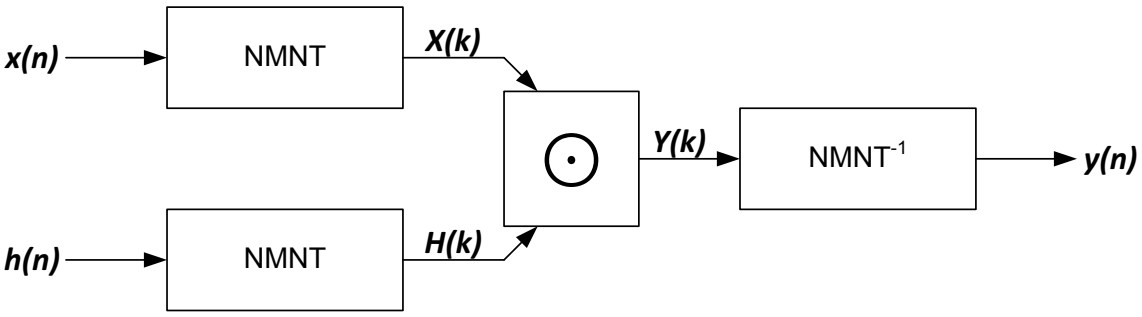


Figure 3.1: Convolution Process Structure for the NMNT

This is shown in Figure 3.1, where the $\odot$ operator is defined as

$$\begin{aligned} Y(k) &= X(k) \odot H(k) \\ &= H^{ev}(k) X(k) + H^{od}(k) X(N-k). \end{aligned} \tag{3.37}$$

An example using the NMNT to obtain the cyclic convolution y_{CC}, generating x and h randomly to represent the input and impulse response signals respectively, with $p = 13$ and length $N = 8$ is first shown as

$$x(n) = [1, 1, 0, 1, 1, 0, 0, 1] \tag{3.38}$$

and

$$h(n) = [1, 1, 0, 1, 1, 0, 1, 0]. \tag{3.39}$$

The resulting NMNT representations are

$$X(k) = [5, 923, 0, 6019, 3, 2547, 8063, 7492, 8190, 7268, 0, 2176, 1, 5644, 128, 703], \tag{3.40}$$

$$H(k) = [5, 8063, 8064, 5096, 1, 128, 8062, 1848, 1, 8063, 129, 3099, 1, 128, 127, 6347]. \tag{3.41}$$

Applying the cyclic convolution algorithm produces

$$Y(k) = [25, 4354, 126, 7994, 3, 4795, 0, 2271, 8190, 3315, 8061, 459, 1, 3898, 0, 5670], \tag{3.42}$$

and applying the inverse NMNT results the first $2N - 1$ elements as

$$ycc = [1, 2, 1, 2, 4, 2, 2, 4, 2, 1, 2, 1, 0, 1, 0]. \tag{3.43}$$

Confirming the results with the built in function of MATLAB produces

$$\text{conv(x, h)} = [1, 2, 1, 2, 4, 2, 2, 4, 2, 1, 2, 1, 0, 1, 0]. \tag{3.44}$$

3.6.2 Cyclic convolution of ONMNT

Following similar steps to derive the NMNT cyclic convolution computation, the ONMNT cyclic convolution can be calculated as

$$\begin{aligned}
\mathcal{G}_O\left[y_{CC}(n)\right] &= \mathcal{G}_O\left[x(n) \circledast h(n)\right] \\
&= \left\langle \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} x(n-m)h(m)\,\beta\left(\frac{2k+1}{2}n\right) \right\rangle_{Mp} \\
&= \left\langle \sum_{m=0}^{N-1} h(m) \sum_{s=0}^{N-1} x(s)\beta\left(\frac{2k+1}{2}s + \frac{2k+1}{2}m\right) \right\rangle_{Mp}
\end{aligned} \tag{3.45}$$

where $s = m - n$, and $\mathcal{G}_O$ denotes the ONMNT transform. From the β identities in provided in [7], the $\beta\left(\frac{2k+1}{2}s + \frac{2k+1}{2}m\right)$ term in (3.45) can be simplified as

$$\begin{aligned}
&\left\langle \sum_{m=0}^{N-1} h(m) \sum_{s=0}^{N-1} x(s)\left[\beta_1(mk)\,\beta\left(\frac{2k+1}{2}s\right) + \beta_2(mk)\,\beta\left(-\frac{2k+1}{2}s\right)\right] \right\rangle_{Mp} \\
&= \left\langle \sum_{m=0}^{N-1} h(m)\,\beta_1\left(\frac{2k+1}{2}m\right) \sum_{s=0}^{N-1} x(s)\beta\left(\frac{2k+1}{2}s\right) \right. \\
&\quad \left. + \sum_{m=0}^{N-1} h(m)\,\beta_2\left(\frac{2k+1}{2}m\right) \sum_{s=0}^{N-1} x(s)\beta\left(-\frac{2k+1}{2}s\right) \right\rangle_{Mp}.
\end{aligned} \tag{3.46}$$

Applying (3.25) and (3.26) now into (3.46), produces

$$\begin{aligned}
&\left\langle \sum_{m=0}^{N-1} h(m)\,\beta_1\left(\frac{2k+1}{2}m\right) \sum_{s=0}^{N-1} x(s)\beta\left(\frac{2k+1}{2}s\right) \right. \\
&\quad \left. + \sum_{m=0}^{N-1} h(m)\,\beta_2\left(\frac{2k+1}{2}m\right) \sum_{s=0}^{N-1} x(s)\beta\left(-\frac{2k+1}{2}s\right) \right\rangle_{Mp} \\
&= \left\langle \sum_{m=0}^{N-1} h(m) \left[\frac{\beta\left(\frac{2k+1}{2}m\right) + \beta\left(-\frac{2k+1}{2}m\right)}{2}\right] X_O(k) \right. \\
&\quad \left. + \sum_{m=0}^{N-1} h(m) \left[\frac{\beta\left(\frac{2k+1}{2}m\right) - \beta\left(-\frac{2k+1}{2}m\right)}{2}\right] X_O(N-k-1) \right\rangle_{Mp}
\end{aligned} \tag{3.47}$$

where $X_O(k)$ and $X_O(N-k)$ are defined by ONMNT variables, as are $H_O(k)$ and $H_O(N-k)$, which are

$$\left\langle \sum_{s=0}^{N-1} x(s)\beta\left(\frac{2k+1}{2}s\right)\right\rangle_{Mp} = X_O\left(k\right), \tag{3.48}$$

$$\begin{aligned}\left\langle \sum_{s=0}^{N-1} x(s)\beta\left(-\frac{2k+1}{2}s\right)\right\rangle_{Mp} &= X_O\left(-k-1\right)\\ &= X_O\left(N-k-1\right)\end{aligned} \tag{3.49}$$

and

$$\left\langle \sum_{m=0}^{N-1} h\left(m\right)\beta\left(\frac{2k+1}{2}m\right)\right\rangle_{Mp} = H_O\left(k\right), \tag{3.50}$$

$$\begin{aligned}\left\langle \sum_{m=0}^{N-1} h\left(m\right)\beta\left(-\frac{2k+1}{2}m\right)\right\rangle_{Mp} &= H_O\left(-k-1\right)\\ &= H_O\left(N-k-1\right).\end{aligned} \tag{3.51}$$

Thus, substituting (3.50) and (3.51) into (3.47) generates

$$\begin{aligned}&\left\langle \sum_{m=0}^{N-1} h\left(m\right)\left[\frac{\beta\left(\frac{2k+1}{2}m\right)+\beta\left(-\frac{2k+1}{2}m\right)}{2}\right]X_O\left(k\right)\right.\\ &\quad\left.+\sum_{m=0}^{N-1} h\left(m\right)\left[\frac{\beta\left(\frac{2k+1}{2}m\right)-\beta\left(-\frac{2k+1}{2}m\right)}{2}\right]X_O\left(N-k-1\right)\right\rangle_{Mp}\\ &=\left\langle \frac{H_O\left(k\right)+H_O\left(N-K-1\right)}{2}X_O\left(k\right)\right.\\ &\quad\left.+\frac{H_O\left(k\right)-H_O\left(N-k-1\right)}{2}X_O\left(N-k-1\right)\right\rangle_{Mp}\\ &=\left\langle \left\{ \left[H_O\left(k\right)+H_O\left(N-k-1\right)\right]X_O\left(k\right)\right.\right.\\ &\quad\left.\left.+\left[H_O\left(k\right)-H_O\left(N-k-1\right)\right]X_O\left(N-k-1\right)\right\}2^{p-1}\right\rangle_{Mp}\\ &=Y_O\left(k\right).\end{aligned} \tag{3.52}$$

Splitting $H_O\left(k\right)$ into even and odd parts produces

$$\begin{aligned}H_O^{ev}(k) &= \left\langle \frac{H_O\left(k\right)+H_O\left(-k\right)}{2}\right\rangle_{Mp}\\ &= \left\langle \left[H_O\left(k\right)+H_O\left(-k\right)\right]2^{p-1}\right\rangle_{Mp}\end{aligned} \tag{3.53}$$

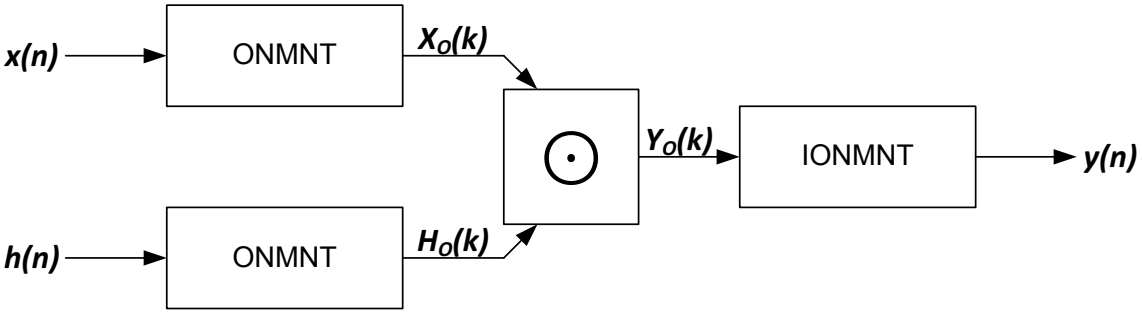


Figure 3.2: Convolution Process Structure for the ONMNT

and

$$\begin{aligned} H_O^{od}(k) &= \left\langle \frac{H_O(k) - H_O(-k)}{2} \right\rangle_{Mp} \\ &= \left\langle [H_O(k) - H_O(-k)]\, 2^{p-1} \right\rangle_{Mp}. \end{aligned} \tag{3.54}$$

Then applying (3.53) and (3.54) into (3.52) produces the desired equation to process the cyclic convolution using the ONMNT as

$$\begin{aligned} \mathcal{G}_O[y_{CC}(n)] &= \mathcal{G}_O[x(n) \circledast h(n)] \\ &= H_O^{ev}(k) X(k) + H_O^{od}(k) X(N-k) \\ &= Y_O(k). \end{aligned} \tag{3.55}$$

The $\odot$ operator in Figure 3.2 is defined as

$$\begin{aligned} Y_O(k) &= X_O(k) \odot H_O(k) \\ &= H_O^{ev}(k) X_O(k) + H_O^{od}(k) X_O(N-k). \end{aligned} \tag{3.56}$$

In Figure 3.2, the $\odot$ operator has the same function as it does in Figure 3.1. An example of a cyclic convolution using the ONMNT given where all of the variables that were used for the NMNT have the same purpose such that

$$x(n) = [1,1,1,1,0,0,0,0], \tag{3.57}$$

$$h(n) = [0,0,0,1,1,1,1,1]. \tag{3.58}$$

The resulting ONMNT representations are

$$\begin{aligned} X_O(k) = [&2309, 5366, 5554, 2448, 5584, 3903, 2196, 3490, \\ &790, 7921, 4485, 7591, 7703, 7387, 4151, 2857], \end{aligned} \tag{3.59}$$

and

$$H_O(k) = [4977, 770, 6362, 1669, 1920, 1344, 235, 4305, \\ 117, 4068, 8174, 433, 1177, 1497, 1611, 2296]. \tag{3.60}$$

Applying the circular convolution algorithm produces

$$Y_O(k) = [5515, 2526, 5538, 5146, 4799, 6275, 8100, 146, \\ 319, 7307, 357, 7850, 5737, 7453, 2375, 4276], \tag{3.61}$$

and applying the inverse ONMNT results in the first $2N-1$ elements as

$$ycc = [0, 0, 0, 1, 2, 3, 4, 4, 3, 2, 1, 0, 0, 0, 0]. \tag{3.62}$$

Confirming the results with the built in function of MATLAB produces

$$\text{conv(x, h)} = [0, 0, 0, 1, 2, 3, 4, 4, 3, 2, 1, 0, 0, 0, 0]. \tag{3.63}$$

3.6.3 Cyclic convolution of O^{2}NMNT

The O^{2}NMNT can also be applied to compute the cyclic convolution. As previously defined in (3.15), the O^{2}NMNT kernel matrix is derived from $\beta\left[\left(\frac{2k+1}{2}\right)\left(\frac{2n+1}{2}\right)\right]$ and is therefore implemented as

$$\begin{aligned} Y_{O^2}(k) &= \left\langle \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} x(n-m)h(m)\,\beta\left[\left(\frac{2k+1}{2}\right)\left(\frac{2n+1}{2}\right)\right]\right\rangle_{Mp} \\ &= \left\langle \sum_{m=0}^{N-1} h(m) \sum_{s=0}^{N-1} x(s)\beta\left[\left(\frac{2k+1}{2}\right)\left(\frac{2n+1}{2}\right)\right]\right\rangle_{Mp} \end{aligned} \tag{3.64}$$

where

$$s = m - n. \tag{3.65}$$

The O^{2}NMNT defines $H_{O^2}(k)$ and $H_{O^2}(N-k-1)$ by

$$\left\langle \sum_{m=0}^{N-1} h(m)\beta\left[\left(\frac{2k+1}{2}\right)\left(\frac{2m+1}{2}\right)\right]\right\rangle_{Mp} = H_{O^2}(k) \tag{3.66}$$

and

$$\begin{aligned}\left\langle \sum_{m=0}^{N-1} h(m)\beta\left[\left(-\frac{2k+1}{2}\right)\left(\frac{2m+1}{2}\right)\right]\right\rangle_{Mp} &= H_O\left(-k-1\right)\\ &= -H_{O^2}\left(N-k-1\right).\end{aligned} \tag{3.67}$$

Applying m, s to the β term in (3.64), produces

$$\beta\left[\left(\frac{2k+1}{2}\right)\left(\frac{2n+1}{2}\right)\right] = \beta\left[\left(\frac{2k+1}{2}\right)\left(\frac{2m+1}{2}\right)+\frac{2k+1}{2}s\right] \tag{3.68}$$

and substituting (3.68) into (3.64) produces

$$\begin{aligned}Y_{O^2}\left(k\right) &= \left\langle \sum_{m=0}^{N-1} h\left(m\right)\sum_{s=0}^{N-1} x(s)\beta\left[\left(\frac{2k+1}{2}\right)\left(\frac{2m+1}{2}+\frac{2k+1}{2}s\right)\right]\right\rangle_{Mp}\\ &= \left\langle \sum_{m=0}^{N-1} h\left(m\right)\sum_{s=0}^{N-1} x(s)\left\{\beta_1\left[\left(\frac{2k+1}{2}\right)\left(\frac{2m+1}{2}\right)\right]\beta\left(\frac{2k+1}{2}s\right)\right.\right.\\ &\quad \left.\left.+\beta_2\left[\left(\frac{2k+1}{2}\right)\left(\frac{2m+1}{2}\right)\right]\beta\left(-\frac{2k+1}{2}s\right)\right\}\right\rangle_{Mp}\\ &= \left\langle \sum_{m=0}^{N-1} h\left(m\right)\beta_1\left[\left(\frac{2k+1}{2}\right)\left(\frac{2m+1}{2}\right)\right]\sum_{s=0}^{N-1} x(s)\beta\left(\frac{2k+1}{2}s\right)\right.\\ &\quad \left.+\sum_{m=0}^{N-1} h\left(m\right)\beta_2\left[\left(\frac{2k+1}{2}\right)\left(\frac{2m+1}{2}\right)\right]\sum_{s=0}^{N-1} x(s)\beta\left(-\frac{2k+1}{2}s\right)\right\rangle_{Mp}.\end{aligned} \tag{3.69}$$

To simplify (3.69) according to (3.66), (3.67) and (3.48), then (3.49) yields

$$\begin{aligned}Y_{O^2}\left(k\right) &= \left\langle \frac{H_{O^2}\left(k\right)-H_{O^2}\left(N-K-1\right)}{2}X_O\left(k\right)\right.\\ &\quad \left.+\frac{H_{O^2}\left(k\right)+H_{O^2}\left(N-k-1\right)}{2}X_O\left(N-k-1\right)\right\rangle_{Mp}\\ &= \left\langle \left\{\left[H_{O^2}\left(k\right)-H_{O^2}\left(N-k-1\right)\right]X_O\left(k\right)\right.\right.\\ &\quad \left.\left.+\left[H_{O^2}\left(k\right)+H_{O^2}\left(N-k-1\right)\right]X_O\left(N-k-1\right)\right\}2^{p-1}\right\rangle_{Mp}.\end{aligned} \tag{3.70}$$

The function operator $\odot$ as shown in Figure 3.3 is defined as

$$\begin{aligned}Y_{O^2}\left(k\right) &= X_O\left(k\right)\odot H_{O^2}\left(k\right)\\ &= \left\langle H^{od}_{O^2}(k)X_O\left(k\right)\right.\\ &\quad \left.+H^{ev}_{O^2}(k)X_O\left(N-k\right)\right\rangle_{Mp}\end{aligned} \tag{3.71}$$

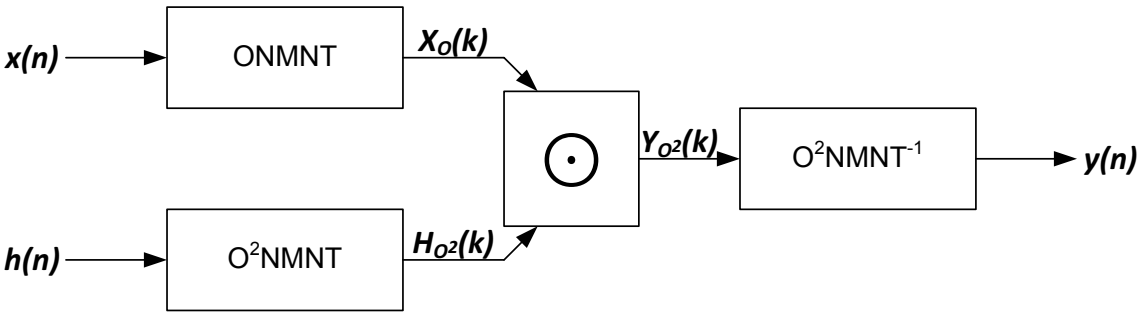


Figure 3.3: Convolution Process Structure for the O^{2}NMNT

where

$$H_{O^2}^{ev}(k) = \left\langle \left[H_{O^2}(k) + H_{O^2}(-k)\right] 2^{p-1} \right\rangle_{Mp} \tag{3.72}$$

and

$$H_{O^2}^{od}(k) = \left\langle \left[H_{O^2}(k) - H_{O^2}(-k)\right] 2^{p-1} \right\rangle_{Mp}. \tag{3.73}$$

An example of using the O^{2}NMNT to perform a cyclic convolution is shown as

$$x(n) = [1, 0, 0, 0, 0, 1, 1, 0], \tag{3.74}$$

$$h(n) = [0, 1, 0, 0, 1, 0, 1, 1]. \tag{3.75}$$

The resulting O^{2}NMNT representations are

$$\begin{aligned} X_O(k) = [&3454, 6272, 3545, 18, 6333, 3215, 6685, 7957, \\ &1642, 7015, 2802, 1830, 4957, 8075, 3354, 6581], \end{aligned} \tag{3.76}$$

and

$$\begin{aligned} H_{O^2}(k) = [&1973, 5938, 6361, 2277, 1222, 606, 3763, 4807, \\ &7005, 5958, 225, 7841, 2599, 4844, 464, 5500]. \end{aligned} \tag{3.77}$$

Applying the circular convolution algorithm produces

$$\begin{aligned} Y_{O^2}(k) = [&7595, 4072, 3529, 543, 963, 3337, 7814, 2753, \\ &4813, 6359, 3429, 4057, 6479, 7859, 2744, 8088], \end{aligned} \tag{3.78}$$

and applying the inverse O^{2}NMNT results in the first $2N-1$ elements as

$$ycc = [0, 1, 0, 0, 1, 0, 2, 2, 0, 1, 1, 1, 2, 1, 0]. \tag{3.79}$$

Confirming the results with the built in function of MATLAB produces

$$\text{conv(x, h)} = [0, 1, 0, 0, 1, 0, 2, 2, 0, 1, 1, 1, 2, 1, 0]. \tag{3.80}$$

Again, the variables x and h are input signal and impulse response accordingly, which are generated randomly and y_{CC} is the resultant cyclic convolution using the O^{2}NMNT. The result was verified using the built-in functions of MATLAB.

3.7 Encryption Example using the GNMNT

This section demonstrates the use of the GNMNT transforms as part of an encryption application. As a demonstration, this is a basic implementation to show some application techniques and the performance of each transform. There are no additional rounds and operations that one would usually find in general encryption schemes. This process can also be considered as a straightforward ECB implementation, which is the simplest form of encryption and, as previously noted in Section 2.5.1, is not a recommended technique for general use implementations. The algorithm will take the plaintext equivalent and transform it to the associate domain by

$$T_{\mathcal{G}} = \mathcal{G}(t) \tag{3.81}$$

where t is the plaintext, and $T_{\mathcal{G}}$ is the transformed plaintext using one of the GNMNT transforms denoted by $\mathcal{G}$. The key is randomly chosen as

$$K = \{\text{0x3e41, 0x1a4d, 0x273f, 0xf670, 0x39f4, 0xea3e, 0xddc1, 0x99a0}\} \tag{3.82}$$

and to test the effect of using an incorrect key, an alternate random key will be used shown as

$$\hat{K} = \{\text{0x9ef7, 0x2913, 0x7503, 0xf2cc, 0x0ea8, 0xd934, 0xaab1, 0x2903}\}. \tag{3.83}$$

The cipher is then created by

$$C_{\mathcal{G}} = \langle T_{\mathcal{G}} \otimes K \rangle_{Mp} \tag{3.84}$$

and

$$c = \mathcal{G}\left(C_{\mathcal{G}}\right)^{-1} \tag{3.85}$$

where $\mathcal{G}\left(\cdot\right)^{-1}$ denotes the matching inverse GNMNT transform to what was used in (3.81) and $\otimes$ denotes point-by-point multiplication. The inverse keys are then derived by

$$K^{-1} = \left\langle K^{Mp-2} \right\rangle_{Mp} \tag{3.86}$$

$$\hat{K}^{-1} = \left\langle \hat{K}^{Mp-2} \right\rangle_{Mp}. \tag{3.87}$$

The unadulterated key from (3.86) is the used to decrypt c to recover r, which represents the original plaintext t by first deriving $R_{\mathcal{G}}$ as

$$R_{\mathcal{G}} = \left\langle C_{\mathcal{G}} \otimes K^{-1} \right\rangle_{Mp} \tag{3.88}$$

and then

$$r = \mathcal{G}\left(R_{\mathcal{G}}\right)^{-1} \tag{3.89}$$

noting that

$$r = t. \tag{3.90}$$

In addition, should we also wish to see the resultant plaintext with the use of a different key then we compute

$$\hat{R}_{\mathcal{G}} = \left\langle C_{\mathcal{G}} \otimes \hat{K}^{-1} \right\rangle_{Mp} \tag{3.91}$$

and then

$$\hat{r} = \mathcal{G}\left(\hat{R}_{\mathcal{G}}\right)^{-1}. \tag{3.92}$$

We can then display t, c, $\hat{r}$ and r, which are the results for each stage. This type of encryption will be ECB mode, which is the simplest form of encryption where each block will be independent from all others. This will in due course also serve to show the inherent weakness of the ECB mode, the use of which is strongly discouraged. Additionally, each cipher will be produced using a single round and a single operation, not the several rounds and further operations that are usually expected from a typical symmetric encryption system.

3.7.1 Encryption using the NMNT

Earlier research has been undertaken using the NMNT, where the transform has been proposed for use as a component within an encryption system [61,110,111,118]. The methodology used in this example is achieved by first considering the parameters of the transform; $N = 8$ and $p = 17$ to derive $Mp = 131071$. The first 16 pixels to be encrypted are then obtained by

$$\begin{aligned} t = \{&\text{0xce, 0xd0, 0xd1, 0xcf, 0xd1, 0xd1, 0xcf, 0xd1,} \\ &\text{0xd0, 0xcf, 0xd0, 0xd1, 0xd3, 0xd1, 0xd0, 0xd2}\} \end{aligned} \tag{3.93}$$

and concatenating subsequent pairs of bytes so that

$$t = \{\text{0xced0, 0xd1cf, 0xd1d1, 0xcfd1, 0xd0cf, 0xd0d1, 0xd3d1, 0xd0d2}\}. \tag{3.94}$$

Applying the NMNT to the vector t we obtain

$$T_{\mathcal{G}} = \{\text{0x08887, 0x00202, 0x1f7ff, 0x00000, 0x001fe, 0x1fdff, 0x1fbf9, 0x1f801}\} \tag{3.95}$$

and subsequently after applying the key from (3.82) in (3.84) we get

$$C_{\mathcal{G}} = \{\text{0x16Ce0, 0x0ceb4, 0x00763, 0x00000, 0x17451, 0x18315, 0x1c7bb, 0x030da}\}. \tag{3.96}$$

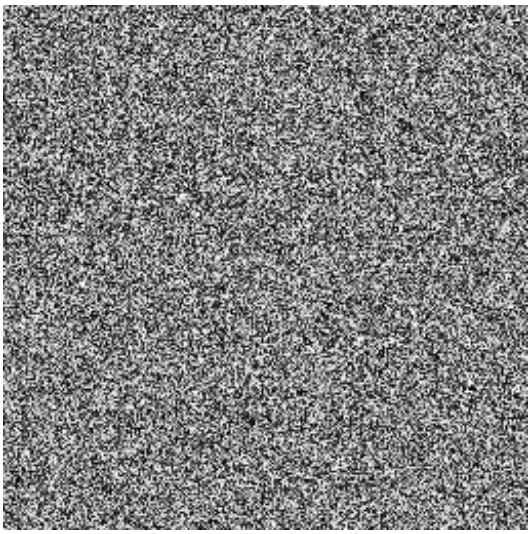
The resultant cipher c is then produced by applying (3.96) to the inverse NMNT in (3.85) to obtain

$$c = \{\text{0x1665e, 0x02da3, 0x0de24, 0x09f5d, 0x145b5, 0x1c096, 0x0e660, 0x06eb0}\}. \tag{3.97}$$

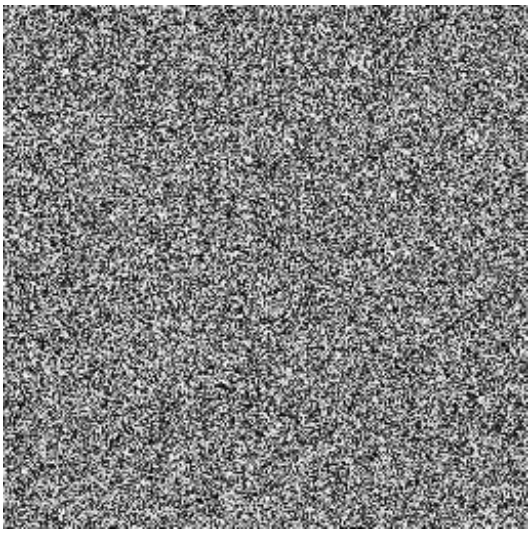
Reverting c to the GNMNT domain $C_{\mathcal{G}}$ allows recovery of the plaintext r, which can be obtained by using the inverse key (3.86) with $C_{\mathcal{G}}$ in (3.89). Similarly, recovering the alternate plaintext $\hat{r}$ using the alternate key from (3.83) can be obtained by applying the inverse of the altered key (3.87) to $C_{\mathcal{G}}$ in (3.91) to respectively produce Figures 3.5(a)-3.5(d) for the cameraman image.

$$\hat{R}_{\mathcal{G}} = \{\text{0x02c52, 0x0dcca, 0x06606, 0x00000, 0x149ec, 0x13a3a, 0x12ac2, 0x191e3}\} \tag{3.98}$$

(a) Original



(b) Cipher



(c) Incorrect Key

(d) Correct Key

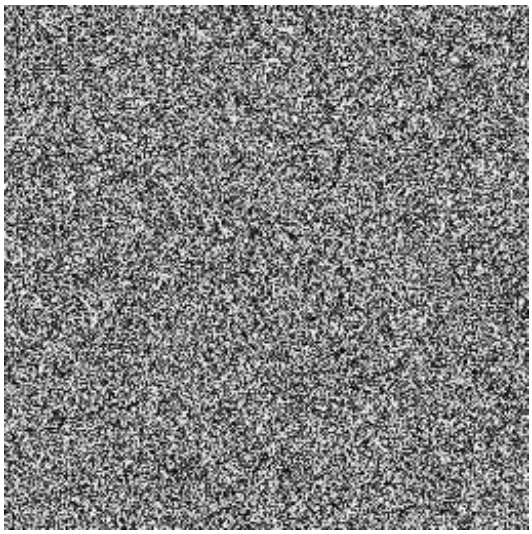
Figure 3.4: Encrypted Cameraman using NMNT, $N = 8$ and $Mp = 131071$

and

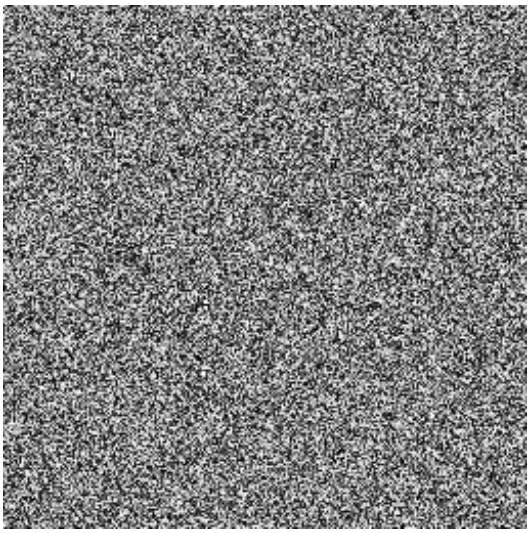
$$\hat{r} = \{0x015fe, 0x0edd6, 0x12c0a, 0x19fc0, 0x1abc3, 0x1fbf1, 0x1cd52, 0x0e7a9\}. \quad (3.99)$$

Using the public domain image of the cameraman, the results are shown for the original image as the plaintext p_T, ciphered image c_T, the recovered image using incorrect key $\hat{r}_T$ and the recovered image using the correct key r_T in Figures 3.4(a)-3.4(d) respectively.

(a) Original



(b) Cipher



(c) Incorrect Key

(d) Correct Key

Figure 3.5: Encrypted Cameraman using ONMNT, $N = 8$ and $Mp = 131071$

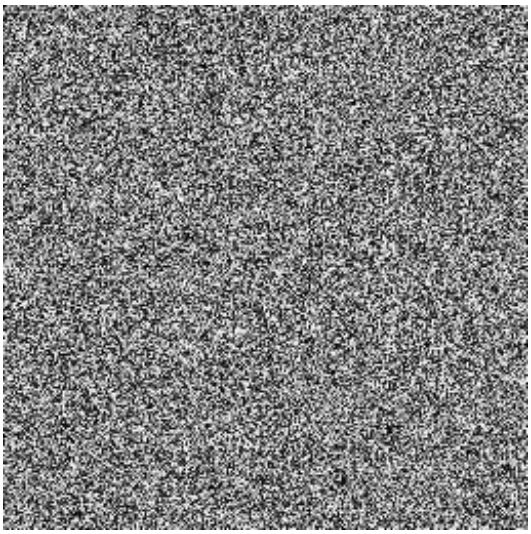
3.7.2 Encryption using the ONMNT

The application of the ONMNT as part of an encryption scenario is a very new concept. While much has been published with the NMNT on this subject, the only research to date using the ONMNT can be found in [119]. The results of the encryption effectiveness can be shown in Figures 3.5(a)-3.5(d).

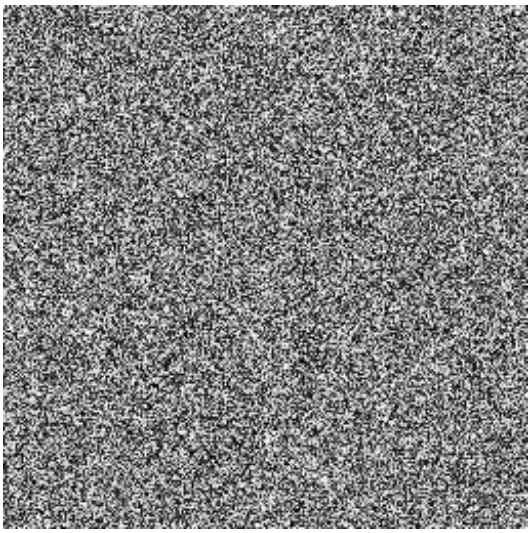
3.7.3 Encryption using the O^{2}NMNT

Like the ONMNT, the application of the O^{2}NMNT within the field of encryption is very recent. However, there is a stronger interest in O^{2}NMNT as characteristics of this transform appear to show that it is extremely resilient according to [119]. The

(a) Original



(b) Cipher



(c) Incorrect Key

(d) Correct Key

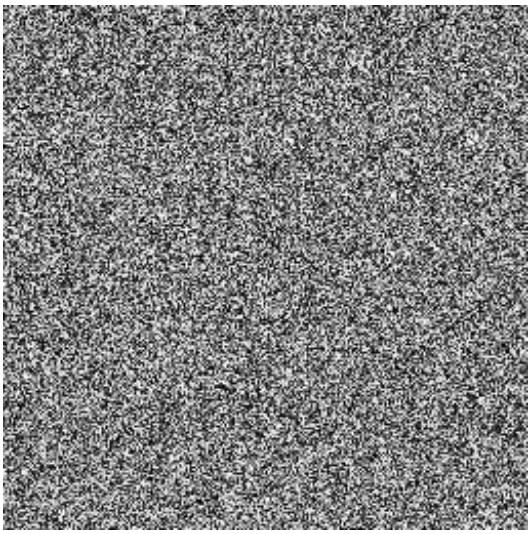
Figure 3.6: Encrypted Cameraman using O^{2}NMNT, $N = 8$ and $Mp = 131071$

results of the process using the O^{2}NMNT are provided in Figures 3.6(a)-3.6(d).

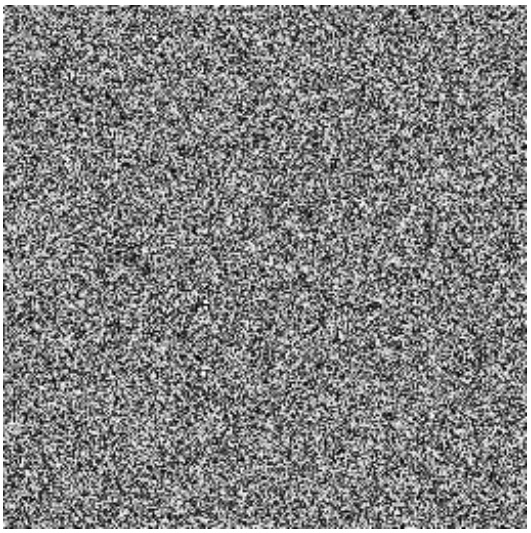
3.7.4 Encryption using the GNMNT

From the initial assessment of the processes covered, it can be seen that all transforms appear to perform equally. This is shown by depicting an unrecoverable image upon decrypting using the incorrect key for each transform, which have been collated in Figures 3.7(a)-3.7(d). However, in order to get a little insight into how the transforms actually perform, the kernel components must be investigated a little more deeply.

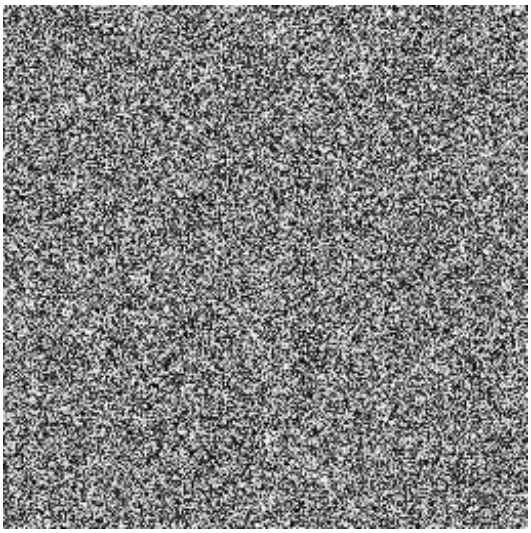
(a) Original



(b) NMNT



(c) ONMNT



(d) O^2NMNT

Figure 3.7: Decrypted Cameraman Error using GNMNT, $N = 8$ and $Mp = 131071$

3.8 The GNMNT Kernel Components

The kernel components can be seen as the heart of the transform. These components are quite easily observed in the GNMNT matrices and as such, can give an indication as to how these attributes will reinforce or diminish the security properties of the transform. One of the significant metrics of an encryption system is how its diffusion characteristics perform. The characteristics that govern this aspect will be covered in greater detail later in Chapter 6. However, it would be relevant to describe the observed characteristics of each transform kernel observation while the kernels have recently been presented.

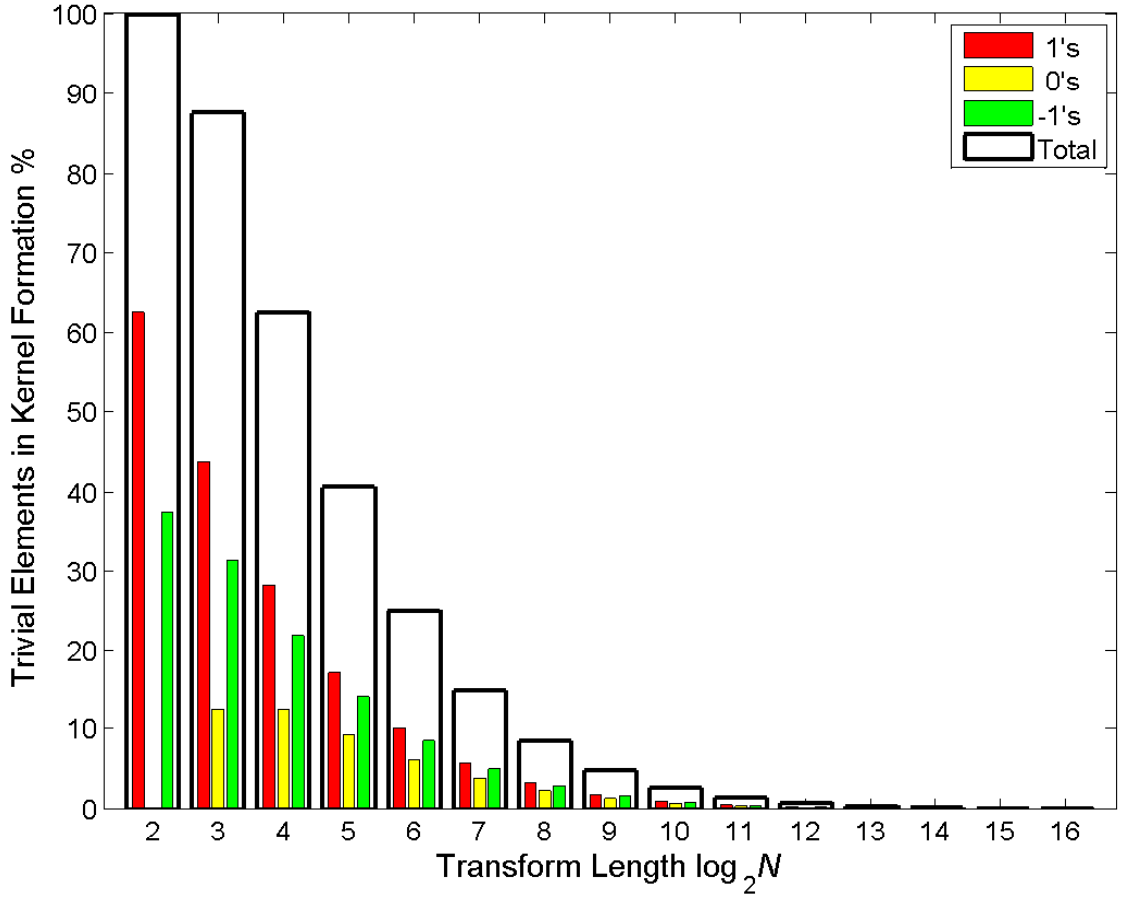


Figure 3.8: Consistency of Trivial Elements in the NMNT Kernel

3.8.1 The NMNT Kernel Components

The NMNT kernel is both orthogonal and symmetric. As such, it is its own inverse when used with the corresponding scaling factor. One of the characteristics that was observed with the NMNT was a potential weakness in its diffusion properties, which when used with encryption, are shown to be very linear. While [110] and [111] provide an intensive study in this area, the work focused more upon the changing of the elements rather than the effect upon the bits. As such, a significant amount of information can be gained through the manipulation of a single bit; this will be discussed further in Chapter 6. However, one of the biggest attributes to this observable problem may well be the construction of the matrix itself in that it contains a significant amount of trivial elements, i.e. one, zero and the field equivilant of minus one in quantities that can be determined by

$$\#\{(\mathcal{G}_N, m(1))\} = N\left[\log_2(N) + 0.5\right] \tag{3.100}$$

$$\#\{(\mathcal{G}_N, m(0))\} = N\left[\log_2(N) - 2\right] \tag{3.101}$$

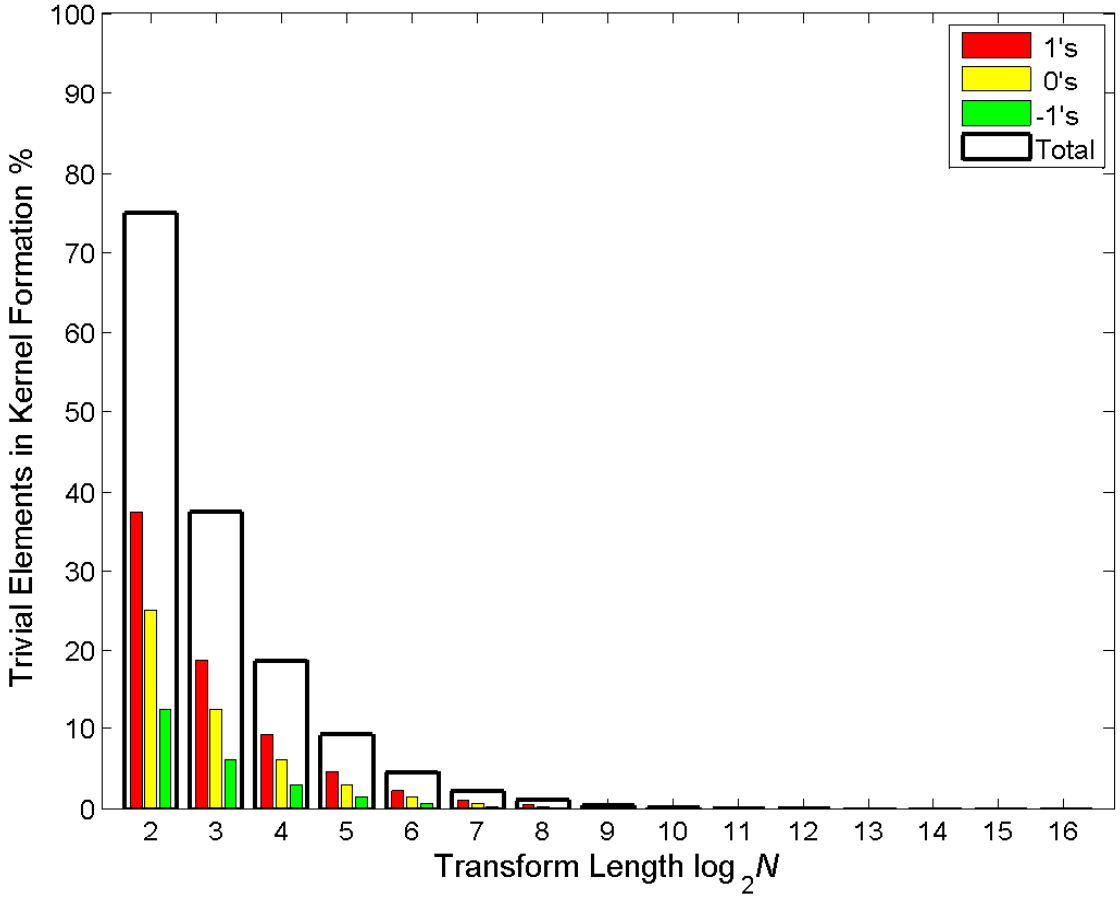


Figure 3.9: Consistency of Trivial Elements in the ONMNT Kernel

and

$$\#\{(\mathcal{G}_N, m\,(-1))\} = N\,[\log_2(N) - 0.5]\,, \tag{3.102}$$

where $\mathcal{G}_N$ denotes the NMNT kernel matrix and the field equivalent of a minus one is $Mp - 1$. Figure 3.8 better illustrates how these elements are dispersed throughout the kernel.

Elements containing either a one or a minus one can potentially be a means to attack the system using cribs, particularly where both the first row and column contain ones, indicating that the first element in the transformed vector is the sum of the initial vector, modulo the Mersenne prime. However, the greatest threat appears to be the number of zeros contained within the NMNT kernel. This can be shown by using the same implementation and process as Section 3.7 but decrypting using a very slightly modified key to that shown in (3.83), where the final bit in the final element of the key has changed to reflect

$$\hat{K} = \{0\text{x}3\text{e}41,\ 0\text{x}1\text{a}4\text{d},\ 0\text{x}273\text{f},\ 0\text{x}f670,\ 0\text{x}39\text{f}4,\ 0\text{x}ea3\text{e},\ 0\text{x}ddc1,\ 0\text{x}99\text{a}\underline{1}\}. \tag{3.103}$$

The result of decrypting the image with the NMNT by using a key that differs by a single bit is shown in Figure 3.10(b), where it can quite easily be observed that part of the image has been recovered. However, it can be further observed that this effect diminishes as the transform length and subsequent key length increases as show in Figures 3.10(c)-3.10(f), where increasingly longer keys were derived randomly. The reason why part of the image has been recovered using a key differing by a single bit will be explained later in this chapter.

3.8.2 The ONMNT Kernel Components

Using the same methods that were used with the NMNT, we can derive the consistency of trivial elements for the ONMNT. As the IONMNT is a transpose of the ONMNT, the same formulae will work between both transforms for determining the number of ones, zeros and minus ones in quantities that can be determined by

$$\#\{(\mathcal{G}_O, m\,(1))\} = \frac{3N}{2} \tag{3.104}$$

$$\#\{(\mathcal{G}_O, m\,(0))\} = N \tag{3.105}$$

and

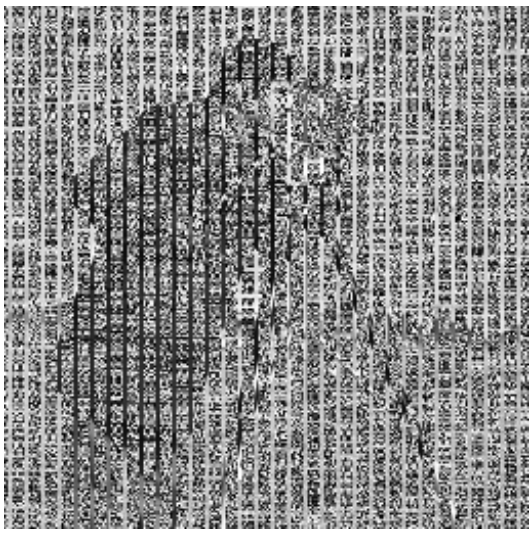
$$\#\{(\mathcal{G}_N, m\,(-1))\} = \frac{N}{2}. \tag{3.106}$$

From Figure 3.11 it can be observed that the ONMNT also suffers from the same inherent problems as the NMNT where part of the image has been recovered using a decryption key that differs from the encryption key by a single bit. It can again be further observed that this effect diminishes as the transform length and subsequent key length increases as show in Figures 3.11(b)-3.11(f), where increasingly longer keys were derived randomly. Proof of this will be provided later in this chapter.

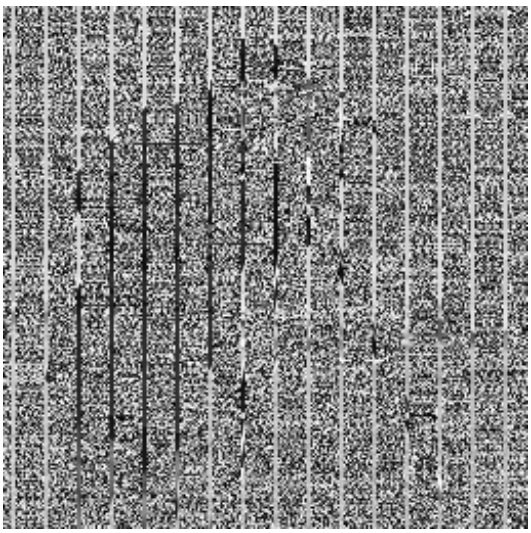
3.8.3 The O^{2}NMNTKernel Components

There is a unique aspect with the O^{2}NMNT in that it does not contain any of the trivial elements whatsoever. As will be analysed later, this suggests that the O^{2}NMNT would perform very well for security applications as there are no components in the kernel to copy or invert elements from the input vector to potentially perform attacks using cribs. More importantly, there are no elements that nullify the effects of the key during the decryption process, as shown in Figure

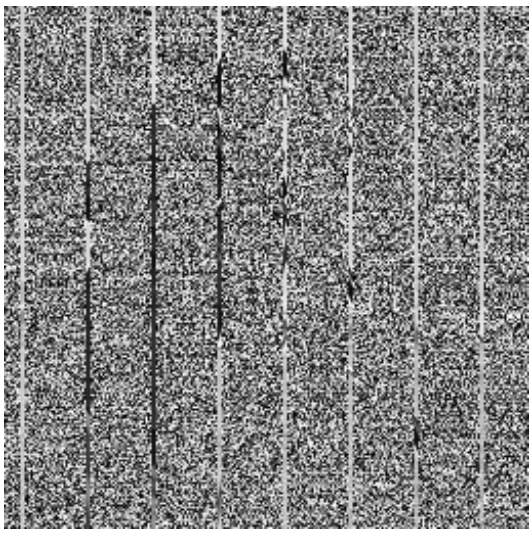
(a) Original Image



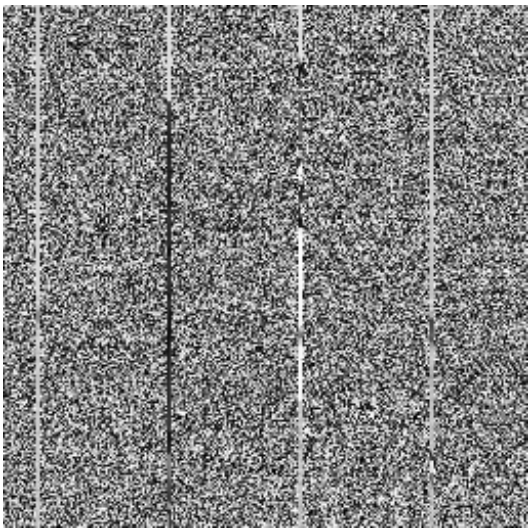
(b) NMNT Bit Error $N = 8$



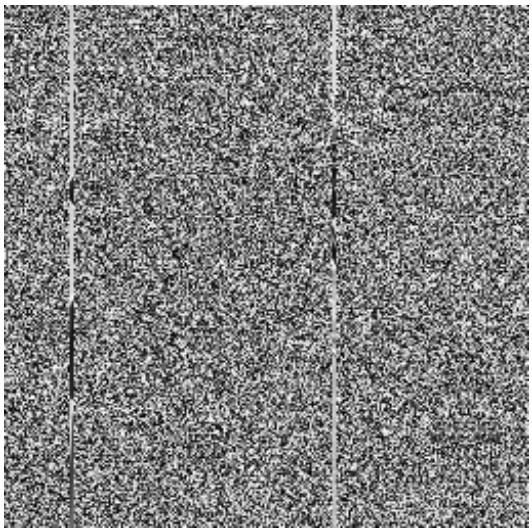
(c) NMNT Bit Error $N = 16$



(d) NMNT Bit Error $N = 32$



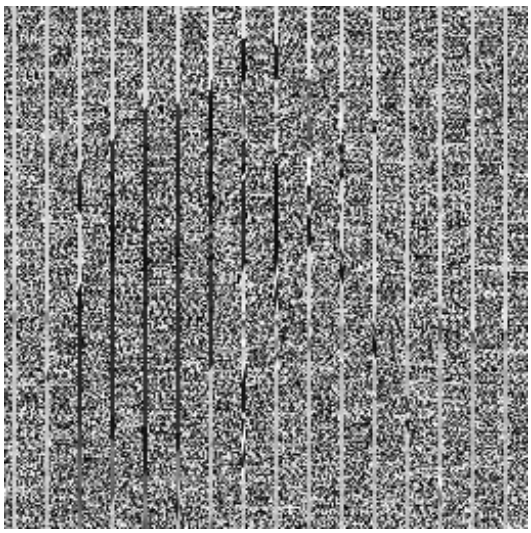
(e) NMNT Bit Error $N = 64$



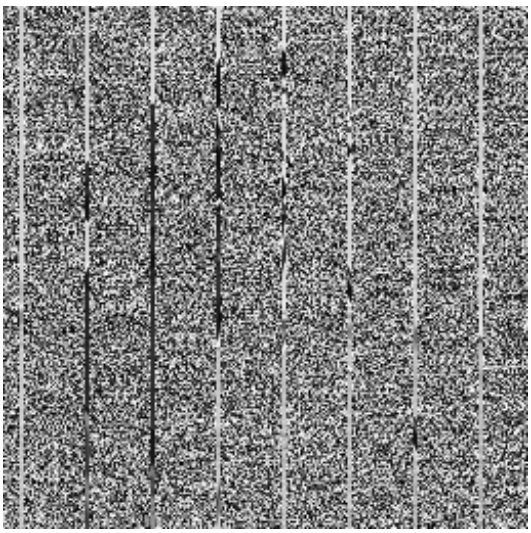
(f) NMNT Bit Error $N = 128$

Figure 3.10: Decrypted 256×256 Cameraman Bit Error using NMNT with $N = 8, 16, 32, 64, 128$, $Mp = 131071$

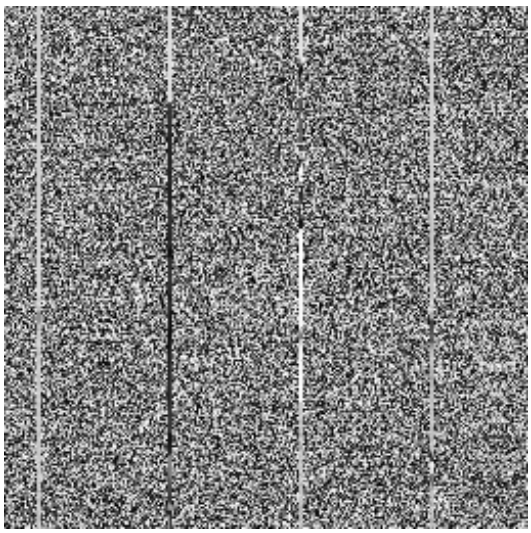
(a) Original Image



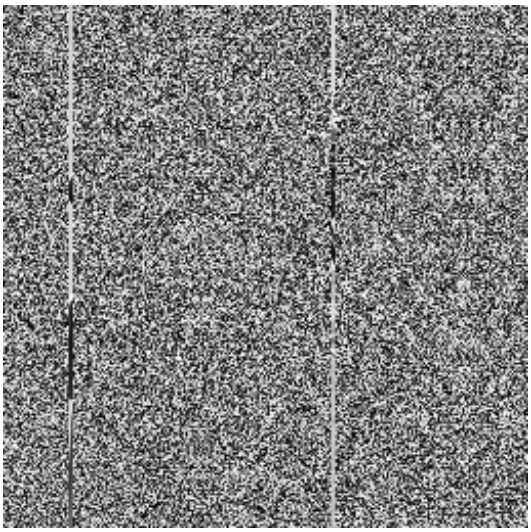
(b) ONMNT Bit Error $N = 8$



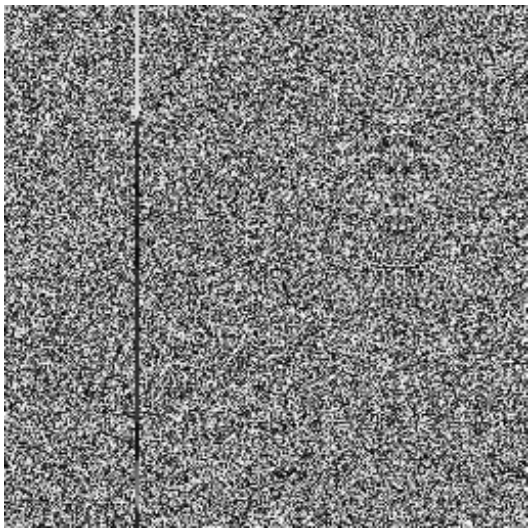
(c) ONMNT Bit Error $N = 16$



(d) ONMNT Bit Error $N = 32$



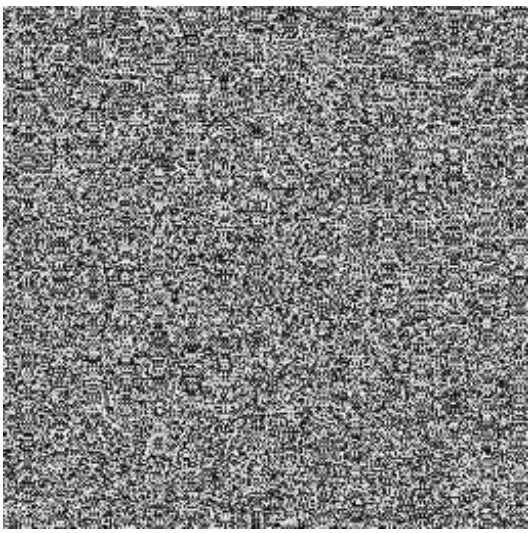
(e) ONMNT Bit Error $N = 64$



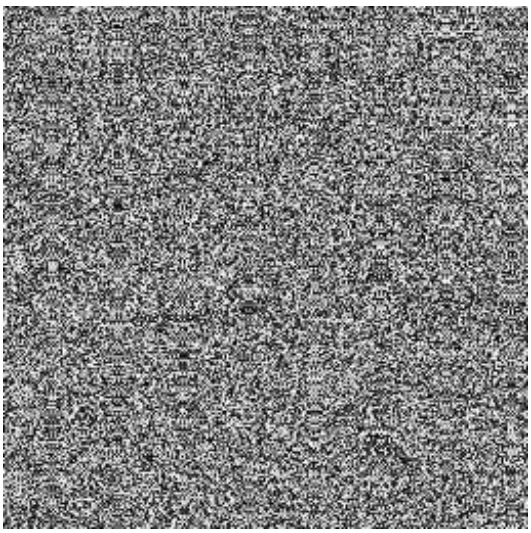
(f) ONMNT Bit Error $N = 128$

Figure 3.11: Decrypted 256×256 Cameraman Bit Error using ONMNT with $N = 8, 16, 32, 64, 128,\ Mp = 131071$

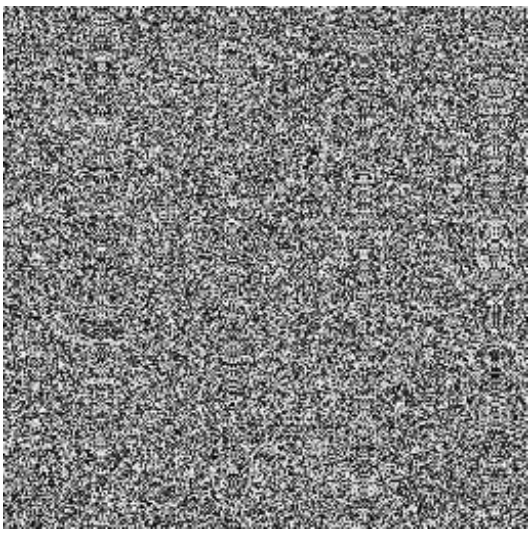
(a) Original Image



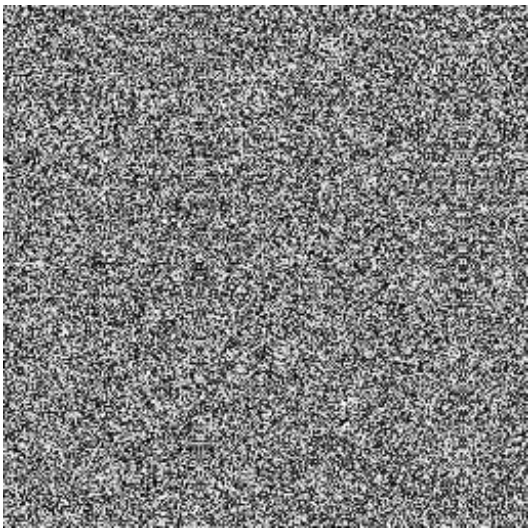
(b) O^{2}NMNT Bit Error $N = 8$



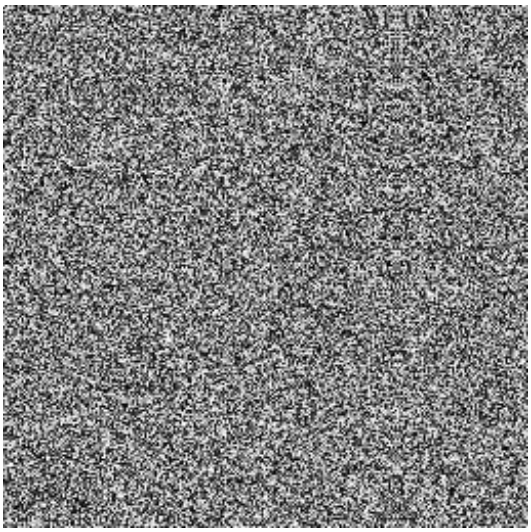
(c) O^{2}NMNT Bit Error $N = 16$



(d) O^{2}NMNT Bit Error $N = 32$



(e) O^{2}NMNT Bit Error $N = 64$



(f) O^{2}NMNT Bit Error $N = 128$

Figure 3.12: Decrypted 256×256 Cameraman Bit Error using O^{2}NMNT with $N = 8, 16, 32, 64, 128$, $Mp = 131071$

3.12. The resultant image using the O^{2}NMNT shows significant promise over the NMNT and ONMNT counterparts, where a similar level of performance appears to have been reached when using a completely different key in Section 3.7.3.

3.8.4 Proving the Detriments of Zero-Elements within the GNMNT

The effects of the phenomenon where stripes can be observed are easily described where earlier claims were made indicating that the zero-elements within the kernel matrices were the cause.

Proof. If we first show the input vector x as

$$x = [x_0, x_1, \ldots, x_{N-1}] \tag{3.107}$$

and the NMNT kernel as

$$M = \begin{bmatrix} \beta(nk) & \beta[n(k+1)] & \beta[n(k+j)] & \ldots & \beta[n(N-k-1)] \\ \beta[(n+1)k] & \beta[(n+1)(k+1)] & \beta[(n+1)(k+j)] & \ldots & \beta[(n+1)(N-k-1)] \\ \beta[(n+i)k] & \beta[(n+i)(k+1)] & \beta[(n+i)(k+j)] & \ldots & \beta[(n+i)(N-k-1)] \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \beta[(N-n-1)k] & \beta[(N-n-1)(k+1)] & \beta[(N-n-1)(k+j)] & \ldots & \beta[(N-n-1)(N-k-1)] \end{bmatrix} \tag{3.108}$$

where

$$0 \leq i, j \leq N-1 \tag{3.109}$$

then by applying

$$X = \langle xM \rangle_{Mp} \tag{3.110}$$

where the transformed vector X is similarly constructed as

$$X = [X_0, X_1, \ldots, X_{N-1}] \tag{3.111}$$

the derivation of each element of X as a result from applying the NMNT to x can

be shown as

$$\begin{aligned}
X_0 &= x_0\beta\left(nk\right) + x_1\beta\left[\left(n+1\right)k\right] \\
&\quad + x_i\beta\left[\left(n+i\right)k\right] + \ldots + x_{N-n-1}\beta\left[\left(N-n-1\right)k\right] \\
X_1 &= x_0\beta\left[n\left(k+1\right)\right] + x_1\beta\left[\left(n+1\right)\left(k+1\right)\right] \\
&\quad + x_i\beta\left[\left(n+i\right)\left(k+1\right)\right] + \ldots + x_{N-n-1}\beta\left[\left(N-n-1\right)\left(k+1\right)\right] \\
X_j &= x_0\beta\left[n\left(k+j\right)\right] + x_1\beta\left[\left(n+1\right)\left(k+j\right)\right] \\
&\quad + x_i\beta\left[\left(n+i\right)\left(k+j\right)\right] + \ldots + x_{N-n-1}\beta\left[\left(N-n-1\right)\left(k+j\right)\right] \\
&\vdots \\
X_{N-k-1} &= x_0\beta\left[n\left(N-k-1\right)\right] + x_1\beta\left[\left(n+1\right)\left(N-k-1\right)\right] \\
&\quad + x_i\beta\left[\left(n+i\right)\left(N-k-1\right)\right] + \ldots + x_{N-n-1}\beta\left[\left(N-n-1\right)\left(N-k-1\right)\right].
\end{aligned} \tag{3.112}$$

If we define $\hat{x}$ as a copy of vector x where a single element has been changed shown as $\hat{x}_r$ then we obtain a new transformed vector by

$$\hat{X} = \langle \hat{x} M \rangle_{Mp} \tag{3.113}$$

and can show that

$$\hat{X}_j = X_j \text{ when } \hat{x}_i \neq x_i \iff \beta\left[\left(n+i\right)\left(k+j\right)\right] = 0, \tag{3.114}$$

where the unaffected resultant vector element(s) correspond to the matrix column(s) that have a zero at the same row position as the modified key element. □

Similarly, we can show that this also works with the ONMNT by applying the matrix transpose, which for the NMNT yields the same matrix kernel.

Proof. Representing the transformed vector X as Y and derive the inverse

$$y = \left\langle \frac{1}{N}\left[YM'\right]\right\rangle_{Mp} \tag{3.115}$$

noting that

$$y = x \tag{3.116}$$

then defining $\hat{Y}$ as a copy of Y where a single element has been changed shown as

$\hat{Y}_i$ and transforming the modified vector out of the NMNT domain by

$$\hat{y} = \left\langle \frac{1}{N} \left[\hat{Y} M^{'} \right] \right\rangle_{Mp} \tag{3.117}$$

we can show that

$$\hat{y}_j = y_j \text{ when } \hat{Y}_i \neq Y_i \iff \beta \left[(n+i)(k+j) \right] = 0, \tag{3.118}$$

where again the unaffected resultant vector element(s) correspond to the matrix column(s) that have a zero at the same row position as the modified key element. □

This corresponds to the diminishing number of zero-elements in the NMNT kernel as described in (3.101) illustrated in Figure 3.8 and depicted in Figure 3.10 building upon the alternate key (3.103). This can be further verified when considering the NMNT kernel that was constructed using the α_1 and $\overline{\alpha_2}$ values from Table 3.6

$$M = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 512 & 1 & 0131070 & 130559 & 131070 & 0 & \\ 1 & 1 & 131070 & 131070 & 1 & 1 & 131070 & 131070 \\ 1 & 0 & 131070 & 512 & 131070 & 0 & 1 & 130559 \\ 1 & 131070 & 1 & 131070 & 1 & 131070 & 1 & 131070 \\ 1 & 130559 & 1 & 0 & 131070 & 512 & 131070 & 0 \\ 1 & 131070 & 131070 & 1 & 1 & 131070 & 131070 & 1 \\ 1 & 0 & 131070 & 130559 & 131070 & 0 & 1 & 512 \end{bmatrix} \tag{3.119}$$

and noting that changing the last element of the key will have no bearing on the resultant operation owing that

$$M_{i,j} = 0 \text{ for } i = N-1, j \in \left\{ \frac{N}{8}, \frac{5N}{8} \right\} \text{ and } 8 \leq N \leq 2^p. \tag{3.120}$$

Remembering that there are two pixels in each element would further explain the recovery of the presented parts of the images.

This phenomenon can also be demonstrated with the ONMNT using the same methods and conditions but replacing (3.119) with the appropriate version of the kernel for the IONMNT that was constructed using the α_1 and $\overline{\alpha_2}$ values from

Table 3.7

$$M = \begin{bmatrix} 1 & 82137 & 512 & 82137 & 1 & 29287 & 0 & 101784 \\ 1 & 82137 & 0 & 48934 & 131070 & 29287 & 512 & 29287 \\ 1 & 29287 & 130559 & 29287 & 1 & 48934 & 0 & 82137 \\ 1 & 101784 & 0 & 29287 & 131070 & 82137 & 130559 & 82137 \\ 1 & 48934 & 512 & 48934 & 1 & 101784 & 0 & 29287 \\ 1 & 48934 & 0 & 82137 & 131070 & 101784 & 512 & 101784 \\ 1 & 101784 & 130559 & 101784 & 1 & 82137 & 0 & 48934 \\ 1 & 29287 & 0 & 101784 & 131070 & 48934 & 130559 & 48934 \end{bmatrix} \tag{3.121}$$

and this time noting that

$$M_{i,j} = 0 \text{ for } i = N-1, j = \frac{N}{4} \text{ and } 4 \leq N \leq 2^p \tag{3.122}$$

would correspond the shortcomings of Figure 3.11. This again would correspond to the diminishing number of zero-elements in the ONMNT kernel as described in (3.105) illustrated in Figure 3.9. Therefore the only way that these anomalies can be avoided is by applying multiple rounds to the process and shuffling the result of each round prior to supplying it as an input to the next round.

3.8.5 Demonstration of the Proof

We can provide an overall demonstration using a simple matrix unrelated to the GNMNTs as follows.

Proof. Using $N = 4$ we define an arbitrary matrix

$$M = \begin{bmatrix} 95 & 70 & 69 & 94 \\ 68 & 121 & 0 & 44 \\ 42 & 113 & 79 & 104 \\ 105 & 0 & 101 & 3 \end{bmatrix} \tag{3.123}$$

and vector

$$x = [52, 92, 99, 46] \tag{3.124}$$

so that

$$\begin{aligned} X &= \langle xM \rangle_{127} \\ &= [118, 51, 53, 66]. \end{aligned} \tag{3.125}$$

Reusing the vector in (3.124) and changing the last value of the input vector $\hat{x}_3 = 3$ shown as

$$\hat{x} = [52, 92, 99, 3] \tag{3.126}$$

results in

$$\begin{aligned} \hat{X} &= \langle \hat{x}M \rangle_{127} \\ &= [48, \underline{51}, 21, 64] \end{aligned} \tag{3.127}$$

so that there is no change in the second value of the output vectors between (3.125) and (3.127), which can be shown as $X_1 = \hat{X}_1$. Similarly, reusing the vector in (3.124) and changing the second value of the input vector $\hat{x}_1 = 3$ shown as

$$\hat{x} = [52, 3, 99, 46] \tag{3.128}$$

results in

$$\begin{aligned} \hat{X} &= \langle [52, 3, 99, 46]\, M \rangle_{127} \\ &= [35, 77, \underline{53}, 87] \end{aligned} \tag{3.129}$$

so that there is no change in the third value of the output vectors between (3.125) and (3.129), which can be shown as $X_2 = \hat{X}_2$. □

3.9 Conclusion

This chapter has introduced the new transforms of the GNMNT and demonstrated that they are easily adaptable for applications based upon convolution, verifying their capability and value in signal-processing. Moreover, while the original NMNT has seen a relevant amount of success and acceptance in the field of security, the extension of the NMNT to the GNMNT offers new and potentially stronger transforms for security applications. The preliminary examples of encryption within this chapter show that the GNMNT, particularly the O^{2}NMNT transforms to have comparable initial results in this field, which will be later explored in Chapter 6.

Chapter 4

Fast Algorithms of the GNMNT

4.1 Introduction

Fast algorithms have been a prime focus of research since the inception of the fast Fourier transform [120], which has initiated myriad pursuits in this area. Naturally, the quicker such processes can be undertaken, the greater throughputs can be achieved. Lowering typical power consumption and having a wider applicability further increases their attractiveness. In this section we will show how the GNMNT can be derived using fast algorithms like other transforms before it, such as the original NMNT, fast Fourier transform [120], fast Hartley transform [121], etc. Firstly, the radix-2 algorithms will be derived, followed by the radix-4 and split-radix algorithms. A complexity analysis will also be developed and timings for each algorithm will be included in order to compare and contrast.

4.2 Radix-2 ONMNT

The radix-2 algorithm is the simplest of the fast algorithms to both describe and derive. It works by splitting an N-length algorithm into two smaller $\frac{N}{2}$-length algorithms. The significance of this approach is quite profound when looking at metrics based on the Fourier transform, which reduces an $O(N^2)$ problem to an $O(N \log_2 N)$ problem. When $N = 1024$ then this represents a saving in processing of over 90%. This process can be repeated $\log_2 N$ times meaning that any length that is a power of two can use this process.

4.2.1 Radix-2 DIT

The first of the two radix-2 algorithms is the DIT algorithm. It is derived by splitting the N-length transform into two smaller $\frac{N}{2}$-length transforms representing even and odd parts. This can easily be visualised when taking into consideration the methods of indexing, which is X_{2n} for even indexing and X_{2n+1} for odd indexing. This decomposes the ONMNT algorithm

$$X(k) = \left\langle \sum_{n=0}^{N-1} x(n)\beta\left(\frac{2k+1}{2}n\right)\right\rangle_{Mp} \tag{4.1}$$

into

$$X(k) = X^{ev}(k) + X^{od}(k). \tag{4.2}$$

The even and odd components in (4.2) can be further expressed as

$$\begin{aligned} X^{ev}(k) &= \left\langle \sum_{n=0}^{\frac{N}{2}-1} x(2n)\beta\left(\frac{2k+1}{2}2n\right)\right\rangle_{Mp} \\ &= X_{2n}(k) \end{aligned} \tag{4.3}$$

and

$$X^{od}(k) = \left\langle \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)\beta\left[\frac{2k+1}{2}(2n+1)\right]\right\rangle_{Mp} \tag{4.4}$$

Simplifying β in (4.4) produces

$$\beta\left[\frac{2k+1}{2}(2n+1)\right] = \beta\left(\frac{2k+1}{2}2n + \frac{2k+1}{2}\right). \tag{4.5}$$

Using the identity given in [7] where

$$\beta(a+b) = \beta_1(a)\beta(b) + \beta_2(a)\beta(-b) \tag{4.6}$$

then (4.5) can be rewritten as

$$\begin{aligned} \beta\left(\frac{2k+1}{2}2n + \frac{2k+1}{2}\right) =& \beta_1\left(\frac{2k+1}{2}\right)\beta\left(\frac{2k+1}{2}2n\right) \\ &+ \beta_2\left(\frac{2k+1}{2}\right)\beta\left(-\frac{2k+1}{2}2n\right). \end{aligned} \tag{4.7}$$

To obtain a general odd form we get

$$X_{2n+1}(k) = \left\langle \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)\beta\left(\frac{2k+1}{2}2n\right)\right\rangle_{Mp} \tag{4.8}$$

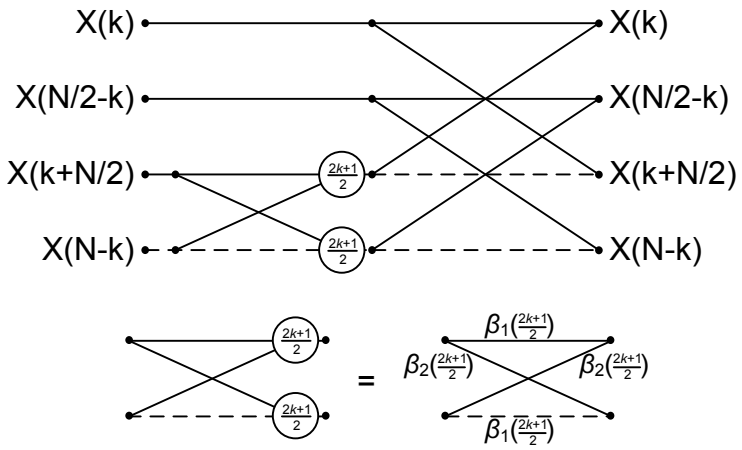


Figure 4.1: Radix-2 1D-ONMNT In-Place DIT Butterfly

and

$$X_{2n+1}\left(N-k\right)=\left\langle \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)\beta\left(-\frac{2k+1}{2}2n\right)\right\rangle_{Mp}. \tag{4.9}$$

Before combining (4.7) and (4.8) we first need to correct the final term in (4.7), which because of the periodic properties of the transform, can be expressed as

$$X^{od}=\left\langle \beta_1\left(\frac{2k+1}{2}\right)X_{2n+1}\left(k\right)+\beta_2\left(\frac{2k+1}{2}\right)X_{2n+1}\left(N-k\right)\right\rangle_{Mp}. \tag{4.10}$$

Combining (4.8) and (4.10) produces the first point k as

$$X\left(k\right)=\left\langle X_{2n}\left(k\right)+\beta_1\left(\frac{2k+1}{2}\right)X_{2n+1}\left(k\right)+\beta_2\left(\frac{2k+1}{2}\right)X_{2n+1}\left(N-k\right)\right\rangle_{Mp}. \tag{4.11}$$

In radix-2 there are four points: k, $\frac{N}{2}-k$, $k+\frac{N}{2}$ and $N-k$. Equation (4.11) shows the point k. Defining subsequent points is a little more involved and requires k to be replaced with the point position that is currently being derived. Similarly, it can be easy to get all equations at different point by replacing k. For example point $\frac{N}{2}-k$, $\because \frac{N}{2}-k :\Leftrightarrow \frac{N}{2}-k-1$ by definition, then

$$\beta\left[\frac{2\left(\frac{N}{2}-k-1\right)+1}{2}2n\right]=\beta\left(\frac{N-2k-1}{2}2n\right) \tag{4.12}$$

$$\beta\left[\frac{2\left(k+\frac{N}{2}\right)+1}{2}2n\right]=\beta\left(\frac{N+2k+1}{2}2n\right) \tag{4.13}$$

and

$$\beta\left[\frac{2\left(N-k-1\right)+1}{2}2n\right]=\beta\left(\frac{2N-2k-1}{2}2n\right). \tag{4.14}$$

According to the β in each point derived in (4.12)-(4.14), the following points can be derived by the respective processes as

$$\begin{aligned} X_{2n+1}\left(\tfrac{N}{2}-k\right) &= \left\langle \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)\beta\left[\frac{2\left(\frac{N}{2}-k-1\right)+1}{2}2n\right]\right\rangle_{Mp} \\ &= \left\langle \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)\beta\left(-\frac{2k+1}{2}2n\right)\right\rangle_{Mp} \\ &= X_{2n+1}\left(N-k\right) \end{aligned} \tag{4.15}$$

$$\begin{aligned} X_{2n+1}\left(k+\tfrac{N}{2}\right) &= \left\langle \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)\beta\left[\frac{2\left(k+\frac{N}{2}\right)+1}{2}2n\right]\right\rangle_{Mp} \\ &= \left\langle \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)\beta\left(\frac{2k+1}{2}2n\right)\right\rangle_{Mp} \\ &= X_{2n+1}\left(k\right) \end{aligned} \tag{4.16}$$

and

$$\begin{aligned} X_{2n+1}\left(N-k\right) &= \left\langle \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)\beta\left[\frac{2\left(N-k-1\right)+1}{2}2n\right]\right\rangle_{Mp} \\ &= \left\langle \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)\beta\left(-\frac{2k+1}{2}2n\right)\right\rangle_{Mp} \\ &= X_{2n+1}\left(N-k\right). \end{aligned} \tag{4.17}$$

The point $\left(\frac{N}{2}-k\right)$ is derived as

$$\begin{aligned} X\left(\tfrac{N}{2}-k\right) &= \left\langle X^{ev}\left(\tfrac{N}{2}-k\right)+X^{od}\left(\tfrac{N}{2}-k\right)\right\rangle_{Mp} \\ &= \left\langle \sum_{n=0}^{\frac{N}{2}-1} x(2n)\beta\left[\frac{2\left(\frac{N}{2}-k-1\right)+1}{2}2n\right]\right. \\ &\quad \left. + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)\beta\left[\frac{2\left(\frac{N}{2}-k-1\right)+1}{2}\left(2n+1\right)\right]\right\rangle_{Mp} \\ &= \left\langle X_{2n}\left(\tfrac{N}{2}-k\right)+\beta_1\left[\frac{2\left(\frac{N}{2}-k-1\right)+1}{2}\right]X_{2n+1}\left(\tfrac{N}{2}-k\right)\right. \\ &\quad \left. +\beta_2\left[\frac{2\left(\frac{N}{2}-k-1\right)+1}{2}\right]X_{2n+1}\left[N-\left(\tfrac{N}{2}-k\right)\right]\right\rangle_{Mp} \\ &= \left\langle X_{2n}\left(N-k\right)-\beta_1\left(\frac{2k+1}{2}\right)X_{2n+1}\left(N-k\right)\right. \\ &\quad \left. +\beta_2\left(\frac{2k+1}{2}\right)X_{2n+1}\left(k\right)\right\rangle_{Mp}. \end{aligned} \tag{4.18}$$

Similarly, in addition to equations (4.11) and (4.18) the rest of the points will be derived respectively as

$$X\left(k+\tfrac{N}{2}\right) = \left\langle X_{2n}(k) - \beta_1\left(\frac{2k+1}{2}\right) X_{2n+1}(k) - \beta_2\left(\frac{2k+1}{2}\right) X_{2n+1}(N-k)\right\rangle_{Mp} \tag{4.19}$$

and

$$X(N-k) = \left\langle X_{2n}(N-k) + \beta_1\left(\frac{2k+1}{2}\right) X_{2n+1}(N-k) - \beta_2\left(\frac{2k+1}{2}\right) X_{2n+1}(k)\right\rangle_{Mp}. \tag{4.20}$$

The radix-2 ONMNT DIT butterfly is shown in Figure 4.1.

4.2.2 Radix-2 DIF

The radix-2 DIF algorithm is the complement of the DIT. With the NMNT, the DIT transform also serves to calculate the forward and inverse transforms, as does the DIF. However, with the ONMNT the transform effectively transposes the kernel, which essentially rules out the ability to use the same transformation technique for the inverse. As such, the inverse ONMNT must therefore rely upon the DIF algorithm. It too breaks the N-length transform into two smaller $\frac{N}{2}$-length transforms but this time representing left and right sequences. This is so represented by the indexing, which is $x(n)$ for the left part and $x(n+\frac{N}{2})$ for the right part.

The DIF starts with a variation of equation (4.1)

$$X(k) = \left\langle \sum_{n=0}^{N-1} x(n)\beta\left(\frac{2n+1}{2}k\right)\right\rangle_{Mp} \tag{4.21}$$

and is split into left and right components where

$$X(k) = X^{left}(2k) + X^{right}(2k+1), \tag{4.22}$$

which expands to

$$X\left(k\right)=\left\langle \sum_{n=0}^{\frac{N}{2}-1}x(n)\beta\left(\frac{2n+1}{2}k\right)+\sum_{n=\frac{N}{2}}^{N-1}x(n)\beta\left(\frac{2n+1}{2}k\right)\right\rangle_{Mp} \tag{4.23}$$

noting that where we previously used $\beta\left(\frac{2k+1}{2}n\right)$ in DIT, we swap n and k to obtain $\beta\left(\frac{2n+1}{2}k\right)$ as the direct algorithm produces a transposed kernel, so we need to invert the transpose during the inverse procedure through the DIF. We can now begin to use a common sequence where

$$X\left(k\right)=\left\langle \sum_{n=0}^{\frac{N}{2}-1}x(n)\beta\left(\frac{2n+1}{2}k\right)+\sum_{n=0}^{\frac{N}{2}-1}x\left(n+\frac{N}{2}\right)\beta\left[\frac{2\left(n+\frac{N}{2}\right)+1}{2}k\right]\right\rangle_{Mp}. \tag{4.24}$$

The next step is to solve the last term of β so

$$\beta\left[\frac{2\left(n+\frac{N}{2}\right)+1}{2}k\right]=\beta\left(\frac{N+2n+1}{2}k\right) \tag{4.25}$$

and subsequently

$$\beta\left(\frac{N+2n+1}{2}k\right)=\beta\left(\frac{Nk}{2}+\frac{2n+1}{2}k\right). \tag{4.26}$$

Applying the identity from [7], equation (4.26) can be written as

$$\beta_1\left(\frac{Nk}{2}\right)\beta\left(\frac{2n+1}{2}k\right)+\beta_2\left(\frac{Nk}{2}\right)\beta\left(-\frac{2n+1}{2}k\right). \tag{4.27}$$

By combining (4.24) and (4.27) we then get

$$X\left(k\right)=\left\langle \sum_{n=0}^{\frac{N}{2}-1}\left[x(n)+\beta_1\left(\frac{Nk}{2}\right)x\left(n+\frac{N}{2}\right)+\beta_2\left(\frac{Nk}{2}\right)x(N-n)\right]\beta\left(\frac{2n+1}{2}k\right)\right\rangle_{Mp}. \tag{4.28}$$

Assessing β_2 in (4.28) it is clear to see that due to the periodicity of $\frac{N}{2}$ and that for all Nk then $\beta_2(\frac{Nk}{2})=0$. The final term cancels out leaving

$$X\left(k\right)=\left\langle \sum_{n=0}^{\frac{N}{2}-1}\left[x(n)+\beta_1\left(\frac{Nk}{2}\right)x\left(n+\frac{N}{2}\right)\right]\beta\left(\frac{2n+1}{2}k\right)\right\rangle_{Mp}. \tag{4.29}$$

Substituting $2k$ and $2k+1$ into (4.29) and replacing k,

$$X\left(2k\right)=\left\langle \sum_{n=0}^{\frac{N}{2}-1}\left[x(n)+\beta_1\left(\frac{2Nk}{2}\right)x\left(n+\frac{N}{2}\right)\right]\beta\left(\frac{2n+1}{2}2k\right)\right\rangle_{Mp} \tag{4.30}$$

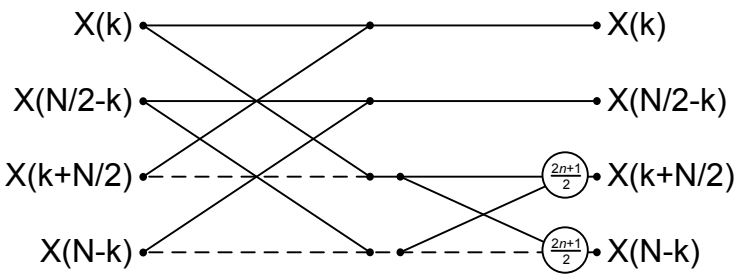


Figure 4.2: Radix-2 1D-ONMNT In-Place DIF Butterfly

where $\beta_1\left(\frac{2Nk}{2}\right) = 1$ producing

$$X\left(2k\right) = \left\langle \sum_{n=0}^{\frac{N}{2}-1} \left[x(n) + x\left(n + \frac{N}{2}\right)\right] \beta\left(\frac{2n+1}{2} 2k\right) \right\rangle_{Mp} \tag{4.31}$$

and

$$X\left(2k+1\right) = \left\langle \sum_{n=0}^{\frac{N}{2}-1} \left\{x(n) + \beta_1\left[\frac{N(2k+1)}{2}\right] x\left(n + \frac{N}{2}\right)\right\} \beta\left[\frac{2n+1}{2}(2k+1)\right] \right\rangle_{Mp}. \tag{4.32}$$

Using the identities in [7] equation (4.32) results in

$$X\left(2k+1\right) = \left\langle \sum_{n=0}^{\frac{N}{2}-1} \left[x(n) - x\left(n + \frac{N}{2}\right)\right] \beta\left(\frac{2n+1}{2} 2k + \frac{2n+1}{2}\right) \right\rangle_{Mp} \tag{4.33}$$

and finally produces

$$\begin{aligned} X\left(2k+1\right) = \Bigg\langle \sum_{n=0}^{\frac{N}{2}-1} \Bigg\{ & \beta_1\left(\frac{2n+1}{2}\right)\left[x(n) - x\left(n + \frac{N}{2}\right)\right] \\ & + \beta_2\left(\frac{2n+1}{2}\right)\left[x\left(\frac{N}{2} - n\right) - x\left(N - n\right)\right] \Bigg\} \beta\left(\frac{2n+1}{2} 2k\right) \Bigg\rangle_{Mp}. \end{aligned} \tag{4.34}$$

The radix-2 ONMNT DIF butterfly is shown in Figure 4.2.

4.3 Radix-4 ONMNT

The main idea of radix-4 is to split the transform into four discrete parts and process them at the same time. These attributes provide additional improvements in terms of complexity and subsequently speed. However, the downside is that only lengths that are derived as a power of four may be used. Therefore sequence lengths of 8, 32, 128, 512, etc. cannot be processed using this method.

4.3.1 Radix-4 DIT

According to the radix-2 ONMNT, equation (4.1) can be decomposed into

$$\begin{aligned} X^{ev}(k) &= \left\langle \sum_{n=0}^{\frac{N}{2}-1} x(2n)\beta\left(\frac{2k+1}{2}2n\right)\right\rangle_{Mp} \\ &= X_{2n}(k) \end{aligned} \tag{4.35}$$

and

$$X^{od}(k) = \left\langle \sum_{n=0}^{\frac{N}{2}-1} x(2n+1)\beta\left[\frac{2k+1}{2}(2n+1)\right]\right\rangle_{Mp}. \tag{4.36}$$

Thus, when $2n$ turns into $4n$ and $4n+2$, $2n+1$ turns into $4n+1$ and $4n+3$, then we have the requisite points to derive a radix-4 ONMNT. Again, noting that X_{2n} is even indexing and X_{2n+1} is odd indexing, then

$$X_{2n}(k) = \left\langle \sum_{n=0}^{\frac{N}{4}-1} x(4n)\beta\left(\frac{2k+1}{2}4n\right) + \sum_{n=0}^{\frac{N}{4}-1} x(4n+2)\beta\left[\frac{2k+1}{2}(4n+2)\right]\right\rangle_{Mp}. \tag{4.37}$$

Decomposing the final β term in (4.37) produces

$$\beta\left[\frac{2k+1}{2}(4n+2)\right] = \beta\left[2n(2k+1)+(2k+1)\right] \tag{4.38}$$

and further decomposition produces

$$\beta\left[2n(2k+1)+(2k+1)\right] = \beta_1(2k+1)\,\beta\left(\frac{2k+1}{2}4n\right) + \beta_2(2k+1)\,\beta\left(-\frac{2k+1}{2}4n\right). \tag{4.39}$$

Applying (4.39) into (4.37) then produces the even part

$$\begin{aligned} X_{2n}(k) = \Bigg\langle & \sum_{n=0}^{\frac{N}{4}-1} x(4n)\beta\left(\frac{2k+1}{2}4n\right) \\ & + \beta_1\left(\frac{2k+1}{2}2\right)\sum_{n=0}^{\frac{N}{4}-1} x(4n+2)\beta\left(\frac{2k+1}{2}4n\right) \\ & + \beta_2\left(\frac{2k+1}{2}2\right)\sum_{n=0}^{\frac{N}{4}-1} x(4n+2)\beta\left(-\frac{2k+1}{2}4n\right)\Bigg\rangle_{Mp}. \end{aligned} \tag{4.40}$$

In order to simplify (4.40) we define

$$X_{4n}(k) = \left\langle \sum_{n=0}^{\frac{N}{4}-1} x(4n)\beta\left(\frac{2k+1}{2}4n\right)\right\rangle_{Mp} \tag{4.41}$$

and

$$X_{4n}\left(N-k\right)=\left\langle \sum_{n=0}^{\frac{N}{4}-1}x(4n)\beta\left(-\frac{2k+1}{2}4n\right)\right\rangle_{Mp}. \tag{4.42}$$

Applying (4.41) and (4.42) to (4.40) now produces

$$X_{2n}\left(k\right)=X_{4n}\left(k\right)+\beta_1\left(\frac{2k+1}{2}\times 2\right)X_{4n+2}\left(k\right)+\beta_2\left(\frac{2k+1}{2}\times 2\right)X_{4n+2}\left(N-k\right). \tag{4.43}$$

The odd part is denoted as

$$\begin{aligned}X^{od}\left(k\right)=&\left\langle \sum_{n=0}^{\frac{N}{4}-1}x(4n+1)\beta\left[\frac{2k+1}{2}\left(4n+1\right)\right]\right.\\&\left.+\sum_{n=0}^{\frac{N}{4}-1}x(4n+3)\beta\left[\frac{2k+1}{2}\left(4n+3\right)\right]\right\rangle_{Mp}.\end{aligned} \tag{4.44}$$

Simplifying both β terms in (4.44), produces

$$\begin{aligned}\beta\left[\frac{2k+1}{2}\left(4n+1\right)\right]&=\beta\left(\frac{2k+1}{2}4n+\frac{2k+1}{2}\right)\\&=\beta_1\left(\frac{2k+1}{2}\right)\beta\left(\frac{2k+1}{2}4n\right)\\&\quad+\beta_2\left(\frac{2k+1}{2}\right)\beta\left(-\frac{2k+1}{2}4n\right)\end{aligned} \tag{4.45}$$

and

$$\begin{aligned}\beta\left[\frac{2k+1}{2}\left(4n+3\right)\right]&=\beta\left(\frac{2k+1}{2}4n+\frac{2k+1}{2}\times 3\right)\\&=\beta_1\left(\frac{2k+1}{2}\times 3\right)\beta\left(\frac{2k+1}{2}4n\right)\\&\quad+\beta_2\left(\frac{2k+1}{2}\times 3\right)\beta\left(-\frac{2k+1}{2}4n\right).\end{aligned} \tag{4.46}$$

Applying equations (4.45) and (4.46) into (4.44) produces

$$\begin{aligned}X^{od}\left(k\right)=\Bigg\langle&\beta_1\left(\frac{2k+1}{2}\right)\sum_{n=0}^{\frac{N}{4}-1}x(4n+1)\beta\left(\frac{2k+1}{2}4n\right)\\&+\beta_2\left(\frac{2k+1}{2}\right)\sum_{n=0}^{\frac{N}{4}-1}x(4n+1)\beta\left(-\frac{2k+1}{2}4n\right)\\&+\beta_1\left(\frac{2k+1}{2}\times 3\right)\sum_{n=0}^{\frac{N}{4}-1}x(4n+3)\beta\left(\frac{2k+1}{2}4n\right)\\&+\beta_2\left(\frac{2k+1}{2}\times 3\right)\sum_{n=0}^{\frac{N}{4}-1}x(4n+3)\beta\left(-\frac{2k+1}{2}4n\right)\Bigg\rangle_{Mp}.\end{aligned} \tag{4.47}$$

Similar to (4.41), define

$$X_{4n+1}(k) = \left\langle \sum_{n=0}^{\frac{N}{4}-1} x(4n+1)\beta\left(\frac{2k+1}{2}4n\right)\right\rangle_{Mp} \tag{4.48}$$

and

$$X_{4n+3}(k) = \left\langle \sum_{n=0}^{\frac{N}{4}-1} x(4n+3)\beta\left(\frac{2k+1}{2}4n\right)\right\rangle_{Mp} \tag{4.49}$$

then by applying (4.48) and (4.49) into (4.47) we get the odd part

$$\begin{aligned} X^{od}(k) = \Bigg\langle & \beta_1\left(\frac{2k+1}{2}\right)X_{4n+1}(k) + \beta_2\left(\frac{2k+1}{2}\right)X_{4n+1}(N-k) \\ & + \beta_1\left(\frac{2k+1}{2}\times 3\right)X_{4n+3}(k) + \beta_2\left(\frac{2k+1}{2}\times 3\right)X_{4n+3}(N-k)\Bigg\rangle_{Mp}. \end{aligned} \tag{4.50}$$

The equation of ONMNT radix-4 at point k is

$$X(k) = X^{ev}(k) + X^{od}(k) \tag{4.51}$$

and in the decomposed form becomes

$$\begin{aligned} X(k) = \Bigg\langle & X_{4n}(k) + \beta_1\left(\frac{2k+1}{2}\times 2\right)X_{4n+2}(k) + \beta_2\left(\frac{2k+1}{2}\times 2\right)X_{4n+2}(N-k) \\ & + \beta_1\left(\frac{2k+1}{2}\right)X_{4n+1}(k) + \beta_2\left(\frac{2k+1}{2}\right)X_{4n+1}(N-k) \\ & + \beta_1\left(\frac{2k+1}{2}\times 3\right)X_{4n+3}(k) + \beta_2\left(\frac{2k+1}{2}\times 3\right)X_{4n+3}(N-k)\Bigg\rangle_{Mp}. \end{aligned} \tag{4.52}$$

In radix-4 there are eight points: k, $\frac{N}{4}-k$, $k+\frac{N}{4}$, $\frac{N}{2}-k$, $k+\frac{N}{2}$, $\frac{3N}{4}-k$, $k+\frac{3N}{4}$ and $N-k$. Equations (4.51) and (4.52) show the point k.

Similarly, it can be easy to get all equations at different point by replacing k. For example point $\frac{N}{4}-k$, $\because \frac{N}{4}-k :\Leftrightarrow \frac{N}{4}-k-1$ by definition, then

$$X_{4n}(k) \xrightarrow{k=\frac{N}{4}-k} X_{4n}\left(\tfrac{N}{4}-k\right) = \left\langle \sum_{n=0}^{\frac{N}{4}-1} x(4n)\beta\left[\frac{2\left(\frac{N}{4}-k-1\right)+1}{2}4n\right]\right\rangle_{Mp}. \tag{4.53}$$

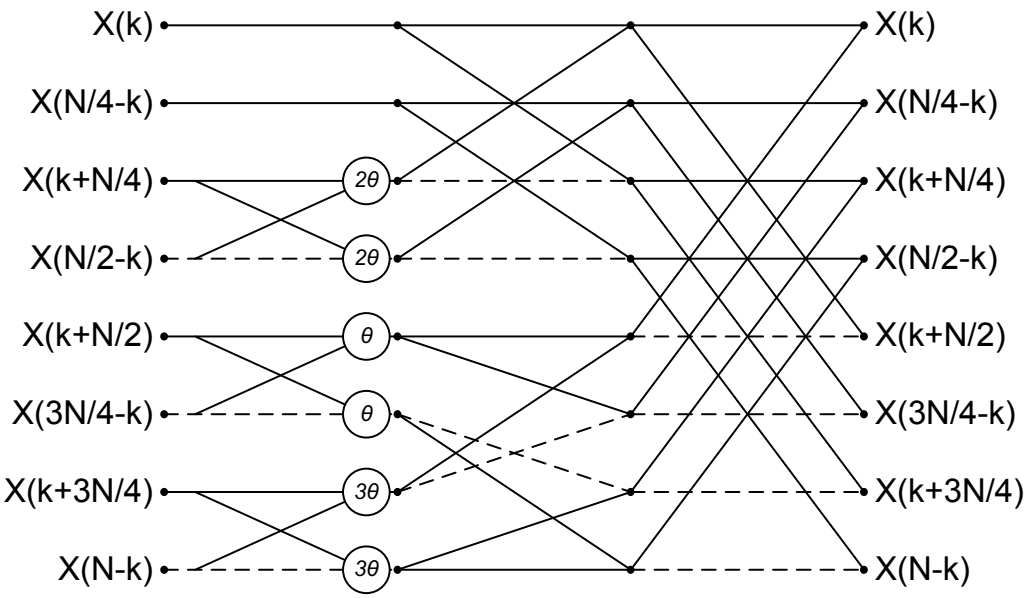


Figure 4.3: Radix-4 1D-ONMNT In-Place DIT Butterfly

Decomposing the β term in (4.53) produces

$$\begin{aligned}\beta\left[\frac{2\left(\frac{N}{4}-k-1\right)+1}{2}4n\right] &= \beta\left(\frac{\frac{N}{2}-2k-1}{2}4n\right)\\ &= \beta\left(-\frac{2k+1}{2}4n\right)\end{aligned} \tag{4.54}$$

We can identify the following β identities:

$$\beta_1\left[\frac{2\left(\frac{N}{4}-k-1\right)+1}{2}\times 2\right] = -\beta_1\left(\frac{2k+1}{2}\times 2\right) \tag{4.55}$$

$$\beta_2\left[\frac{2\left(\frac{N}{4}-k-1\right)+1}{2}\times 2\right] = \beta_2\left(\frac{2k+1}{2}\times 2\right) \tag{4.56}$$

$$\beta_1\left[\frac{2\left(\frac{N}{4}-k-1\right)+1}{2}\right] = \beta_2\left(\frac{2k+1}{2}\right) \tag{4.57}$$

$$\beta_2\left[\frac{2\left(\frac{N}{4}-k-1\right)+1}{2}\right] = \beta_1\left(\frac{2k+1}{2}\right) \tag{4.58}$$

$$\beta_1\left[\frac{2\left(\frac{N}{4}-k-1\right)+1}{2}\times 3\right] = -\beta_2\left(\frac{2k+1}{2}\times 3\right) \tag{4.59}$$

and

$$\beta_2\left[\frac{2\left(\frac{N}{4}-k-1\right)+1}{2}\times 3\right] = -\beta_1\left(\frac{2k+1}{2}\times 3\right). \tag{4.60}$$

Substituting the identities (4.55)-(4.60) into (4.52) will derive the point $\frac{N}{4}-k$. The rest of the points can be shown as follows, where we substitute $\frac{2k+1}{2}$ with θ for ease of

reference

$$\begin{bmatrix} X(k) \\ X\left(\frac{N}{4}-k\right) \\ X\left(k+\frac{N}{4}\right) \\ X\left(\frac{N}{2}-k\right) \\ X\left(k+\frac{N}{2}\right) \\ X\left(\frac{3N}{4}-k\right) \\ X\left(k+\frac{3N}{4}\right) \\ X(N-k) \end{bmatrix} = \begin{bmatrix} A+B_2+B_1+B_3 \\ \overline{A}+\overline{B_2}+B_1-B_3 \\ A-B_2-\overline{B_1}+\overline{B_3} \\ \overline{A}-\overline{B_2}+\overline{B_1}+\overline{B_3} \\ A+B_2-B_1-B_3 \\ \overline{A}+\overline{B_2}-B_1+B_3 \\ A-B_2+\overline{B_1}-\overline{B_3} \\ \overline{A}-\overline{B_2}-\overline{B_1}-\overline{B_3} \end{bmatrix} \tag{4.61}$$

where

$$A = X_{4n}(k) \tag{4.62}$$

$$\overline{A} = X_{4n}\left(\tfrac{N}{4}-k\right) \tag{4.63}$$

$$B_m = \left\langle \left[\beta_1(\theta m)\, X_{4n+m}(k) + \beta_2(\theta m)\, X_{4n+m}\left(\tfrac{N}{4}-k\right)\right]\right\rangle_{Mp} \tag{4.64}$$

and

$$\overline{B_m} = \left\langle \left[\beta_2(\theta m)\, X_{4n+m}(k) - \beta_1(\theta m)\, X_{4n+m}\left(\tfrac{N}{4}-k\right)\right]\right\rangle_{Mp} \tag{4.65}$$

for $m \in \{1,2,3\}$. The radix-4 ONMNT DIT butterfly is shown in Figure 4.3.

4.3.2 Radix-4 DIF

Using the variation of the original algorithm shown in (4.21), the whole sequence is evenly divided into sequences that are of length $\frac{N}{4}$ as

$$\begin{aligned} X(k) = \Bigg\langle & \sum_{n=0}^{\frac{N}{4}-1} x(n)\beta\left(\frac{2n+1}{2}k\right) + \sum_{n=\frac{N}{4}}^{\frac{N}{2}-1} x(n)\beta\left(\frac{2n+1}{2}k\right) \\ & + \sum_{n=\frac{N}{2}}^{\frac{3N}{4}-1} x(n)\beta\left(\frac{2n+1}{2}k\right) + \sum_{n=\frac{3N}{4}}^{N-1} x(n)\beta\left(\frac{2n+1}{2}k\right)\Bigg\rangle_{Mp}. \end{aligned} \tag{4.66}$$

Adjusting the range in (4.66) we then obtain

$$\begin{aligned} X(k) = \Bigg\langle \sum_{n=0}^{\frac{N}{4}-1} \Bigg\{ & x(n)\beta\left(\frac{2n+1}{2}k\right) + x\left(n+\frac{N}{4}\right)\beta\left[\frac{2\left(n+\frac{N}{4}\right)+1}{2}k\right] \\ & + x\left(n+\frac{2N}{4}\right)\beta\left[\frac{2\left(n+\frac{2N}{4}\right)+1}{2}k\right] + x\left(n+\frac{3N}{4}\right)\beta\left[\frac{2\left(n+\frac{3N}{4}\right)+1}{2}k\right]\Bigg\}\Bigg\rangle_{Mp}. \end{aligned} \tag{4.67}$$

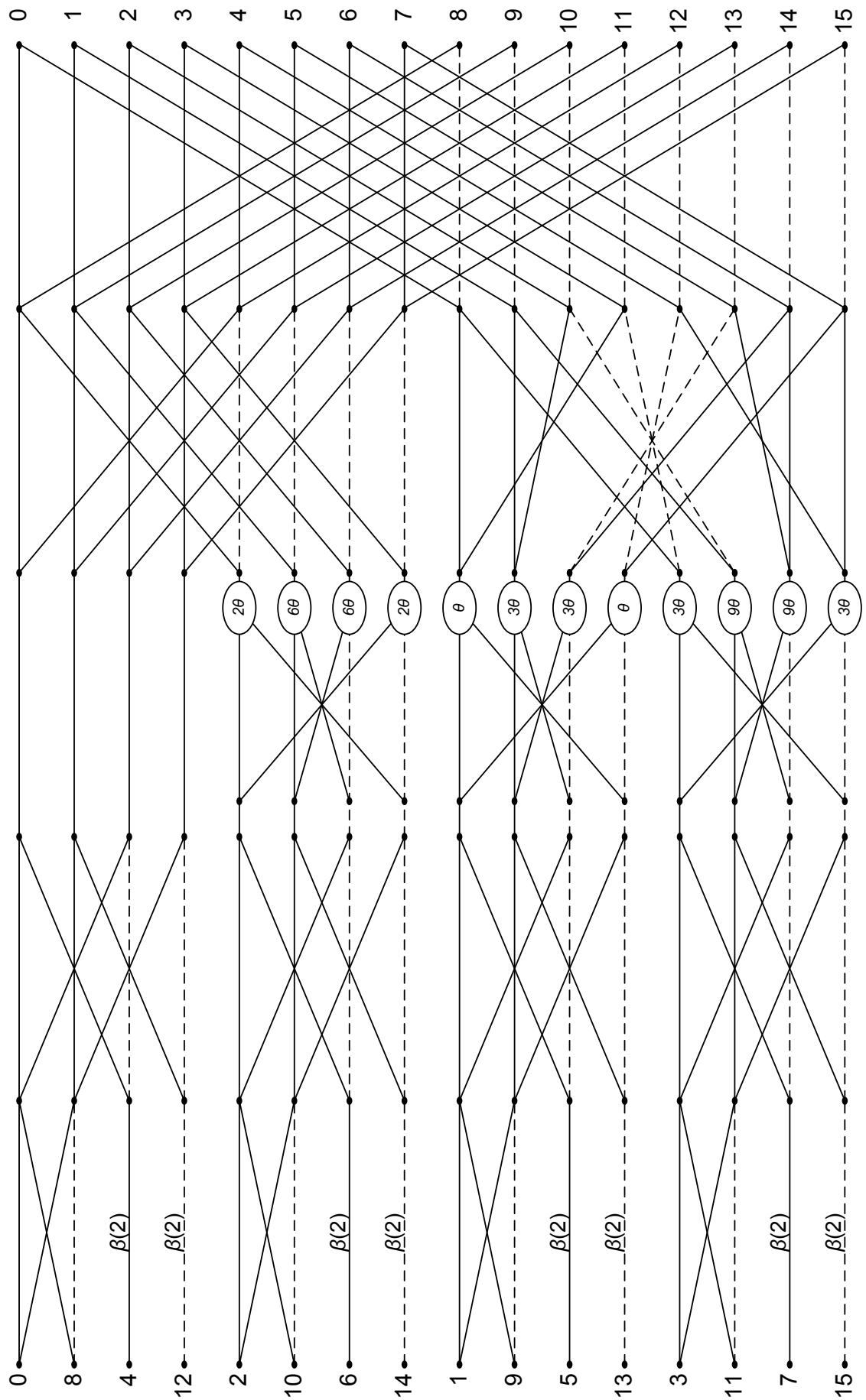


Figure 4.4: Radix-4 In-Place DIT Flow Diagram, $N = 16$

and can be simplified as

$$X(k) = \left\langle \sum_{n=0}^{\frac{N}{4}-1} \left[x(n)\beta\left(\frac{2n+1}{2}k\right) + x\left(n+\frac{N}{4}\right)\beta\left(\frac{2n+1}{2}k+\frac{N}{4}k\right) + x\left(n+\frac{N}{2}\right)\beta\left(\frac{2n+1}{2}k+\frac{N}{2}k\right) + x\left(n+\frac{3N}{4}\right)\beta\left(\frac{2n+1}{2}k+\frac{3N}{4}k\right)\right]\right\rangle_{Mp}. \tag{4.68}$$

The β terms in (4.68) can decomposed as

$$\beta\left(\frac{2n+1}{2}k+\frac{N}{2}k\right) = \beta_1\left(\frac{N}{2}k\right)\beta\left(\frac{2n+1}{2}k\right) + \beta_2\left(\frac{N}{2}k\right)\beta\left(-\frac{2n+1}{2}k\right) \tag{4.69}$$

noting that the β_2 term in (4.69) will always equate to zero, we further obtain

$$\beta\left(\frac{2n+1}{2}k+\frac{N}{4}k\right) = \beta_1\left(\frac{N}{4}k\right)\beta\left(\frac{2n+1}{2}k\right) + \beta_2\left(\frac{N}{4}k\right)\beta\left(-\frac{2n+1}{2}k\right) \tag{4.70}$$

and

$$\beta\left(\frac{2n+1}{2}k+\frac{3N}{4}k\right) = \beta_1\left(\frac{3N}{4}k\right)\beta\left(\frac{2n+1}{2}k\right) + \beta_2\left(\frac{3N}{4}k\right)\beta\left(-\frac{2n+1}{2}k\right). \tag{4.71}$$

Substituting (4.69)-(4.71) into (4.68) produces

$$X(k) = \left\langle \sum_{n=0}^{\frac{N}{4}-1} \left[x(n) + \beta_1\left(\frac{N}{2}k\right)x\left(n+\frac{N}{2}\right) + \beta_1\left(\frac{N}{4}k\right)x\left(n+\frac{N}{4}\right) + \beta_2\left(\frac{N}{4}k\right)x\left(\frac{N}{2}-n\right) + \beta_1\left(\frac{3N}{4}k\right)x\left(n+\frac{3N}{4}\right) + \beta_2\left(\frac{3N}{4}k\right)x(N-n)\right]\beta\left(\frac{2n+1}{2}k\right)\right\rangle_{Mp}. \tag{4.72}$$

In radix-4, after reducing the length to $\frac{N}{4}$, we have even parts: $4k$, $4k+2$ and odd parts: $4k+1$ and $4k+3$. In terms of the even parts, we get the following identities similar to (4.55)-(4.60), by first substituting $4k$ into (4.72) replacing k:

$$\beta_1\left(\frac{N}{2}4k\right) = 1 \tag{4.73}$$

$$\beta_1\left(\frac{N}{4}4k\right) = 1 \tag{4.74}$$

$$\beta_2\left(\frac{N}{4}4k\right) = 0 \tag{4.75}$$

and $4k+2$ into (4.72) replacing k:

$$\beta_1\left[\frac{N}{2}(4k+2)\right]=1 \tag{4.76}$$

$$\beta_1\left[\frac{N}{4}(4k+2)\right]=-1 \tag{4.77}$$

and

$$\beta_2\left[\frac{N}{4}(4k+2)\right]=0. \tag{4.78}$$

Putting equations (4.73)-(4.75) back into (4.72) we get

$$\begin{aligned} X(4k)=\Bigg\langle\sum_{n=0}^{\frac{N}{4}-1}\Bigg[&x(n)+x\left(n+\frac{N}{2}\right)\\ &+x\left(n+\frac{N}{4}\right)+x\left(n+\frac{3N}{4}\right)\Bigg]\beta\left(\frac{2n+1}{2}4k\right)\Bigg\rangle_{Mp} \end{aligned} \tag{4.79}$$

and also replacing equations (4.76)-(4.78) we get

$$\begin{aligned} X(4k+2)&=\Bigg\langle\sum_{n=0}^{\frac{N}{4}-1}\Bigg[x(n)+x\left(n+\frac{N}{2}\right)\\ &\quad-x\left(n+\frac{N}{4}\right)-x\left(n+\frac{3N}{4}\right)\Bigg]\beta\left(\frac{2n+1}{2}4k+\frac{2n+1}{2}\times 2\right)\Bigg\rangle_{Mp}\\ &=\Bigg\langle\sum_{n=0}^{\frac{N}{4}-1}\Bigg\{\beta_1\left(\frac{2n+1}{2}\times 2\right)\left[x(n)+x\left(n+\frac{N}{2}\right)\right.\\ &\quad\left.-x\left(n+\frac{N}{4}\right)-x\left(n+\frac{3N}{4}\right)\right]\\ &\quad+\beta_2\left(\frac{2n+1}{2}2\right)\left[x\left(\frac{N}{4}-n\right)+x\left(\frac{3N}{4}-n\right)\right.\\ &\quad\left.-x\left(\frac{N}{2}-n\right)-x\left(N-n\right)\right]\Bigg\}\beta\left(\frac{2n+1}{2}4k\right)\Bigg\rangle_{Mp}. \end{aligned} \tag{4.80}$$

In terms of the odd parts, we get the following identities similar to (4.73)-(4.78), by first substituting $4k+1$ again into (4.72) replacing k:

$$\beta_1\left[\frac{N}{2}(4k+1)\right]=-1 \tag{4.81}$$

$$\beta_1\left[\frac{N}{4}(4k+1)\right]=0 \tag{4.82}$$

$$\beta_2\left[\frac{N}{4}(4k+1)\right]=1 \tag{4.83}$$

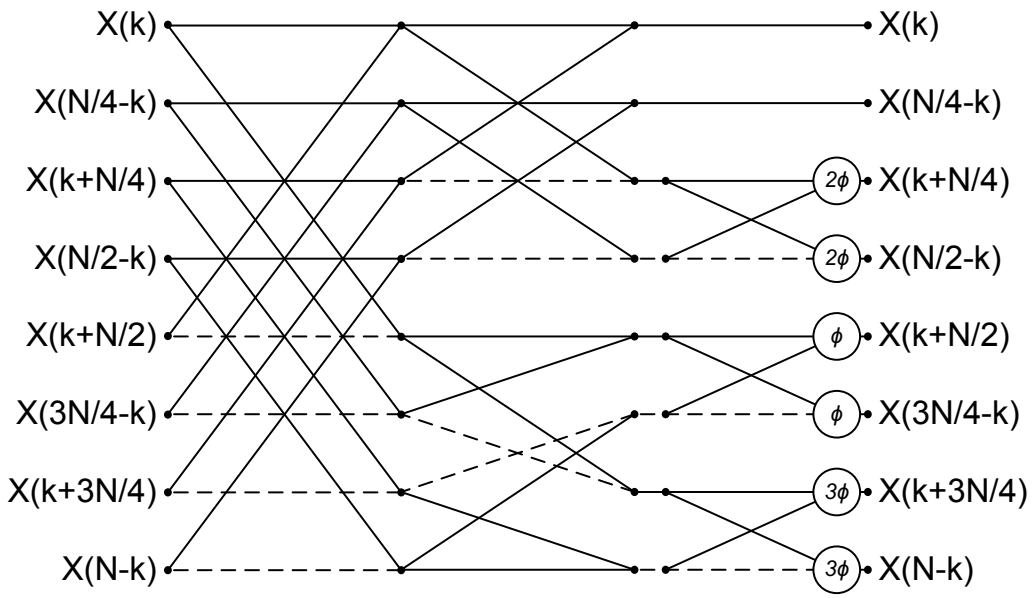


Figure 4.5: Radix-4 1D-ONMNT In-Place DIF Butterfly

and finally $4k+3$ into (4.72) replacing k:

$$\beta_1 \left[\frac{N}{2}(4k+3)\right] = -1 \tag{4.84}$$

$$\beta_1 \left[\frac{N}{4}(4k+3)\right] = 0 \tag{4.85}$$

$$\beta_2 \left[\frac{3N}{4}(4k+3)\right] = 1. \tag{4.86}$$

Putting equations (4.81)-(4.83) back into (4.72) we get

$$\begin{aligned}
X\left(4k+1\right) = \Bigg\langle \sum_{n=0}^{\frac{N}{4}-1} \Bigg\{ & x(n)\beta\left[\frac{2n+1}{2}\left(4k+1\right)\right] \\
& + x\left(n+\frac{N}{4}\right)\beta\left[\frac{2\left(n+\frac{N}{4}\right)+1}{2}\left(4k+1\right)\right] \\
& + x\left(n+\frac{N}{2}\right)\beta\left[\frac{2\left(n+\frac{N}{2}\right)+1}{2}\left(4k+1\right)\right] \\
& + x\left(n+\frac{3N}{4}\right)\beta\left[\frac{2\left(n+\frac{3N}{4}\right)+1}{2}\left(4k+1\right)\right]\Bigg\}\Bigg\rangle_{Mp} \\
= \Bigg\langle \sum_{n=0}^{\frac{N}{4}-1} \Bigg\{ & \beta_1\left(\frac{2n+1}{2}\right)\Bigg[x(n) - x\left(n+\frac{N}{2}\right) \\
& + x\left(\frac{N}{4}-n\right) - x\left(\frac{3N}{4}-n\right)\Bigg] \\
& + \beta_2\left(\frac{2n+1}{2}\right)\Bigg[x\left(\frac{N}{2}-n\right) - x\left(N-n\right) \\
& - x\left(n+\frac{N}{4}\right) + x\left(n+\frac{3N}{4}\right)\Bigg]\Bigg\}\beta\left(\frac{2n+1}{2}4k\right)\Bigg\rangle_{Mp}.
\end{aligned} \tag{4.87}$$

and replacing (4.84)-(4.86) into (4.72) we get

$$\begin{aligned}
X\left(4k+3\right) = \Bigg\langle \sum_{n=0}^{\frac{N}{4}-1} \Bigg\{ & x(n)\beta\left[\frac{2n+1}{2}\left(4k+3\right)\right] \\
& + x\left(n+\frac{N}{4}\right)\beta\left[\frac{2\left(n+\frac{N}{4}\right)+1}{2}\left(4k+3\right)\right] \\
& + x\left(n+\frac{N}{2}\right)\beta\left[\frac{2\left(n+\frac{N}{2}\right)+1}{2}\left(4k+3\right)\right] \\
& + x\left(n+\frac{3N}{4}\right)\beta\left[\frac{2\left(n+\frac{3N}{4}\right)+1}{2}\left(4k+3\right)\right]\Bigg\}\Bigg\rangle_{Mp} \\
= \Bigg\langle \sum_{n=0}^{\frac{N}{4}-1} \Bigg\{ & \beta_1\left(\frac{2n+1}{2}3\right)\Bigg[x(n) - x\left(n+\frac{N}{2}\right) \\
& - x\left(\frac{N}{4}-n\right) + x\left(\frac{3N}{4}-n\right)\Bigg] \\
& + \beta_2\left(\frac{2n+1}{2}3\right)\Bigg[x\left(\frac{N}{2}-n\right) - x\left(N-n\right) \\
& + x\left(n+\frac{N}{4}\right) - x\left(n+\frac{3N}{4}\right)\Bigg]\Bigg\}\beta\left(\frac{2n+1}{2}4k\right)\Bigg\rangle_{Mp}.
\end{aligned} \tag{4.88}$$

The equations of the radix-4 ONMNT DIF can be shown as a condensed representation, similar to equations (4.61)-(4.65) but this time substituting $\frac{2n+1}{2}$ with ϕ for ease of reference

$$\begin{bmatrix} X(4k) \\ X(4k+1) \\ X(4k+2) \\ X(4k+3) \end{bmatrix} = \beta_1(\phi m) \begin{bmatrix} A+B \\ A-B \\ \overline{A}+\overline{C} \\ \overline{A}-\overline{C} \end{bmatrix} + \beta_2(\phi m) \begin{bmatrix} C+D \\ C-D \\ \overline{D}-\overline{B} \\ \overline{D}+\overline{B} \end{bmatrix} \tag{4.89}$$

where

$$A = x(n) + x\left(n + \frac{N}{2}\right) \tag{4.90}$$

$$\overline{A} = x(n) - x\left(n + \frac{N}{2}\right) \tag{4.91}$$

$$B = x\left(n + \frac{N}{4}\right) + x\left(n + \frac{3N}{4}\right) \tag{4.92}$$

$$\overline{B} = x\left(n + \frac{N}{4}\right) - x\left(n + \frac{3N}{4}\right) \tag{4.93}$$

$$C = x\left(\frac{N}{4} - n\right) + x\left(\frac{3N}{4} - n\right) \tag{4.94}$$

$$\overline{C} = x\left(\frac{N}{4} - n\right) - x\left(\frac{3N}{4} - n\right) \tag{4.95}$$

$$D = x\left(\frac{N}{2} - n\right) + x(N-n) \tag{4.96}$$

$$\overline{D} = x\left(\frac{N}{2} - n\right) - x(N-n). \tag{4.97}$$

and m is derived from $X(4k+m)$ for $m \in \{0,1,2,3\}$. The radix-4 ONMNT DIF butterfly is shown in Figure 4.5.

4.4 Split-Radix ONMNT

The split-radix, which is sometimes referred to as radix-2/4, combines the versatility of the radix-2 with the efficiency of the radix-4 and manages to further reduce the number of operations in the process. It accomplishes this by using L shaped butterflies, which are based on radix-4 that interlock against each other until two points remain. These L shaped butterflies can have either a single twiddle stage or a dual twiddle stage, which are denoted in Figure 4.6. When there is no further room to implement an L-butterfly, which typically encompasses two sub-stages of butterflies and a sub-stage for the twiddle, a radix-2 butterfly is then used.

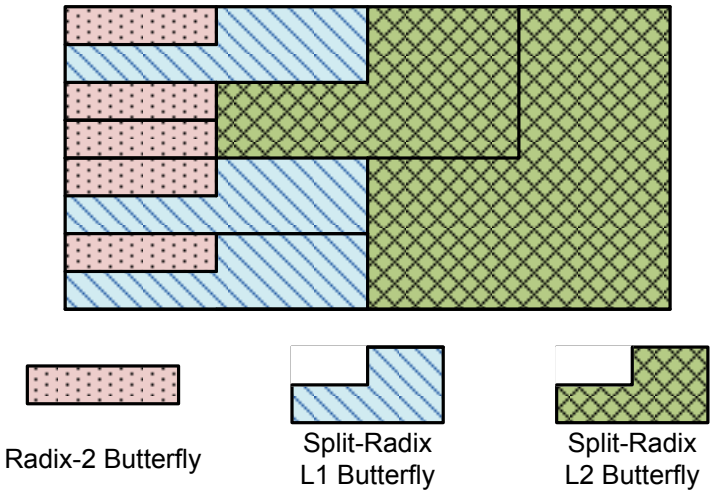


Figure 4.6: Split-Radix DIT Structure

This section will show how combining algorithms from the radix-2 and radix-4 implementations can derive the split-radix DIT and DIF algorithms.

4.4.1 Split-Radix DIT

The split-radix combines radix-2 and radix-4, specifically the even part of radix-2 and the odd parts of radix-4. Therefore we decompose the initial ONMNT algorithm in (4.1) into

$$X\left(k\right) = X^{ev}\left(k\right) + X^{od}\left(k\right) \tag{4.98}$$

where

$$X^{ev}\left(k\right) = \left\langle \sum_{n=0}^{\frac{N}{2}-1} x(2n)\beta\left(\frac{2k+1}{2}2n\right)\right\rangle_{Mp} \tag{4.99}$$

and

$$\begin{aligned} X^{od}\left(k\right) = \Bigg\langle & \sum_{n=0}^{\frac{N}{4}-1} x(4n+1)\beta\left[(4n+1)\,\frac{2k+1}{2}\right] \\ & + \sum_{n=0}^{\frac{N}{4}-1} x(4n+3)\beta\left[(4n+3)\,\frac{2k+1}{2}\right]\Bigg\rangle_{Mp}. \end{aligned} \tag{4.100}$$

Defining point $X_{2n}(k)$ as

$$X_{2n}\left(k\right) = \left\langle \sum_{n=0}^{\frac{N}{2}-1} x(2n)\beta\left(\frac{2k+1}{2}2n\right)\right\rangle_{Mp} \tag{4.101}$$

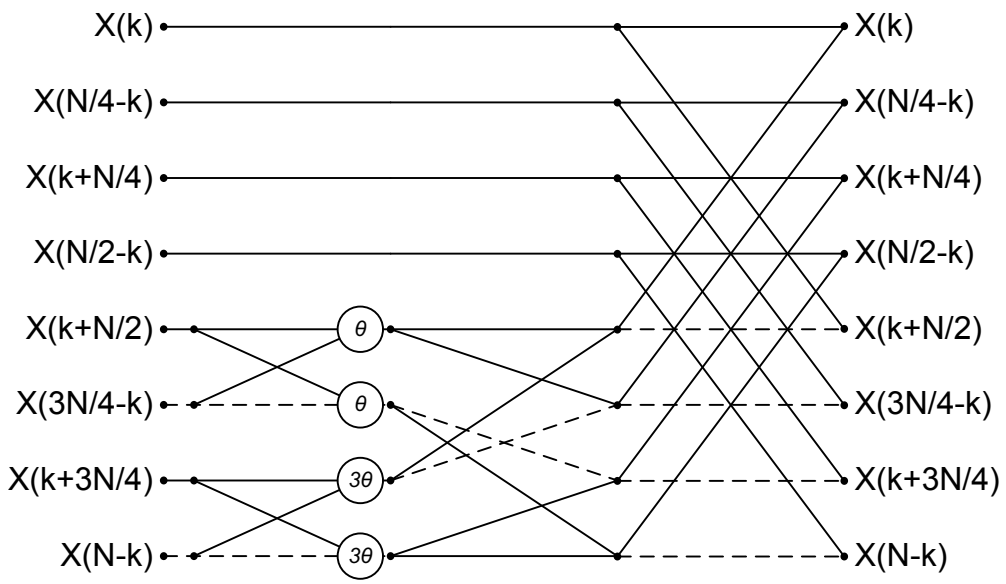


Figure 4.7: Split-Radix 1D-ONMNT In-Place DIT Butterfly

point $X_{4n+1}(k)$ as

$$X_{4n+1}(k) = \left\langle \sum_{n=0}^{\frac{N}{4}-1} x(4n+1)\beta\left(\frac{2k+1}{2}4n\right)\right\rangle_{Mp} \tag{4.102}$$

and point $X_{4n+3}(k)$ as

$$X_{4n+3}(k) = \left\langle \sum_{n=0}^{\frac{N}{4}-1} x(4n+3)\beta\left(\frac{2k+1}{2}4n\right)\right\rangle_{Mp} \tag{4.103}$$

we can then apply (4.101) into (4.99) and (4.102)-(4.103) into (4.100) to produce the recursive equation

$$\begin{aligned} X(k) = \Bigg\langle & X_{2n}(k) + \beta_1\left(\frac{2k+1}{2}\right)X_{4n+1}(k) + \beta_2\left(\frac{2k+1}{2}\right)X_{4n+1}(k) \\ & + \beta_1\left(\frac{2k+1}{2}\times 3\right)X_{4n+3}(k) + \beta_2\left(\frac{2k+1}{2}\times 3\right)X_{4n+3}(k)\Bigg\rangle_{Mp}. \end{aligned} \tag{4.104}$$

Equations for all points of the split-radix ONMNT DIT butterfly can now be derived from

$$\begin{bmatrix} X(k) \\ X\left(\frac{N}{4}-k\right) \\ X\left(k+\frac{N}{4}\right) \\ X\left(\frac{N}{2}-k\right) \\ X\left(k+\frac{N}{2}\right) \\ X\left(\frac{3N}{4}-k\right) \\ X\left(k+\frac{3N}{4}\right) \\ X(N-k) \end{bmatrix} = \begin{bmatrix} A+B_1+B_3 \\ \overline{A}+B_1-B_3 \\ A-\overline{B_1}+\overline{B_3} \\ \overline{A}+\overline{B_1}+\overline{B_3} \\ A-B_1-B_3 \\ \overline{A}-\overline{B_1}+\overline{B_3} \\ A+\overline{B_1}-\overline{B_3} \\ \overline{A}-\overline{B_1}-\overline{B_3} \end{bmatrix} \tag{4.105}$$

where

$$A = X_{2n}(k) \tag{4.106}$$

$$\overline{A} = X_{2n}\left(\tfrac{N}{2}-k\right) \tag{4.107}$$

$$B_m = \left\langle \left[\beta_1(\theta m)\, X_{4n+m}(k) + \beta_2(\theta m)\, X_{4n+m}\left(\tfrac{N}{4}-k\right)\right]\right\rangle_{Mp} \tag{4.108}$$

$$\overline{B_m} = \left\langle \left[\beta_1(\theta m)\, X_{4n+m}(k) - \beta_2(\theta m)\, X_{4n+m}\left(\tfrac{N}{4}-k\right)\right]\right\rangle_{Mp} \tag{4.109}$$

and $m \in \{1,3\}$. The split-radix ONMNT DIT butterfly is shown in Figure 4.7.

4.4.2 Split-Radix DIF

Similar to the DIT split-radix, we initially start from (4.1) and subsequently derive

$$X(k) = X^{left}(k) + X^{right}(k) \tag{4.110}$$

where $X^{left}(k)$ is the left part. More specifically it comes from the radix-2 ONMNT DIF. Additionally, $X^{right}(k)$ contains the right parts of the radix-4 ONMNT DIF. Thus, the left sequence that is first shown in (4.31) as

$$X(2k) = \left\langle \sum_{n=0}^{\frac{N}{2}-1} \left[x(n) + x\left(n+\frac{N}{2}\right)\right] \beta\left(\frac{2n+1}{2}2k\right)\right\rangle_{Mp} \tag{4.111}$$

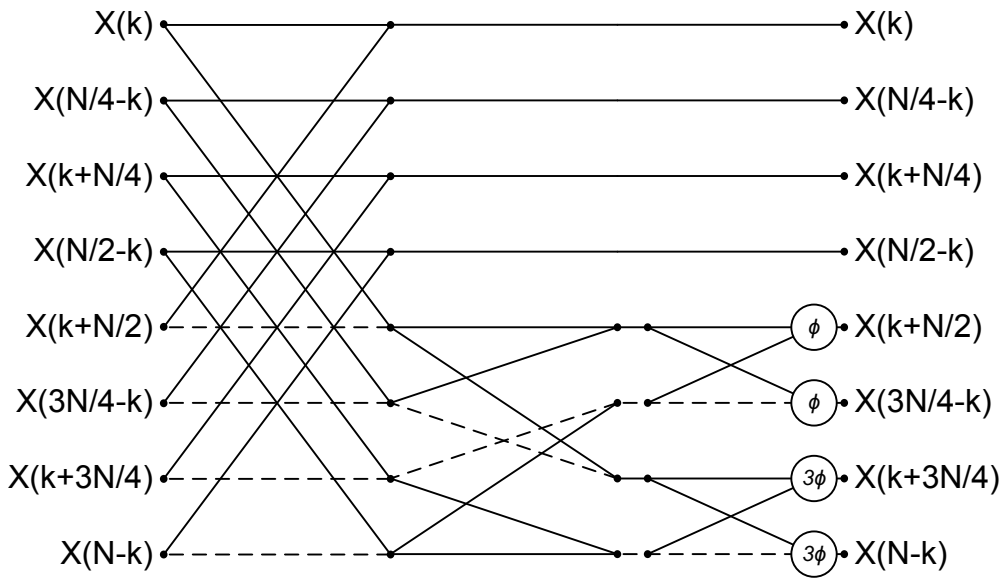


Figure 4.8: Split-Radix 1D-ONMNT In-Place DIF Butterfly

Similarly, the right parts of the radix-4 ONMNT DIF (4.87) and (4.88) as

$$\begin{aligned} X\left(4k+1\right) = \Bigg\langle \sum_{n=0}^{\frac{N}{4}-1} \Bigg\{ & \beta_1 \left(\frac{2n+1}{2}\right) \left[x(n) - x\left(n+\frac{N}{2}\right) \right. \\ & \left. + x\left(\frac{N}{4}-n\right) - x\left(\frac{3N}{4}-n\right) \right] \\ & + \beta_2 \left(\frac{2n+1}{2}\right) \left[x\left(\frac{N}{2}-n\right) - x\left(N-n\right) \right. \\ & \left. - x\left(n+\frac{N}{4}\right) + x\left(n+\frac{3N}{4}\right) \right] \Bigg\} \beta \left(\frac{2n+1}{2} 4k\right) \Bigg\rangle_{Mp} \end{aligned} \tag{4.112}$$

and

$$\begin{aligned} X\left(4k+3\right) = \Bigg\langle \sum_{n=0}^{\frac{N}{4}-1} \Bigg\{ & \beta_1 \left(\frac{2n+1}{2} \times 3\right) \left[x(n) - x\left(n+\frac{N}{2}\right) \right. \\ & \left. - x\left(\frac{N}{4}-n\right) + x\left(\frac{3N}{4}-n\right) \right] \\ & + \beta_2 \left(\frac{2n+1}{2} \times 3\right) \left[x\left(\frac{N}{2}-n\right) - x\left(N-n\right) \right. \\ & \left. + x\left(n+\frac{N}{4}\right) - x\left(n+\frac{3N}{4}\right) \right] \Bigg\} \beta \left(\frac{2n+1}{2} 4k\right) \Bigg\rangle_{Mp} . \end{aligned} \tag{4.113}$$

The split-radix ONMNT DIF butterfly is shown in Figure 4.8.

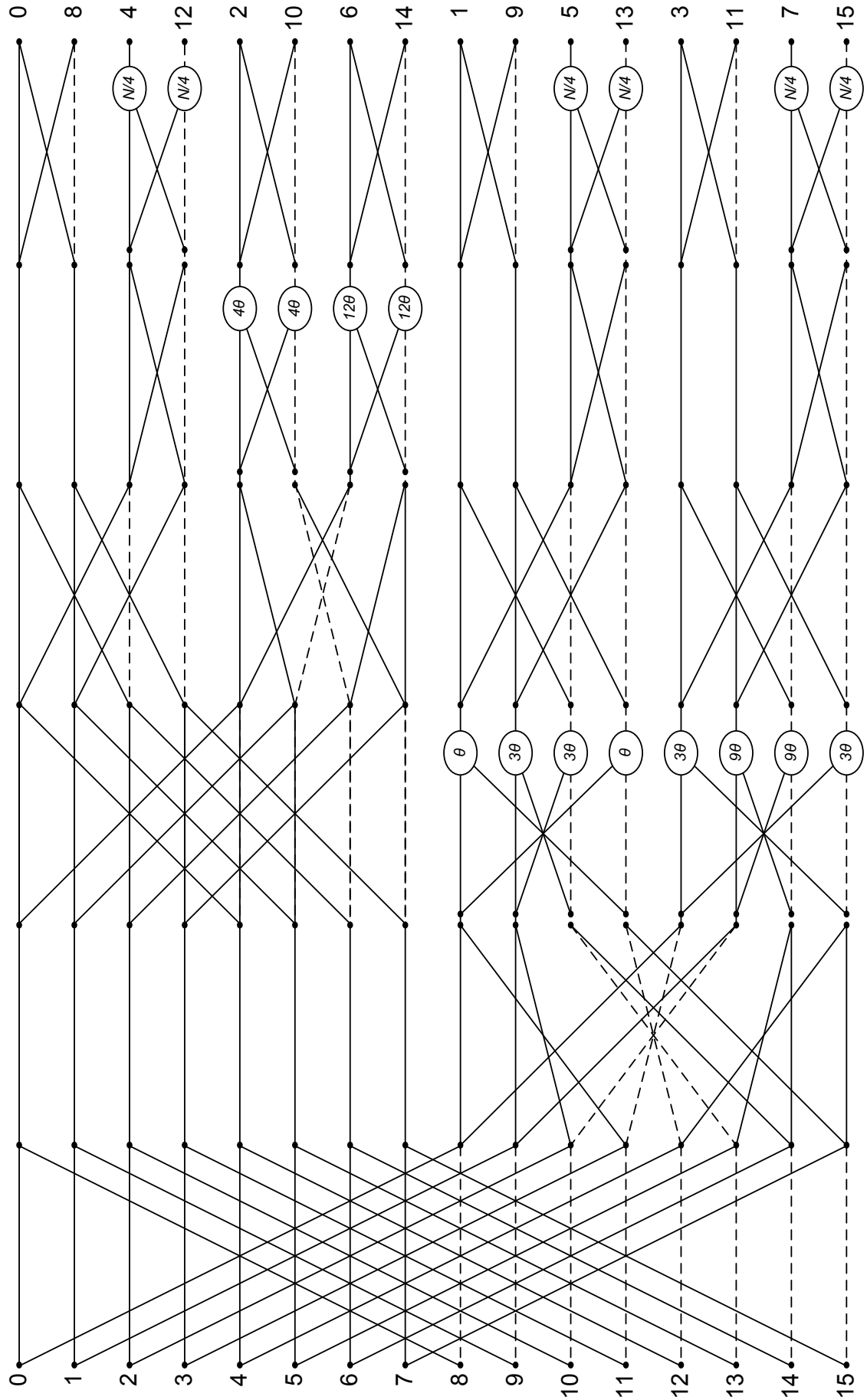


Figure 4.9: Split-Radix In-Place DIF Flow Diagram, $N = 16$

4.5 Complexity Analysis

Using the direct methods, similar to the discrete Fourier transform, the complexity is represented as N^2 multiplications and $N(N-1)$ additions. This is typical of multiplying a vector with a matrix. Applying the fast algorithm techniques that were proposed by [120] can have a significant impact in reducing this complexity. Using the fast algorithms significantly reduces this computation complexity as shown in Table 4.1. However, focusing on implementations where it is common for CPUs to take longer performing multiplication in comparison to additions, so the number of multiplications will be doubled to reflect this in respect of [122] as a typical example. Therefore, in order to integrate this adjustment and normalise the complexity, an adjust factor of $\lambda = 2$ will be applied to derive the complexities of multiplication and fused-multiply-add operations with respect to complexity. The complexity for the radix-2 ONMNT can be shown to be

$$M(N) = N\left(\log_2 N - 1\right)\lambda \tag{4.114}$$

and

$$A(N) = \frac{3}{2}N\log_2 N + \frac{N}{2}, \tag{4.115}$$

taking note that the first round of twiddle calculations can be reduced to trivial additions and this has therefore been reflected in the calculation of the number

Table 4.1: ONMNT Radix-2 Complexity

	Radix-2			Direct			Improvement %		
N	Mults	Adds	Total	Mults	Adds	Total	Mults	Adds	Total
4	8	14	22	32	12	44	75.00	-16.67	50.00
8	32	40	72	128	56	184	75.00	28.57	60.87
16	96	104	200	512	240	752	81.25	56.67	73.40
32	256	256	512	2048	992	3040	87.50	74.19	83.16
64	640	608	1248	8192	4032	12224	92.19	84.92	89.79
128	1536	1408	2944	32768	16256	49024	95.31	91.34	93.99
256	3584	3200	6784	131072	65280	196352	97.27	95.10	96.54
512	8192	7168	15360	524288	261632	785920	98.44	97.26	98.05
1024	18432	15872	34304	2097152	1047552	3144704	99.12	98.48	98.91
2048	40960	34816	75776	8388608	4192256	12580864	99.51	99.17	99.40
4096	90112	75776	165888	33554432	16773120	50327552	99.73	99.55	99.67
8192	196608	163840	360448	134217728	67100672	201318400	99.85	99.76	99.82
16384	425984	352256	778240	536870912	268419072	805289984	99.92	99.87	99.90
32768	917504	753664	1671168	2147483648	1073709056	3221192704	99.96	99.93	99.95
65536	1966080	1605632	3571712	8589934592	4294901760	12884836352	99.98	99.96	99.97
131072	4194304	3407872	7602176	34359738368	17179738112	51539476480	99.99	99.98	99.99
262144	8912896	7208960	16121856	137438953472	68719214592	206158168064	99.99	99.99	99.99

Table 4.2: ONMNT Radix-4 Complexity

	Radix-4			Radix-2			Improvement %		
N	Mults	Adds	Total	Mults	Adds	Total	Mults	Adds	Total
4	8	10	18	8	14	22	0.00	28.57	18.18
16	80	84	164	96	104	200	16.67	19.23	18.00
64	512	512	1024	640	608	1248	20.00	15.79	17.95
256	2816	2752	5568	3584	3200	6784	21.43	14.00	17.92
1024	14336	13824	28160	18432	15872	34304	22.22	12.90	17.91
4096	69632	66560	136192	90112	75776	165888	22.73	12.16	17.90
16384	327680	311296	638976	425984	352256	778240	23.08	11.63	17.89
65536	1507328	1425408	2932736	1966080	1605632	3571712	23.33	11.22	17.89
262144	6815744	6422528	13238272	8912896	7208960	16121856	23.53	10.91	17.89

of multiplications and additions respectively in (4.114) and (4.115). The radix-4 ONMNT improves upon this complexity by significantly reducing the number of operations through halving the number of stages. However, this requires a further sub-stage to be calculated for each stage, which results in N more additions taking place. Depending on how this sub-stage is handled, a further re-ordering may be required so as typically there will be $\frac{N}{4}$ signal lines that will be derived out of place, which can be clearly seen in Figure 4.3. The complexity of the radix-4 transform can be shown to be

$$M(N) = \left[\frac{N}{2}\left(3\log_4 N - 1\right)\right]\lambda \tag{4.116}$$

and

$$A(N) = \frac{N}{4}\left(11\log_4 N - 1\right), \tag{4.117}$$

Table 4.3: ONMNT Split-Radix Complexity

	Split-Radix			Radix-4			Improvement %			Radix-2			Improvement %		
N	Mults	Adds	Total	Mults	Adds	Total	Mults	Adds	Total	Mults	Adds	Total	Mults	Adds	Total
4	8	10	18	8	10	18	0.00	0.00	0.00	8	14	22	0.00	28.57	18.18
8	24	30	54							32	40	72	25.00	25.00	25.00
16	72	82	154	80	84	164	10.00	2.38	6.10	96	104	200	25.00	21.15	23.00
32	184	206	390							256	256	512	28.13	19.53	23.83
64	456	498	954	512	512	1024	10.94	2.73	6.84	640	608	1248	28.75	18.09	23.56
128	1080	1166	2246							1536	1408	2944	29.69	17.19	23.71
256	2504	2674	5178	2816	2752	5568	11.08	2.83	7.00	3584	3200	6784	30.13	16.44	23.67
512	5688	6030	11718							8192	7168	15360	30.57	15.88	23.71
1024	12744	13426	26170	14336	13824	28160	11.10	2.88	7.07	18432	15872	34304	30.86	15.41	23.71
2048	28216	29582	57798							40960	34816	75776	31.11	15.03	23.73
4096	61896	64626	126522	69632	66560	136192	11.11	2.91	7.10	90112	75776	165888	31.31	14.71	23.73
8192	134712	140174	274886							196608	163840	360448	31.48	14.44	23.74
16384	291272	302194	593466	327680	311296	638976	11.11	2.92	7.12	425984	352256	778240	31.62	14.21	23.74
32768	626232	648078	1274310							917504	753664	1671168	31.75	14.01	23.75
65536	1339848	1383538	2723386	1507328	1425408	2932736	11.11	2.94	7.14	1966080	1605632	3571712	31.85	13.83	23.75
131072	2854456	2941838	5796294							4194304	3407872	7602176	31.94	13.68	23.75
262144	6058440	6233202	12291642	6815744	6422528	13238272	11.11	2.95	7.15	8912896	7208960	16121856	32.03	13.54	23.76

Table 4.4: ONMNT Fused-Split-Radix Complexity

	Fused-Split-Radix				Split-Radix			Improvement %			
N	Mults	Adds	FMAs	Total	Mults	Adds	Total	Mults	Adds	FMAs	Total
4	4	8	4	16	8	10	18	50.00	20.00	-22.22	11.11
8	12	24	12	48	24	30	54	50.00	20.00	-22.22	11.11
16	36	64	36	136	72	82	154	50.00	21.95	-23.38	11.69
32	92	160	92	344	184	206	390	50.00	22.33	-23.59	11.79
64	228	384	228	840	456	498	954	50.00	22.89	-23.90	11.95
128	540	896	540	1976	1080	1166	2246	50.00	23.16	-24.04	12.02
256	1252	2048	1252	4552	2504	2674	5178	50.00	23.41	-24.18	12.09
512	2844	4608	2844	10296	5688	6030	11718	50.00	23.58	-24.27	12.14
1024	6372	10240	6372	22984	12744	13426	26170	50.00	23.73	-24.35	12.17
2048	14108	22528	14108	50744	28216	29582	57798	50.00	23.85	-24.41	12.20
4096	30948	49152	30948	111048	61896	64626	126522	50.00	23.94	-24.46	12.23
8192	67356	106496	67356	241208	134712	140174	274886	50.00	24.03	-24.50	12.25
16384	145636	229376	145636	520648	291272	302194	593466	50.00	24.10	-24.54	12.27
32768	313116	491520	313116	1117752	626232	648078	1274310	50.00	24.16	-24.57	12.29
65536	669924	1048576	669924	2388424	1339848	1383538	2723386	50.00	24.21	-24.60	12.30
131072	1427228	2228224	1427228	5082680	2854456	2941838	5796294	50.00	24.26	-24.62	12.31
262144	3029220	4718592	3029220	10777032	6058440	6233202	12291642	50.00	24.30	-24.64	12.32

representing multiplication and addition respectively, with representations of these complexities shown for different lengths of N in Table 4.2. Using the radix-4 significantly improves upon the radix-2. However, there is a penalty for this in that the radix-4 algorithm can only process lengths that are themselves a power of four. This may well mean that these bounds are too big to adequately and efficiently process a radix-4 length. Representing the final derivation of a fast ONMNT implementation, the complexity for the split-radix ONMNT can be shown to be

$$M(N) = \left[\frac{2}{3}N\log_2 N - \frac{4}{9}N + \frac{4}{9}(-1)^{log_2 N}\right]\lambda \tag{4.118}$$

and while the number of additions can be expressed as

$$A(N) = \frac{4}{3}N\log_2 N - \frac{2}{9}N + \frac{2}{9}(-1)^{log_2 N}. \tag{4.119}$$

At the cost of a little inconvenience in processing the split-radix, we gain both an improvement in the processing complexity and the ability to process all vector lengths that are available to radix-2 as shown in Table 4.3. Typically, a critical evaluation of complexity analysis in current architectures usually shows that there is an emphasis in contrasting algorithms between the multiplication and addition operations, rather than considering the total number of operations and how they can be grouped and implemented. With current architectures, a multiplication operation can be typically processed in the same time as it takes to perform an

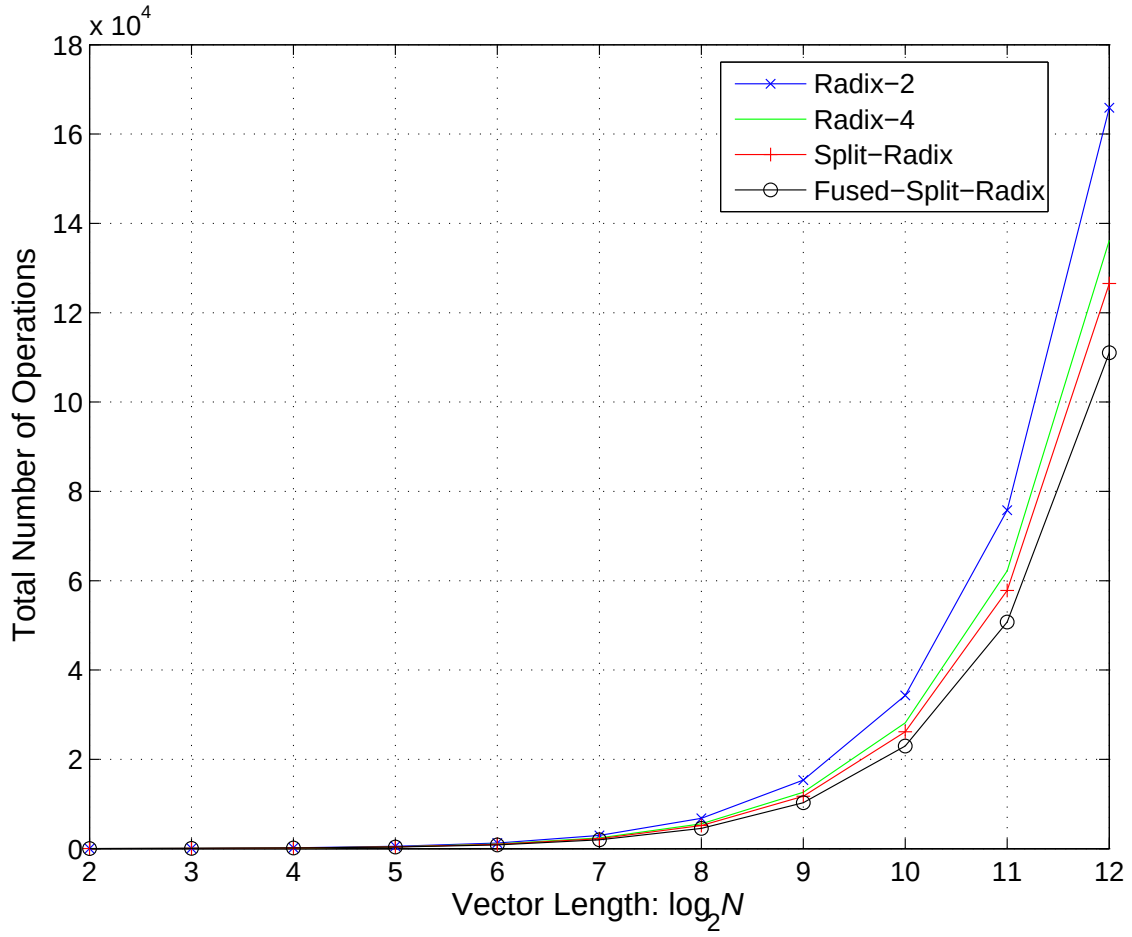


Figure 4.10: Complexity of Different Radices by Total Operations

addition operation [123]. Also, operators are now available that can combine three operands into a fused multiplication and accumulate (FMA), a process first introduced in [124]. Such operators have been implemented in many architectures including [123], [125] and [126] for example, which can be carried out at regular addition speeds within the constraints of the word size [123]. Using these types of architectures, it would therefore be possible to combine half of the multiplicative operations of the ONMNT with an equal number of addition operations using fused multiply and accumulate operators. This would significantly reduce the total number of operations that are required by half of the number of multiplications. By implementing on appropriate architecture, the number of operations would be reflected to those shown in Table 4.4. A modified complexity calculation can therefore be shown to be

$$F(N) = \left[\frac{1}{3}N\log_2 N - \frac{2}{9}N + \frac{2}{9}(-1)^{log_2 N}\right]\lambda, \tag{4.120}$$

$$M(N) = \left[\frac{1}{3}N\log_2 N - \frac{2}{9}N + \frac{2}{9}(-1)^{log_2 N}\right]\lambda \tag{4.121}$$

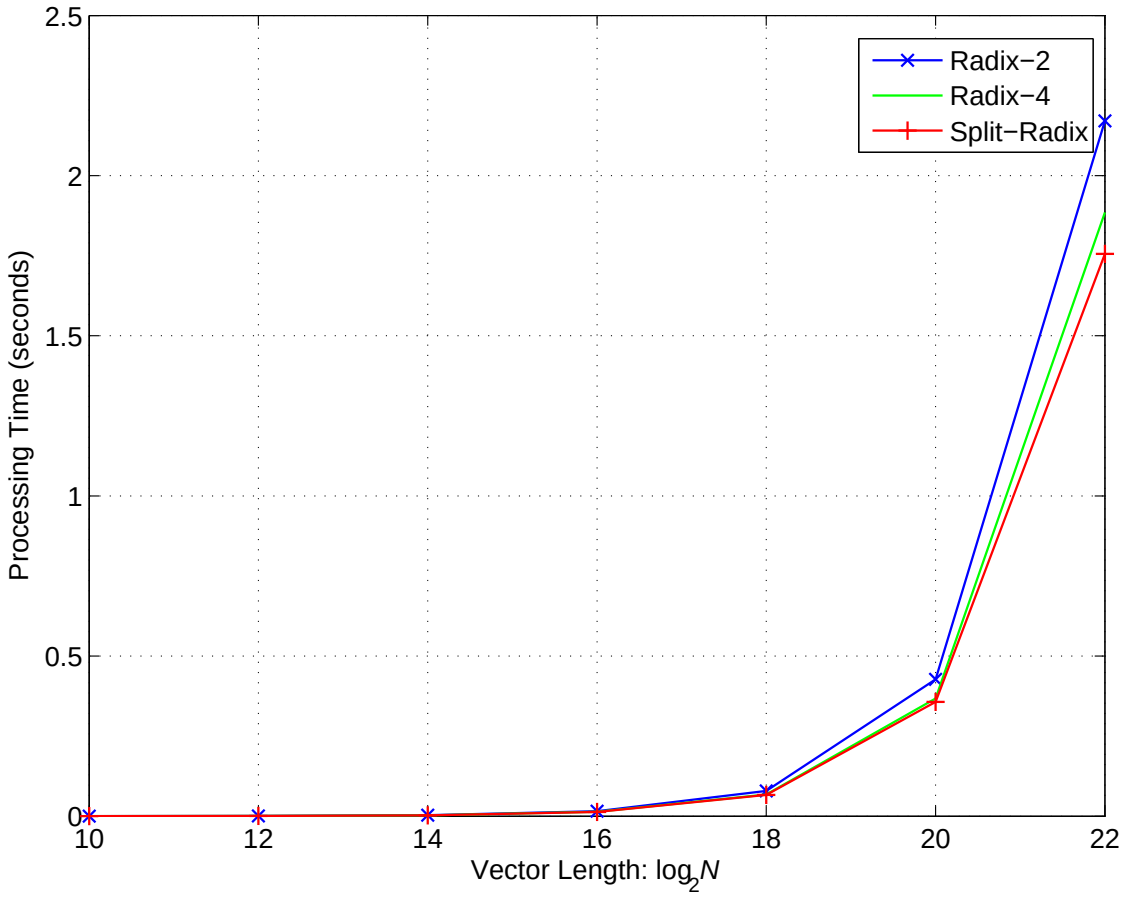


Figure 4.11: Processing Time for Vectors of Length 2^N

and

$$A(N) = N \log_2 N, \tag{4.122}$$

representing the FMA operations and discrete multiply and addition operations respectively. The total number of operations is calculated by

$$T(N) = \frac{2\lambda + 3}{3} N \log_2 N - \frac{4\lambda}{9} N + \frac{4\lambda}{9} (-1)^{log_2 N}. \tag{4.123}$$

The improvement in reducing processing complexity that this process can provide is shown in Figure 4.10. This is further demonstrated by observing and example of a typical implementation in Listing A.6.

Listing 4.1: Typical Implementation

```
unsigned int t00, t11, t10, t01, in0, in1, out0, out1;
unsigned int B1[N], B2[N];
t00 = in0 * B1[theta];
t11 = in1 * B1[theta];
t10 = in1 * B2[theta];
```

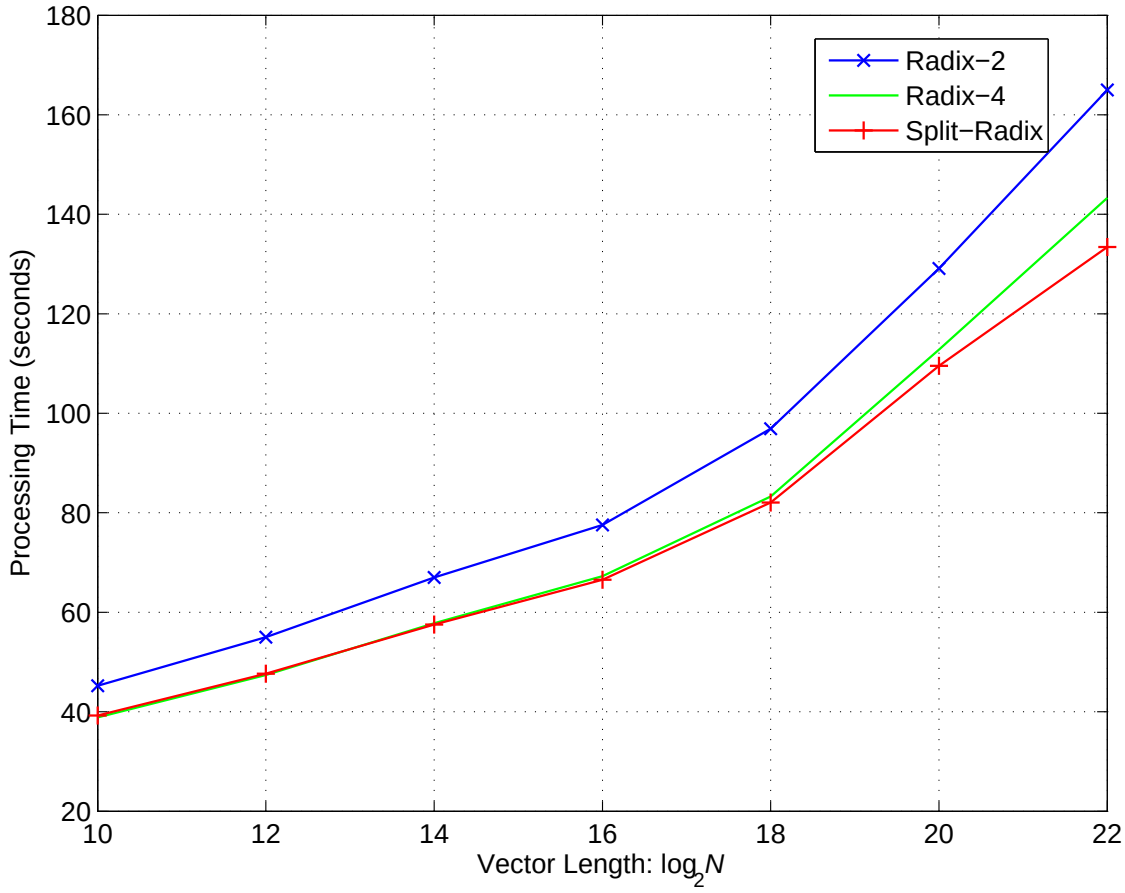


Figure 4.12: Time to Process $\approx 10^{10}$ bits Using Vectors of Length 2^N

```
t01 = in0 * B2[theta];
out0 = t00 + t10;
out1 = t11 - t01;
```

By removing half of the multiplications and the equivalent number of additions to implement the FMA commands, we can see that not only has the complexity been reduced but also the number of operations as shown in Listing 4.2.

Listing 4.2: Fused-Multiply-Accumulate Implementation

```
unsigned int t0, t1, in0, in1, out0, out1;
unsigned int B1[N], iB[N], B2[N];
t0 = in0 * B1[theta];
t1 = in1 * iB1[theta];
out0 = fma(in1, B2[theta], t0);
out1 = fma(in0, B2[theta], t1);
```

In this updated variation, there remains the requirement to provide the requisite subtraction operation. However, this minor issue can be overcome by implementing

twos-complement negation equivalent table for β_1 (shown in 4.2 as `iB1[theta]`) and would reduce the processing time from six-cycles to four-cycles per butterfly for a 33.3% processing improvement, at a cost of introducing an additional table of size $\log_2 N$. At worst, performing the extra subtraction to the variable `t1` in order to save memory may be preferable, in which case this processing time is only reduced to five-cycles, which would improve the processing time by only 16.7%.

4.5.1 Higher Radices of the GNMNT

While there are clearly tangible benefits in terms of reduced complexity to be obtained in developing the radix-4 and split-radix-2/4, this comes at a cost of implementation complexity, which becomes progressively more difficult with successive increases of the radices. Moreover, increasing the radix also restricts the flexibility of the transform as it constrains the available lengths of the transform to multiples of $N = r^s$ where r is the selected radix and s is the exponent used to configure the length, which is shown more clearly in Table 4.5. To incorporate higher radices with lengths that aren't directly supported by these radices as shown in Table 4.5 would require that the implementation consist of a split-radix scheme. However, this would impede the implementation with respect to structural complexity, which can also have a negative impact on resource constrained devices

Table 4.5: Available Lengths $N = r^s$ According to Radix

		r			
		2	4	8	16
s	2	4	-	-	-
	3	8	-	-	-
	4	16	16	-	-
	5	32	-	-	-
	6	64	64	64	-
	7	128	-	-	-
	8	256	256	-	256
	9	512	-	512	-
	10	1024	1024	-	-
	11	2048	-	-	-
	12	4096	4096	4096	4096

where program and memory sizes are primary concerns. What is unintuitive with Table 4.5 is that some lengths are unavailable owing to the butterfly for the particular radix consisting of $2r$ points, therefore suggesting that the minimum implementation size would be r^2 and not r.

4.6 Performance Analysis of the One-Dimensional Derivations

A significant factor governing implementation should be taken into consideration by using languages such as OpenCL [127] and other architectures incorporating features such as advanced vector extensions (AVX) [126], where it is possible to combine similar operations across multiple signal lines within vector variables simultaneously. These types of techniques are encompassed by single-instruction multiple-data (SIMD) described in [128], which debuted in Intels processer with multimedia extensions (MMX) [129]. An example how this would benefit a butterfly implementation using OpenCL is shown in Listing 4.3.

Listing 4.3: Vectorised Fused-Multiply-Accumulate Implementation

```
unsigned int2 t0, in, out;
unsigned int B1[N], iB[N], B2[N];
t = (unsigned int2)(in.0 * B1[theta], in.1 * iB1[theta]);
out = fma(in.10, (unsigned int2)(B2[theta]), t);
```

This new method would further reduce the processing time to only two cycles when using the additional negated β_1 table or three-cycles if incorporating the extra subtraction, resulting in a overall performance improvements of 66.7% and 50% respectively. Such methodologies imply natural parallelism where a small number of operations can be unrolled out of loops into vectors. However, there are limits imposed upon such practices with respect to the maximum allowable size of the vector and the optimum size before the hardware incurs penalties. The latter imposition is a reflection to how the architecture is addressing memory. For example, the AMD 7970 is able to manipulate up to 16 elements and can access 128 bits of memory using a single variable without incurring any performance penalties. These constraints reflect configurations of 16 $\times$ 8-bit chars, 8 $\times$ 16-bit shorts, 4 $\times$ 32-bit integers / single precision floats or 2 $\times$ 64-bit long integers

/ double precision floats without any performance penalties. Further constraints would be imposed if one or more data words are accessed that cross a 128-bit boundary, which would likely to be similar to an unaligned memory access. Aside from these constraints, it would be possible to process, depending on the word size being used, a number of operations simultaneously in the time it would normally take to process a single operation, thereby increasing the processing performance even further. This would obviously require additional implementation planning as where a loop would be designed to process N lines within a stage, the loop could potentially be reduced by $\frac{N}{2}$, $\frac{N}{4}$, $\frac{N}{8}$ or even $\frac{N}{16}$, depending on the capabilities and versatilities of the architecture and implementation respectively. The next step in enhancing the performance is naturally to employ parallel processing techniques on parallel capable devices. With respect to the current development of CPUs, there is already scope at least to incorporate multiple processing cores. Going further would utilise available streams within a CPU that are dedicated to the onboard GPU. Increasing performance even more would either suggest using more CPUs, which can be costly and require specialised hardware to incorporate multiple CPUs on the same board, or to use other options such as discrete GPUs or CPU cards. Using additional discrete hardware can by far be the most versatile solution and sometimes the cheapest. While a CPU card, such as the many integrated core (MIC) architecture offered by Intel [130] offer the greatest versatility, but the highest cost. A cheaper alternative would be to use a GPU that is applicable for GPGPU computing.

There are many benefits with following the GPGPU route in that there are usually a significantly greater number of cores available with GPGPU computing typically than MIC solutions. Respectively, this currently represents a ratio of 36:1 with respect to available stream processors / cores. Additionally, the GPUs are significantly cheaper than MICs. However, GPGPUs are more specialised and require a significant amount of application development to fully realise their potential, whereas MICs, being more general purpose, are typically more adaptable.

Of course, should one go fully down the parallel route, it may turn out that the fastest method to implement by far would be the direct method, as depending on the size of N and the constraints of the architecture, so all of the multiplications would occur simultaneously. Subsequently, there would be $\log_2 N$ reduction stages that would add all of the appropriate results. This would suggest a significant

improvement over a sequence involving $2\log_2 N - 1$ radix-2 or $3\log_2 N$ radix-4 / split-radix total sub-stages. In addition to this, there would be a delay of at least two successive operations in each twiddle sub-stage; one to perform the initial multiplication of either β_1 or β_2, followed by combining the result with a fused multiply accumulate of the composite signal path. However, in order to perform a vector by matrix multiplication simultaneously in parallel, the involvement of operating with twiddles would be completely removed.

4.7 Conclusion

This chapter has introduced complete developments of the radix-2, radix-4 and split-radix algorithms for the ONMNT, in both DIT and DIF. The importance in developing fast and efficient implementations is to ensure that economical, fast and versatile methods exist such that applications for real-time systems are tenable. These implementations have been assessed according to the number of mathematical operations that they require and tested to confirm the successive improvements in execution time. Additionally, methods using new architectures have been included to signify the impact that the architectures have in the future development of new algorithms. While no direct comparisons in terms of speed by incorporating these new architectures have been made, they have been implemented in a parallel processing capacity, which was used to provide the exhaustive assessments in Chapter 6.

Chapter 5

The Row-Column GNMNT

5.1 Introduction

In the previous chapters, the 1D GNMNT has been presented and shown that it can be calculated either as a fast algorithm or by the direct method. However, there are also a number of areas where 2D processing is relevant, particularly in image- and signal-processing applications where there are many areas that thrive on such techniques and also on cryptography [104, 131–134]. As such, there are many algorithms to facilitate the wealth of applications including the 2D-Fourier Transform, 2D-Wavelet Transform and the 2D-Hartley Transform, to name but few [135–137].

This chapter introduces the 2D-GNMNT using the row-column (RC) method, including the extra calculations that are required in order to use the RC technique, which is a non-separable algorithm used to attain a true 2D-algorithm. The RC method of processing an $N \times N$ area can be seen as one of the most convenient methods of using a single-dimensional algorithm to provide the functionality of a two-dimensional algorithm. The RC achieves this by first transforming the desired matrix row-by-row and then transforming the resultant rows in a column-by-column fashion. However, the RC algorithm on its own will not provide the desired results when using the GNMNT because the β term is not separable when used in this manner and therefore the RC method must be further processed. The initial two-dimensional algorithm is shown for each of the GNMNTs, beginning with the NMNT for an introductory point of reference, where steps to derive the forward and inverse 2D transforms; also included are the steps required to enable the transforms

to be separable are provided.

5.2 RC-NMNT

The standard algorithm for the 2D-NMNT is

$$X(k_1,k_2)=\left\langle \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x(n_1,n_2)\beta\left(n_1k_1,n_2k_2\right)\right\rangle_{Mp} \quad \text{for } k_1,k_2=0,1,2,...,N-1. \tag{5.1}$$

However, as the β term in the 2D-NMNT is non-separable due to its characteristic, then it can be shown that processing the NMNT using the RC method actually yields the following algorithm

$$X_{RC}(k_1,k_2)=\left\langle \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x(n_1,n_2)\beta\left(n_1k_1\right)\beta\left(n_2k_2\right)\right\rangle_{Mp} \quad \text{for } k_1,k_2=0,1,2,...,N-1. \tag{5.2}$$

This is obviously a different algorithm because it is clear that

$$\beta\left(n_1k_1,n_2k_2\right)\neq\beta\left(n_1k_1\right)\beta\left(n_2k_2\right). \tag{5.3}$$

Therefore, in order to achieve the separable version of the NMNT an additional step is required, which was first described in [138]. This step is achieved by breaking down the $\beta\left(n_1k_1,n_2k_2\right)$ term to

$$\beta\left(n_1k_1,n_2k_2\right)=\left\langle\beta_1\left(n_1k_1,n_2k_2\right)+\beta_2\left(n_1k_1,n_2k_2\right)\right\rangle_{Mp}. \tag{5.4}$$

where we can further derive β_1 and β_2 by their definitions as

$$\beta_1\left(n_1k_1,n_2k_2\right)=\left\langle \mathrm{Re}\left(\alpha_1+j\alpha_2\right)^{n_1k_1+n_2k_2}\right\rangle_{Mp} \tag{5.5}$$

and

$$\beta_2\left(n_1k_1,n_2k_2\right)=\left\langle \mathrm{Im}\left(\alpha_1+j\alpha_2\right)^{n_1k_1+n_2k_2}\right\rangle_{Mp}. \tag{5.6}$$

Equations (5.5) and (5.6) can be further processed by expanding the exponential term as

$$\beta_1\left(n_1k_1,n_2k_2\right)=\left\langle \mathrm{Re}\left[\left(\alpha_1+j\alpha_2\right)^{n_1k_1}\left(\alpha_1+j\alpha_2\right)^{n_2k_2}\right]\right\rangle_{Mp} \tag{5.7}$$

and

$$\beta_2\left(n_1k_1,n_2k_2\right)=\left\langle \mathrm{Im}\left[\left(\alpha_1+j\alpha_2\right)^{n_1k_1}\left(\alpha_1+j\alpha_2\right)^{n_2k_2}\right]\right\rangle_{Mp}. \tag{5.8}$$

Simplifying the β_1 term in (5.7) produces

$$\begin{aligned}\beta_1(n_1k_1, n_2k_2) &= \mathrm{Re}\Big\langle \Big[\mathrm{Re}(\alpha_1 + j\alpha_2)^{n_1k_1} + j\mathrm{Im}(\alpha_1 + j\alpha_2)^{n_1k_1}\Big] \\ &\qquad \times \Big[\mathrm{Re}(\alpha_1 + j\alpha_2)^{n_2k_2} + j\mathrm{Im}(\alpha_1 + j\alpha_2)^{n_2k_2}\Big] \Big\rangle_{Mp} \\ &= \mathrm{Re}\Big\langle \beta_1(n_1k_1)\beta_1(n_2k_2) + j\beta_1(n_1k_1)\beta_2(n_2k_2) \\ &\qquad + j\beta_1(n_2k_2)\beta_2(n_1k_1) - \beta_2(n_1k_1)\beta_2(n_2k_2) \Big\rangle_{Mp} \\ &= \beta_1(n_1k_1 + n_2k_2)\end{aligned} \tag{5.9}$$

and similarly

$$\beta_2(n_1k_1, n_2k_2) = \beta_2(n_1k_1 + n_2k_2). \tag{5.10}$$

Substituting (5.9) and (5.10) into (5.4) produces

$$\begin{aligned}\beta(n_1k_1, n_2k_2) &= \Big\langle \beta_1(n_1k_1 + n_2k_2) + \beta_2(n_1k_1 + n_2k_2) \Big\rangle_{Mp} \\ &= \beta(n_1k_1 + n_2k_2) \\ &= \Big\langle \beta_1(n_1k_1)\beta(n_2k_2) + \beta_2(n_1k_1)\beta(-n_2k_2) \Big\rangle_{Mp}\end{aligned} \tag{5.11}$$

because

$$\beta_1(n_1k_1) = \Big\langle \beta(n_1k_1) - \beta_2(n_1k_1) \Big\rangle_{Mp} \tag{5.12}$$

and

$$\begin{aligned}\beta_1(-n_1k_1) &= \beta_1(n_1k_1) \\ &= \Big\langle \beta(-n_1k_1) + \beta_2(n_1k_1) \Big\rangle_{Mp}.\end{aligned} \tag{5.13}$$

Combining (5.12) and (5.13) produces

$$2\beta_1(n_1k_1) = \Big\langle \beta(n_1k_1) + \beta(-n_2k_2) \Big\rangle_{Mp} \tag{5.14}$$

thus

$$\beta_1(n_1k_1) = \left\langle \frac{1}{2}\Big[\beta(n_1k_1) + \beta(-n_2k_2)\Big] \right\rangle_{Mp} \tag{5.15}$$

and

$$\beta_2(n_1k_1) = \left\langle \frac{1}{2}\Big[\beta(n_1k_1) - \beta(-n_2k_2)\Big] \right\rangle_{Mp}. \tag{5.16}$$

Substituting (5.15) and (5.16) into (5.11) produces

$$\begin{aligned}\beta(n_1k_1, n_2k_2) &= \Big\langle \beta_1(n_1k_1)\,\beta(n_2k_2) + \beta_2(n_1k_1)\,\beta(-n_2k_2) \Big\rangle_{Mp} \\ &= \Bigg\langle \frac{1}{2}\Big[\beta(n_1k_1) + \beta(-n_1k_1)\Big]\beta(n_2k_2) \\ &\qquad + \frac{1}{2}\Big[\beta(n_1k_1) - \beta(-n_1k_1)\Big]\beta(-n_2k_2) \Bigg\rangle_{Mp} \\ &= \Big\langle \Big[\beta(n_1k_1)\,\beta(n_2k_2) + \beta(-n_1k_1)\,\beta(n_2k_2) \\ &\qquad + \beta(n_1k_1)\,\beta(-n_2k_2) - \beta(-n_1k_1)\,\beta(-n_2k_2)\Big]2^{p-1} \Big\rangle_{Mp}.\end{aligned} \tag{5.17}$$

Applying $X_{RC}(k_1, k_2)$ to (5.1) produces the full separable 2D transform for the NMNT and can therefore be written as

$$\begin{aligned}X(k_1, k_2) = \Bigg\langle \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x(n_1, n_2) \Bigg\{ \Big[\beta(n_1k_1)\,\beta(n_2k_2) + \beta(-n_1k_1)\,\beta(n_2k_2) \\ + \beta(n_1k_1)\,\beta(-n_2k_2) - \beta(-n_1k_1)\,\beta(-n_2k_2)\Big]2^{p-1} \Bigg\} \Bigg\rangle_{Mp}\end{aligned} \tag{5.18}$$

and further simplified as

$$\begin{aligned}X(k_1, k_2) = \Big\langle \Big[X_{RC}(k_1, k_2) + X_{RC}(-k_1, k_2) \\ + X_{RC}(k_1, -k_2) - X_{RC}(-k_1, -k_2)\Big]2^{p-1} \Big\rangle_{Mp}.\end{aligned} \tag{5.19}$$

A demonstration of how this is presented is shown in Figure 5.1, which started by reducing the cameraman image to an 8×8 array and using the RC technique to transform the data to a 2D-NMNT, using $N = 8$, $p = 13$ and $Mp = 8191$. The data is shown as it progresses through the various stages of the process, both forwards and backwards.

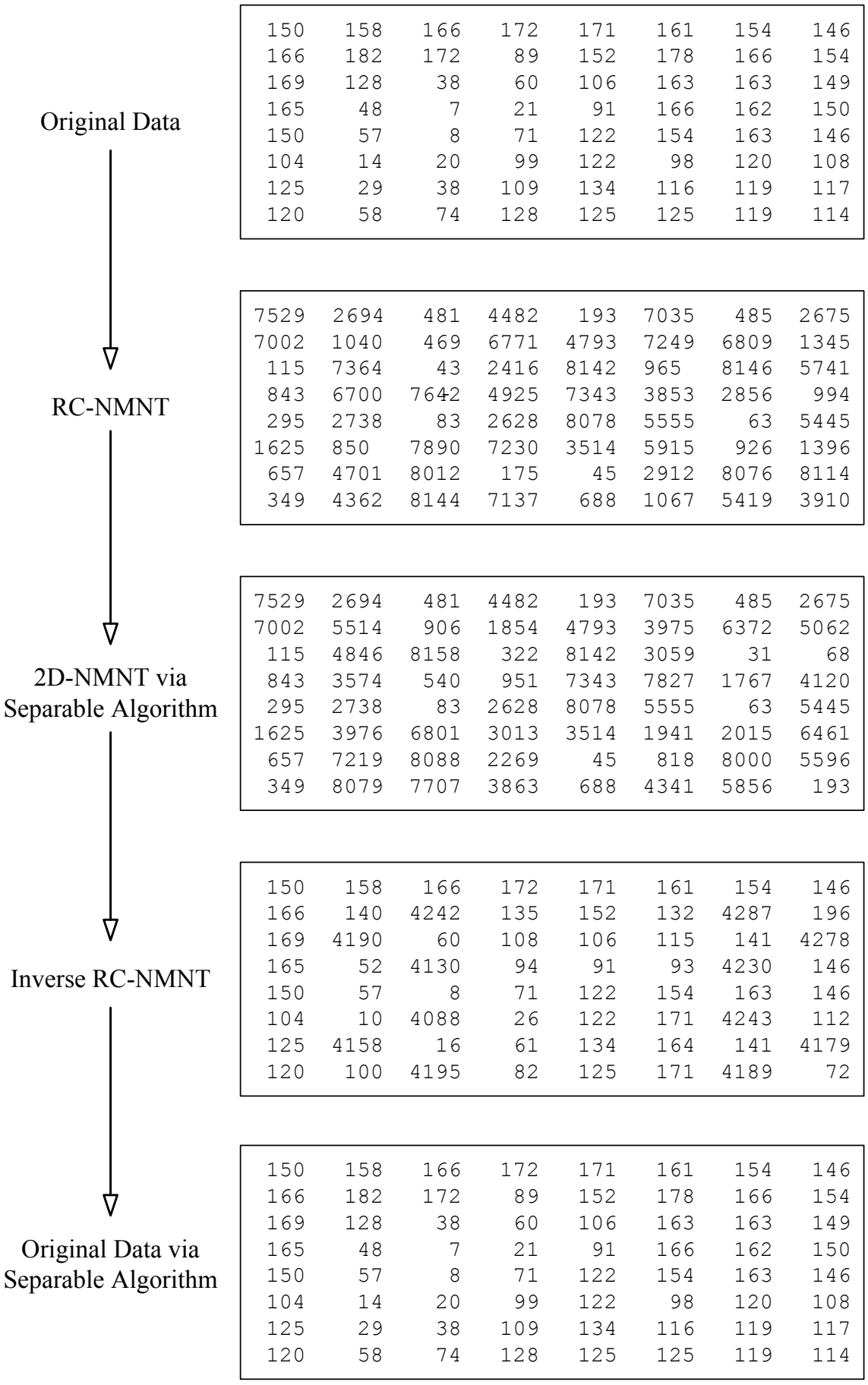


Figure 5.1: Example of 2D-NMNT using Row-Column Method

5.3 RC-ONMNT

The 2D-ONMNT is similar to the 2D-NMNT; however to obtain the full 2D-ONMNT using the RC method, there are two distinct procedures that must be derived for the forward and inverse transforms. The forward transform uses $\beta\left(\frac{2k_1+1}{2}n_1, \frac{2k_2+1}{2}n_2\right)$, shown as

$$X(k_1,k_2) = \left\langle \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x(n_1,n_2)\beta\left(\frac{2k_1+1}{2}n_1, \frac{2k_2+1}{2}n_2\right)\right\rangle_{Mp} \quad \text{for } k_1,k_2 = 0,1,2,...,N-1. \tag{5.20}$$

As the negative instances of k were replaced with $N-k$ in the NMNT, they must now be replaced by $N-k-1$ for the ONMNT otherwise negating k would otherwise produce

$$\begin{aligned}\beta\left[\frac{2(N-k)+1}{2}n\right] &= \beta\left(\frac{2N-2k+1}{2}n\right)\\ &= \beta\left(\frac{2Nn}{2}+\frac{-2k+1}{2}n\right)\\ &= \beta\left(\frac{-2k+1}{2}n\right)\\ &= \beta\left(-\frac{2k-1}{2}n\right).\end{aligned} \tag{5.21}$$

This detail is necessary when using negative k in the fractional part of the index so that correct index is maintained by

$$\begin{aligned}\beta\left[\frac{2(N-k-1)+1}{2}n\right] &= \beta\left(\frac{2N-2k-2+1}{2}n\right)\\ &= \beta\left(\frac{2Nn}{2}+\frac{-2k-1}{2}n\right)\\ &= \beta\left(\frac{-2k-1}{2}n\right)\\ &= \beta\left(-\frac{2k+1}{2}n\right).\end{aligned} \tag{5.22}$$

The RC definition for the 2D-ONMNT is

$$X_{RC}(k_1,k_2) = \left\langle \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x(n_1,n_2)\beta\left(\frac{2k_1+1}{2}n_1\right)\beta\left(\frac{2k_2+1}{2}n_2\right)\right\rangle_{Mp} \quad \text{for } k_1,k_2 = 0,1,2,...,N-1 \tag{5.23}$$

and the full separable equation that was shown in (5.18) now becomes

$$\begin{aligned} X\left(k_1,k_2\right)=\Bigg\langle \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x\left(n_1,n_2\right)\Bigg[&\beta\left(\frac{2k_1+1}{2}n_1\right)\beta\left(\frac{2k_2+1}{2}n_2\right)\\ &+\beta\left(-\frac{2k_1+1}{2}n_1\right)\beta\left(\frac{2k_2+1}{2}n_2\right)\\ &+\beta\left(\frac{2k_1+1}{2}n_1\right)\beta\left(-\frac{2k_2+1}{2}n_2\right)\\ &-\beta\left(-\frac{2k_1+1}{2}n_1\right)\beta\left(-\frac{2k_2+1}{2}n_2\right)\Bigg]2^{p-1}\Bigg\rangle_{Mp}. \end{aligned} \tag{5.24}$$

In order to obtain $\beta\left(-\frac{2k_1+1}{2}n_1\right)$ from $\beta\left(\frac{2k_1+1}{2}n_1\right)$, changing the indices that contain $-k_1$ to $-k_1-1$ is necessary, this is shown as

$$\begin{aligned} \beta\left(\frac{2k_1+1}{2}n_1\right) &\xrightarrow{k_1=N-k_1-1} \beta\left[\frac{2\left(N-k_1-1\right)+1}{2}n_1\right]\\ &= \beta\left(\frac{2N-2k_1-1}{2}n_1\right)\\ &= \beta\left(-\frac{2k_1+1}{2}n_1\right) \end{aligned} \tag{5.25}$$

and can be represented by using X_{RC} so that

$$\begin{aligned} X\left(k_1,k_2\right)=\Big\langle\Big[&X_{RC}\left(k_1,k_2\right)+X_{RC}\left(-k_1-1,k_2\right)\\ &+X_{RC}\left(k_1,-k_2-1\right)-X_{RC}\left(-k_1-1,-k_2-1\right)\Big]2^{p-1}\Big\rangle_{Mp}. \end{aligned} \tag{5.26}$$

The IONMNT begins by first applying $\beta\left(\frac{2n_1+1}{2}k_1,\frac{2n_2+1}{2}k_2\right)$ to the original 2D-ONMNT formula and using $\overline{X}$ to indentify that the IONMNT transform is currently selected. The standard equation for the IONMNT then becomes

$$\overline{X}(k_1,k_2)=\left\langle N^{-2}\sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x(n_1,n_2)\beta\left(\frac{2n_1+1}{2}k_1,\frac{2n_2+1}{2}k_2\right)\right\rangle_{Mp} \tag{5.27}$$
$$\text{for } k_1,k_2=0,1,2,...,N-1$$

where the RC method is defined as

$$\overline{X}_{RC}(k_1,k_2)=\left\langle N^{-2}\sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x(n_1,n_2)\beta\left(\frac{2n_1+1}{2}k_1\right)\beta\left(\frac{2n_2+1}{2}k_2\right)\right\rangle_{Mp} \tag{5.28}$$
$$\text{for } k_1,k_2=0,1,2,...,N-1.$$

As previously shown from (5.17), it is obvious that $\beta\left(\frac{2n_1+1}{2}k_1, \frac{2n_2+1}{2}k_2\right)$ is applied to (5.11) so that

$$\begin{aligned}
\beta\left(\frac{2n_1+1}{2}k_1, \frac{2n_2+1}{2}k_2\right) &= \beta\left(\frac{2n_1+1}{2}k_1 + \frac{2n_2+1}{2}k_2\right) \\
&= \Bigg\langle \frac{1}{2}\left[\beta\left(\frac{2n_1+1}{2}k_1\right) + \beta\left(-\frac{2n_1+1}{2}k_1\right)\right] \\
&\quad \times \beta\left(\frac{2n_2+1}{2}k_2\right) \\
&\quad + \frac{1}{2}\left[\beta\left(\frac{2n_1+1}{2}k_1\right) - \beta\left(-\frac{2n_1+1}{2}k_1\right)\right] \\
&\quad \times \beta\left(-\frac{2n_2+1}{2}k_2\right)\Bigg\rangle_{Mp} \\
&= \Bigg\langle \Bigg[\beta\left(\frac{2n_1+1}{2}k_1\right)\beta\left(\frac{2n_2+1}{2}k_2\right) \\
&\quad + \beta\left(-\frac{2n_1+1}{2}k_1\right)\beta\left(\frac{2n_2+1}{2}k_2\right) \\
&\quad + \beta\left(\frac{2n_1+1}{2}k_1\right)\beta\left(-\frac{2n_2+1}{2}k_2\right) \\
&\quad - \beta\left(-\frac{2n_1+1}{2}k_1\right)\beta\left(-\frac{2n_2+1}{2}k_2\right)\Bigg]2^{p-1}\Bigg\rangle_{Mp}.
\end{aligned} \tag{5.29}$$

Substituting (5.29) into (5.27) produces

$$\begin{aligned}
\overline{X}(k_1,k_2) = \Bigg\langle N^{-2}\sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x(n_1,n_2)\Bigg[&\beta\left(\frac{2n_1+1}{2}k_1\right)\beta\left(\frac{2n_2+1}{2}k_2\right) \\
&+ \beta\left(-\frac{2n_1+1}{2}k_1\right)\beta\left(\frac{2n_2+1}{2}k_2\right) \\
&+ \beta\left(\frac{2n_1+1}{2}k_1\right)\beta\left(-\frac{2n_2+1}{2}k_2\right) \\
&- \beta\left(-\frac{2n_1+1}{2}k_1\right)\beta\left(-\frac{2n_2+1}{2}k_2\right)\Bigg]2^{p-1}\Bigg\rangle_{Mp}
\end{aligned} \tag{5.30}$$

However, as k_1 and k_2 are not part of the fractional part then (5.30) can further be simplified to

$$\begin{aligned}
\overline{X}\left(k_1,k_2\right) = \Big\langle \Big[&\overline{X}_{RC}\left(k_1,k_2\right) + \overline{X}_{RC}\left(-k_1,k_2\right) \\
&+ \overline{X}_{RC}\left(k_1,-k_2\right) - \overline{X}_{RC}\left(-k_1,-k_2\right)\Big]2^{p-1}\Big\rangle_{Mp}.
\end{aligned} \tag{5.31}$$

However, as the ONMNT kernel is not a symmetrical transform, it must

therefore rely upon the transpose of itself to obtain the inverse. Pre-processing is required for the 2D-ONMNT transform using the RC method.

According to the identity in 2D-ONMNT, the basic formula for forward transform is

$$X(k_1,k_2)=\left\langle \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x(n_1,n_2)\beta\left(\frac{2k_1+1}{2}n_1,\frac{2k_2+1}{2}n_2\right)\right\rangle_{Mp} \quad \text{for } k_1,k_2=0,1,2,...,N-1 \tag{5.32}$$

(k_1,k_2) can be deemed as different positions throughout a matrix, thus, when $k_1=0$, which represents the first column, then (5.32) can be written as

$$X(0,k_2)=\left\langle \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x(n_1,n_2)\beta\left(\frac{2k_2+1}{2}n_2\right)\right\rangle_{Mp} \quad \text{for } k_2=0,1,2,...,N-1. \tag{5.33}$$

Similarly, when $k_2=0$, which represents the first row, then (5.32) can be written as

$$X(k_1,0)=\left\langle \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x(n_1,n_2)\beta\left(\frac{2k_1+1}{2}n_1\right)\right\rangle_{Mp} \quad \text{for } k_1=0,1,2,...,N-1. \tag{5.34}$$

As shown in (5.32) and (5.34), both $X(0,k_2)$ and $X(k_1,0)$ have identical values by changing k_1 and k_2. However at position $(1,1)$ and $(N-1,N-1)$ produce transposed numbers in terms of the kernel matrix shown as

$$X(1,1)=\left\langle \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x(n_1,n_2)\beta\left(\frac{3}{2}n_1,\frac{3}{2}n_2\right)\right\rangle_{Mp} \tag{5.35}$$

When calculating position $(N-1,N-1)$, the actual position in the ONMNT is $(N-1-1,N-1-1)$ due to its identity, so that

$$X(N-1,N-1)=\left\langle \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x(n_1,n_2)\beta\left(-\frac{3}{2}n_1,-\frac{3}{2}n_2\right)\right\rangle_{Mp} \tag{5.36}$$

Comparing (5.35) and (5.36), the value in β terms at the position $X(1,1)$ and

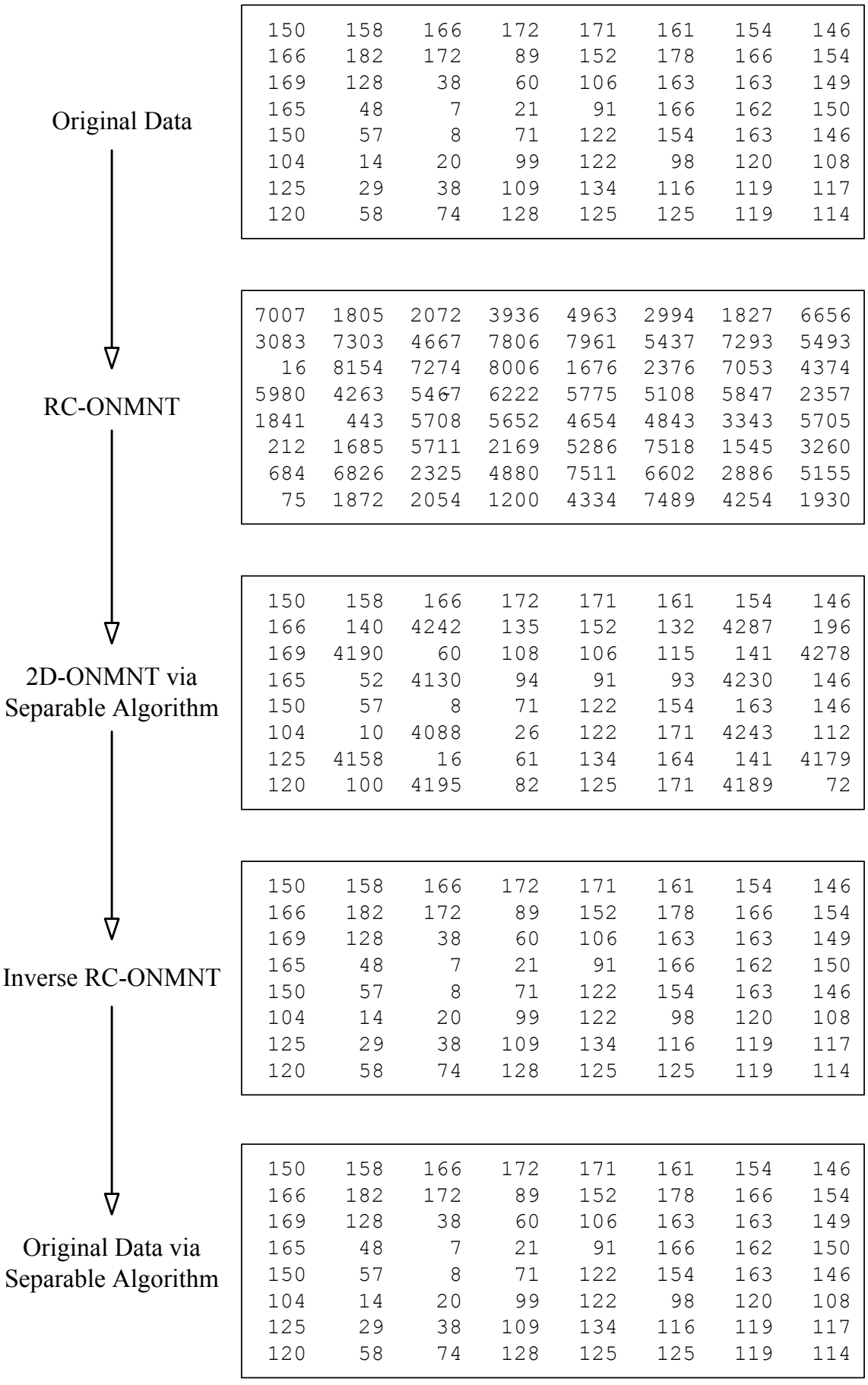


Figure 5.2: Example of 2D-ONMNT using Row-Column Method

position $X(N-1, N-1)$ is negated. Moreover, these positions can be represented as the positions $X(k_1, k_2)$ and $X(N-k_1, N-k_2)$, when k_1 or k_2 are not zero values. Thus, the pre-process step is to flip over and negate the input matrix, apart from first row and first column. A demonstration of how this is presented is shown in Figure 5.2, which started by reducing the cameraman image to an 8×8 array and using the RC technique to transform the data to a 2D-ONMNT, using $N = 8$, $p = 13$ and $Mp = 8191$. The data is shown as it progresses through the various stages of the process, both forwards and backwards.

5.4 RC-O^{2}NMNT

As discussed in the previous chapter, the GNMNT consists of the NMNT, ONMNT and the O^{2}NMNT. Therefore, the 2D-O^{2}NMNT using the RC method will be derived in this section. The O^{2}NMNT has a symmetrical and orthogonal kernel matrix, which has been described in Chapter 3, and therefore the basic equation of forward 2D-O^{2}NMNT and inverse 2D-O^{2}NMNT are the same other than the scaling factor, shown as

$$X(k_1, k_2) = \left\langle \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1, n_2) \beta \left[\frac{(2k_1+1)(2n_1+1)}{4}, \frac{(2k_2+1)(2n_2+1)}{4} \right] \right\rangle_{Mp} \quad (5.37)$$

$$\text{for } k_1, k_2 = 0, 1, 2, ..., N-1$$

and

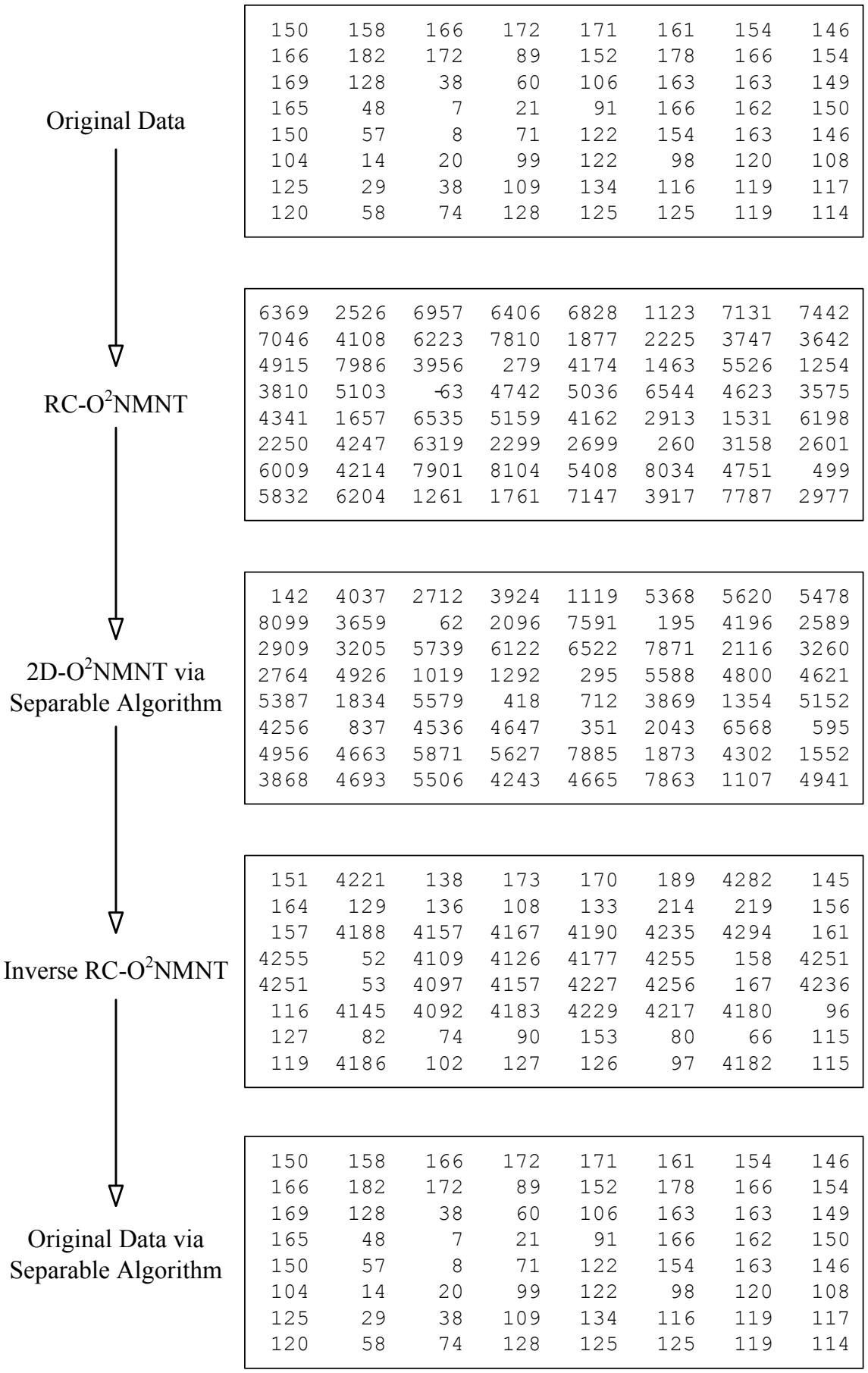
$$X(k_1, k_2) = \left\langle N^{-2} \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1, n_2) \beta \left[\frac{(2k_1+1)(2n_1+1)}{4}, \frac{(2k_2+1)(2n_2+1)}{4} \right] \right\rangle_{Mp}$$

$$\text{for } k_1, k_2 = 0, 1, 2, ..., N-1 \quad (5.38)$$

respectively. The RC definition for the 2D-O^{2}NMNT is

$$X_{RC}(k_1, k_2) = \left\langle \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1, n_2) \beta \left[\frac{(2k_1+1)(2n_1+1)}{4} \right] \beta \left[\frac{(2k_2+1)(2n_2+1)}{4} \right] \right\rangle_{Mp}$$

$$\text{for } k_1, k_2 = 0, 1, 2, ..., N-1. \quad (5.39)$$



150	158	166	172	171	161	154	146
166	182	172	89	152	178	166	154
169	128	38	60	106	163	163	149
165	48	7	21	91	166	162	150
150	57	8	71	122	154	163	146
104	14	20	99	122	98	120	108
125	29	38	109	134	116	119	117
120	58	74	128	125	125	119	114

6369	2526	6957	6406	6828	1123	7131	7442
7046	4108	6223	7810	1877	2225	3747	3642
4915	7986	3956	279	4174	1463	5526	1254
3810	5103	-63	4742	5036	6544	4623	3575
4341	1657	6535	5159	4162	2913	1531	6198
2250	4247	6319	2299	2699	260	3158	2601
6009	4214	7901	8104	5408	8034	4751	499
5832	6204	1261	1761	7147	3917	7787	2977

142	4037	2712	3924	1119	5368	5620	5478
8099	3659	62	2096	7591	195	4196	2589
2909	3205	5739	6122	6522	7871	2116	3260
2764	4926	1019	1292	295	5588	4800	4621
5387	1834	5579	418	712	3869	1354	5152
4256	837	4536	4647	351	2043	6568	595
4956	4663	5871	5627	7885	1873	4302	1552
3868	4693	5506	4243	4665	7863	1107	4941

151	4221	138	173	170	189	4282	145
164	129	136	108	133	214	219	156
157	4188	4157	4167	4190	4235	4294	161
4255	52	4109	4126	4177	4255	158	4251
4251	53	4097	4157	4227	4256	167	4236
116	4145	4092	4183	4229	4217	4180	96
127	82	74	90	153	80	66	115
119	4186	102	127	126	97	4182	115

150	158	166	172	171	161	154	146
166	182	172	89	152	178	166	154
169	128	38	60	106	163	163	149
165	48	7	21	91	166	162	150
150	57	8	71	122	154	163	146
104	14	20	99	122	98	120	108
125	29	38	109	134	116	119	117
120	58	74	128	125	125	119	114

Figure 5.3: Example of 2D-O^{2}NMNT using Row-Column Method

Following the same procedure in (5.29), $\beta\left[\frac{(2k_1+1)(2n_1+1)}{4}, \frac{(2k_2+1)(2n_2+1)}{4}\right]$ can be calculated as

$$\begin{aligned}
&\beta\left[\frac{(2k_1+1)(2n_1+1)}{4}, \frac{(2k_2+1)(2n_2+1)}{4}\right] \\
&= \beta\left[\frac{(2k_1+1)(2n_1+1)}{4} + \frac{(2k_2+1)(2n_2+1)}{4}\right] \\
&= \left\langle \left[\beta\left[\frac{(2k_1+1)(2n_1+1)}{4}\right]\beta\left[\frac{(2k_2+1)(2n_2+1)}{4}\right]\right.\right. \\
&\quad + \beta\left[-\frac{(2k_1+1)(2n_1+1)}{4}\right]\beta\left[\frac{(2k_2+1)(2n_2+1)}{4}\right] \\
&\quad + \beta\left[\frac{(2k_1+1)(2n_1+1)}{4}\right]\beta\left[-\frac{(2k_2+1)(2n_2+1)}{4}\right] \\
&\quad \left.\left. - \beta\left[-\frac{(2k_1+1)(2n_1+1)}{4}\right]\beta\left[-\frac{(2k_2+1)(2n_2+1)}{4}\right]\right] 2^{p-1} \right\rangle_{Mp}.
\end{aligned} \tag{5.40}$$

Finally, applying (5.41) to (5.37) and combing with X_{RC} produces

$$\begin{aligned}
X(k_1,k_2) = &\Big\langle \Big[X_{RC}(k_1,k_2) + X_{RC}(-k_1-1,k_2) \\
&+ X_{RC}(k_1,-k_2-1) - X_{RC}(-k_1-1,-k_2-1)\Big] 2^{p-1} \Big\rangle_{Mp}.
\end{aligned} \tag{5.41}$$

A demonstration of how this is presented is shown in Figure 5.3, which started by reducing the cameraman image to an 8×8 array and using the RC technique to transform the data to a 2D-O^{2}NMNT, using $N = 8$, $p = 13$ and $Mp = 8191$. The data is shown as it progresses through the various stages of the process, both forwards and backwards.

5.5 Complexity Analysis

Calculating the complexity of using the RC method and subsequent separable algorithm to obtain the 2D-GNMNT can easily be achieved by applying a series of modifications to the complexities of the 1D counterparts provided in Section 4.5. By noting that the RC method will in fact invoke a 1D method $2N$ times for each of the N rows and then N columns, we can respectively obtain

$$M_{RC}(N) = 2N^2(\log_2 N - 1)\lambda \tag{5.42}$$

and

$$A_{RC}(N) = 3N^2 \log_2 N + N^2 \tag{5.43}$$

for the radix-2 implementation;

$$M_{RC}(N) = \left[N^2 \left(3 \log_4 N - 1\right)\right] \lambda \tag{5.44}$$

and

$$A_{RC}(N) = \frac{N^2}{2} \left(11 \log_4 N - 1\right) \tag{5.45}$$

for the radix-4 implementation;

$$M_{RC}(N) = \left[\frac{4}{3} N^2 \log_2 N - \frac{8}{9} N^2 + \frac{8}{9} N (-1)^{\log_2 N}\right] \lambda \tag{5.46}$$

and

$$A_{RC}(N) = \frac{8}{3} N^2 \log_2 N - \frac{4}{9} N^2 + \frac{4}{9} N (-1)^{\log_2 N}. \tag{5.47}$$

for the split-radix implementation. The results using $\lambda = 2$ to normalise the multiplications against the additions can be observed in Figure 5.4. In Section 4.5, the notion of using more up to date instructions that combine more than one operator was introduced using the FMA command. Applying this technique for the RC method using the split-radix algorithm produces

$$F_{RC}(N) = \left[\frac{2}{3} N^2 \log_2 N - \frac{4}{9} N^2 + \frac{4}{9} N (-1)^{\log_2 N}\right] \lambda, \tag{5.48}$$

$$M_{RC}(N) = \left[\frac{2}{3} N^2 \log_2 N - \frac{4}{9} N^2 + \frac{4}{9} N (-1)^{\log_2 N}\right] \lambda \tag{5.49}$$

and

$$A_{RC}(N) = 2N^2 \log_2 N, \tag{5.50}$$

to produce a total operational count of

$$T_{RC}(N) = \frac{4\lambda + 6}{3} N^2 \log_2 N - \frac{8\lambda}{9} N^2 + \frac{8\lambda}{9} N (-1)^{\log_2 N}. \tag{5.51}$$

To complete the process of taking the results from the RC algorithm to the 2D algorithm requires the application of the separable algorithm. This stage has a significant impact, as shown in Figure 5.5, owing that the multiplication and three additions that are required for each point. Therefore, applying a further modification of N^2 multiplications and $3N^2$ additions to (5.42) and (5.43) produces

$$M_S(N) = 2N^4 \left(\log_2 N - 1\right) \lambda \tag{5.52}$$

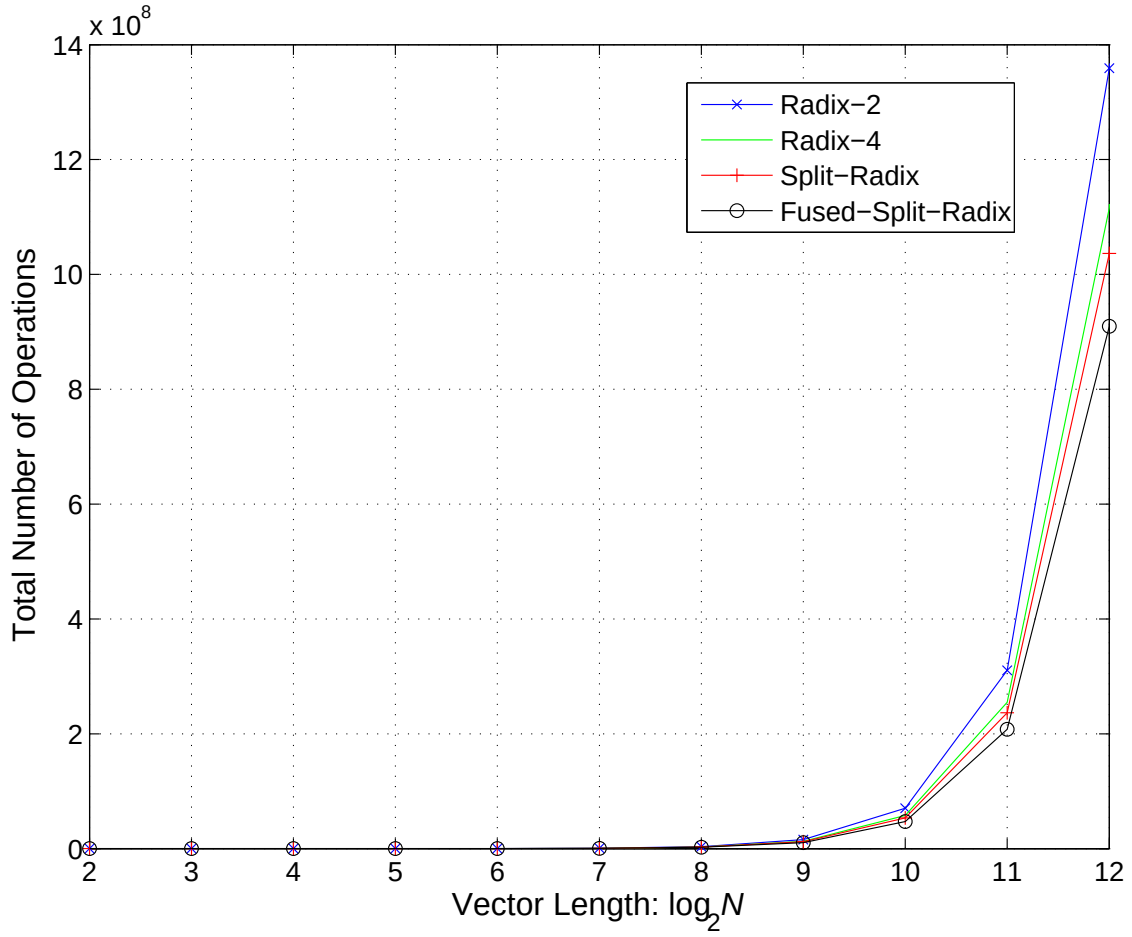


Figure 5.4: Complexity of Different Radices for Row-Column by Total Operations

and

$$A_S(N) = 9N^4 \log_2 N + 3N^4 \tag{5.53}$$

for the radix-2 implementation; to (5.44) and (5.45) produces

$$M_S(N) = \left[N^4 \left(3 \log_4 N - 1\right)\right] \lambda \tag{5.54}$$

and

$$A_S(N) = \frac{3N^4}{2} \left(11 \log_4 N - 1\right), \tag{5.55}$$

for the radix-4 implementation and finally to (5.46) and (5.47) produces

$$M_S(N) = \left[\frac{4}{3} N^4 \log_2 N - \frac{8}{9} N^4 + \frac{8}{9} N^3 (-1)^{\log_2 N}\right] \lambda \tag{5.56}$$

and

$$A_S(N) = 8N^4 \log_2 N - \frac{4}{3} N^4 + \frac{4}{3} N^3 (-1)^{\log_2 N} \tag{5.57}$$

for the split-radix implementation of the separable algorithm. As the separable algorithm consists of three additions then a multiplication, applying this adjustment

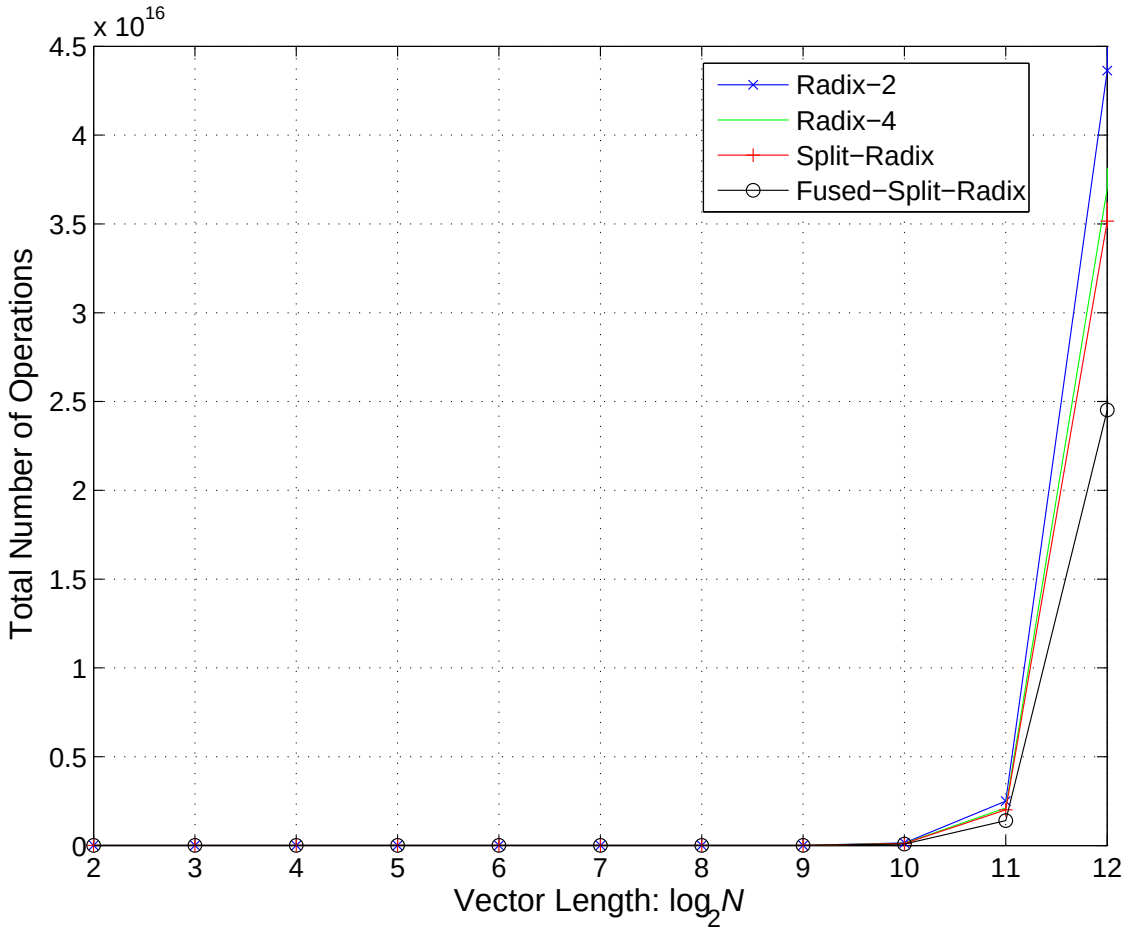


Figure 5.5: Complexity of Different Radices for Row-Column with Separable Algorithm by Total Operations

to the FMA version in (5.48)-(5.51) is limited to the multiplications and additions without offloading operations to the FMA operator. Therefore, adjusting (5.49) with an additional N^2 multiplications produces

$$M_S(N) = \left[\frac{2}{3}N^4 \log_2 N - \frac{4}{9}N^4 + \frac{4}{9}N^3(-1)^{\log_2 N}\right]\lambda, \tag{5.58}$$

adjusting (5.51) with an additional $3N^2$ multiplications produces

$$A_S(N) = 6N^4 \log_2 N, \tag{5.59}$$

and combining (5.48) with (5.58) and (5.59) produces a total operational count of

$$\begin{aligned} T_S(N) =& \frac{2\lambda+3}{3}N^4 \log_2 N - \frac{4\lambda}{3}N^4 + \frac{4\lambda}{3}N^3(-1)^{\log_2 N} \\ &+ \frac{2\lambda}{3}N^2 \log_2 N - \frac{4\lambda}{3}N^2 + \frac{4\lambda}{3}N(-1)^{\log_2 N}. \end{aligned} \tag{5.60}$$

5.6 The 2D Cyclic Convolution for the GNMNT

The 1D cyclic convolution for the GNMNT was discussed previously in Chapter 3. A common application of the 2D cyclic convolution is image processing, where filters can be applied quickly and efficiently. The derivations for the various convolutions forthwith were first derived by [7] for the NMNT and [8, 139] for the expansion to the GNMNT, and have been included here for the sake of completeness with further insights and demonstrations. Typically, the signals $x(n_1, n_2)$ and $h(n_1, n_2)$ are the inputs containing the image and filter of size $N \times N$, while $y_{CC}(n_1, n_2)$ provides the output of the 2D convolution using the GNMNT.

5.6.1 Cyclic Convolution for the 2D-NMNT

Denoting $X(k_1, k_2)$, $H(k_1, k_2)$ and $Y_{CC}(k_1, k_2)$ as the signal values within the NMNT domain of $x(n_1, n_2)$, $h(n_1, n_2)$ and $y_{CC}(n_1, n_2)$, the 2D-NMNT Cyclic Convolution is calculated as

$$\begin{aligned} Y_{CC}(k_1, k_2) = \Big\langle & 2^{p-1}\left[H(k_1, k_2) + H(N-k_1, N-k_2)\right] X(k_1, k_2) \\ & + 2^{p-1}\left[H(k_1, k_2) - H(N-k_1, N-k_2)\right] X(N-k_1, N-k_2) \Big\rangle_{Mp} \end{aligned} \tag{5.61}$$

As previously described in [7], a new operator can be used to simplify (5.61), for which $\odot$ will be used, so that

$$\begin{aligned} Y_{CC}(k_1, k_2) &= X(k_1, k_2) \odot H(k_1, k_2) \\ &= \Big\langle H^{ev}(k_1, k_2) X(k_1, k_2) \\ &\quad + H^{od}(k_1, k_2) X(N-k_1, N-k_2) \Big\rangle_{Mp} \end{aligned} \tag{5.62}$$

where $H^{ev}(k_1, k_2)$ and $H^{od}(k_1, k_2)$ are the even and odd parts of $H(k_1, k_2)$ shown respectively as

$$H^{ev}(k_1, k_2) = \Big\langle \left[H(k_1, k_2) + H(N-k_1, N-k_2)\right] 2^{p-1} \Big\rangle_{Mp} \tag{5.63}$$

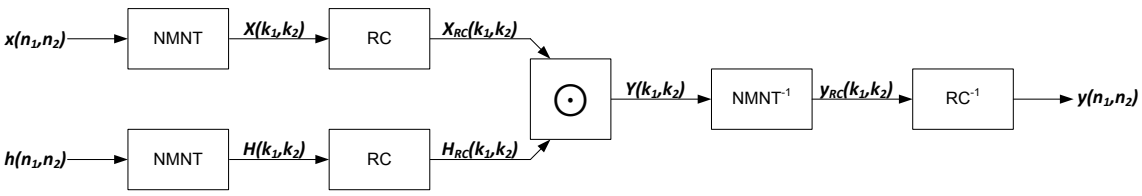


Figure 5.6: Convolution Process Structure for the 2D-NMNT

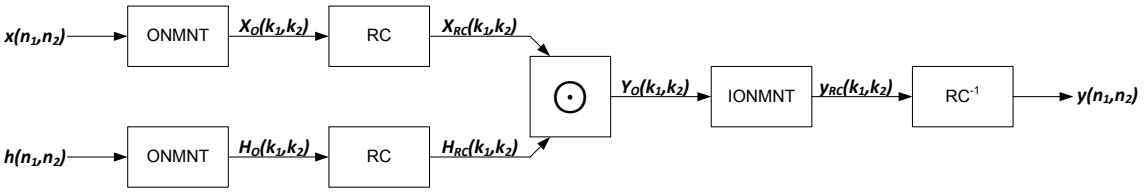


Figure 5.7: Convolution Process Structure for the 2D-ONMNT

and

$$H^{od}(k_1,k_2) = \left\langle \left[H\left(k_1,k_2\right) - H\left(N-k_1,N-k_2\right)\right] 2^{p-1} \right\rangle_{Mp}. \tag{5.64}$$

Figure 5.6 further illustrates the process of the 2D-NMNT cyclic convolution, which shows two RC blocks that are slightly different from the 1D convolution owing to the RC method being applied to the 2D-GNMNT.

5.6.2 Cyclic Convolution for the 2D-ONMNT

The variables $X_O\left(k_1,k_2\right)$, $H_O\left(k_1,k_2\right)$ and $Y_{OCC}\left(k_1,k_2\right)$ are the signals $x\left(n_1,n_2\right)$, $h\left(n_1,n_2\right)$ and $y_{CC}\left(n_1,n_2\right)$ that are within the ONMNT domain. The operation to derive the ONMNT result $Y_{OCC}\left(k_1,k_2\right)$ is processed using $X_O\left(k_1,k_2\right)$, $H_O\left(k_1,k_2\right)$ by

$$\begin{aligned} Y_{OCC}\left(k_1,k_2\right) &= X_O\left(k_1,k_2\right) \odot H_O\left(k_1,k_2\right) \\ &= \Big\langle H_O^{ev}(k_1,k_2) X_O\left(k_1,k_2\right) \\ &\quad + H_O^{od}(k_1,k_2) X_O\left(N-k_1-1,N-k_2-1\right) \Big\rangle_{Mp} \end{aligned} \tag{5.65}$$

where $H_O^{ev}(k_1,k_2)$ and $H_O^{od}(k_1,k_2)$ stand for even and odd parts of $H_O\left(k_1,k_2\right)$ respectively as

$$H_O^{ev}(k_1,k_2) = \left\langle \left[H_O\left(k_1,k_2\right) + H_O\left(N-k_1-1,N-k_2-1\right)\right] 2^{p-1} \right\rangle_{Mp} \tag{5.66}$$

and

$$H_O^{od}(k_1,k_2) = \left\langle \left[H_O\left(k_1,k_2\right) - H_O\left(N-k_1-1,N-k_2-1\right)\right] 2^{p-1} \right\rangle_{Mp}. \tag{5.67}$$

The process of computing the 2D cyclic convolution for the ONMNT is shown in Figure 5.7.

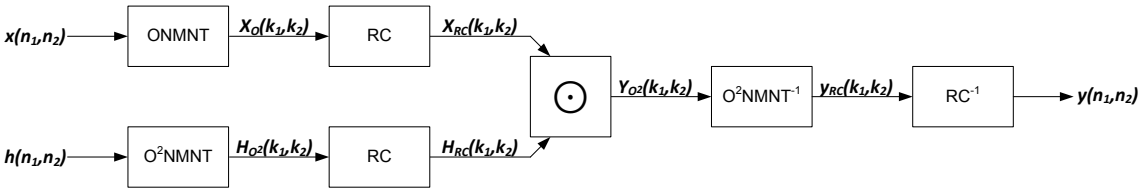


Figure 5.8: Convolution Process Structure for the 2D-O^{2}NMNT

5.6.3 Cyclic Convolution for the 2D-O^{2}NMNT

Transforming into the O^{2}NMNT domain, the input and output signals $x\left(n_1, n_2\right)$, $h\left(n_1, n_2\right)$ and $y_{CC}\left(n_1, n_2\right)$ are represented by $X_{O^2}\left(k_1, k_2\right)$, $H_{O^2}\left(k_1, k_2\right)$ and $Y_{{O^2}CC}\left(k_1, k_2\right)$ respectively and processed as

$$\begin{aligned} Y_{{O^2}CC}\left(k_1, k_2\right) &= X_O\left(k_1, k_2\right) \odot H_{O^2}\left(k_1, k_2\right) \\ &= \Big\langle H_{O^2}^{od}(k_1, k_2) X_O\left(k_1, k_2\right) \\ &\quad + H_{O^2}^{ev}(k_1, k_2) X_O\left(N-k_1-1, N-k_2-1\right) \Big\rangle_{Mp} \end{aligned} \tag{5.68}$$

where $H_{O^2}^{ev}(k_1, k_2)$ and $H_{O^2}^{od}(k_1, k_2)$ stand for even and odd parts of $H_{O^2}\left(k_1, k_2\right)$ respectively as

$$H_{O^2}^{ev}(k_1, k_2) = \Big\langle \left[H_{O^2}\left(k_1, k_2\right) + H_{O^2}\left(N-k_1-1, N-k_2-1\right)\right] 2^{p-1} \Big\rangle_{Mp} \tag{5.69}$$

and

$$H_{O^2}^{od}(k_1, k_2) = \Big\langle \left[H_{O^2}\left(k_1, k_2\right) - H_{O^2}\left(N-k_1-1, N-k_2-1\right)\right] 2^{p-1} \Big\rangle_{Mp}. \tag{5.70}$$

Figure 5.8 shows the process of computing 2D cyclic convolution for O^{2}NMNT.

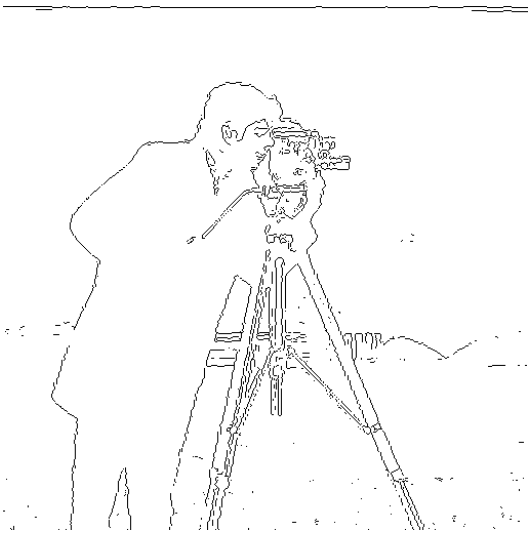
5.6.4 Verification using Cyclic Convolution

An example application for using convolution would be to apply the Sobel filter to an image, which is used to provide edge detection [140]; this is configured as

$$H = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}. \tag{5.71}$$

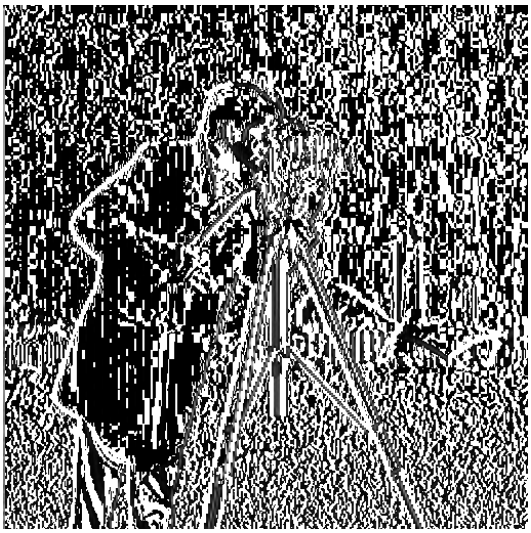
Using the NMNT to process the required convolutions is shown Figure 5.9, where the original image Figure 5.9(a) is first processed using MATLAB for reference, producing Figure 5.9(b). Applying the Sobel filter to the image Y via convolution

(a) Original Image



(b) MATLAB Implementation

(c) Sobel Filter x-Axis Derivative



(d) Sobel Filter y-Axis Derivative

(e) Magnitude using xy-Derivatives

(f) GNMNT Implementation

Figure 5.9: Convolution using Sobel Filter for Edge Detection using Cameraman Image with NMNT

(a) Original Image

(b) MATLAB Implementation

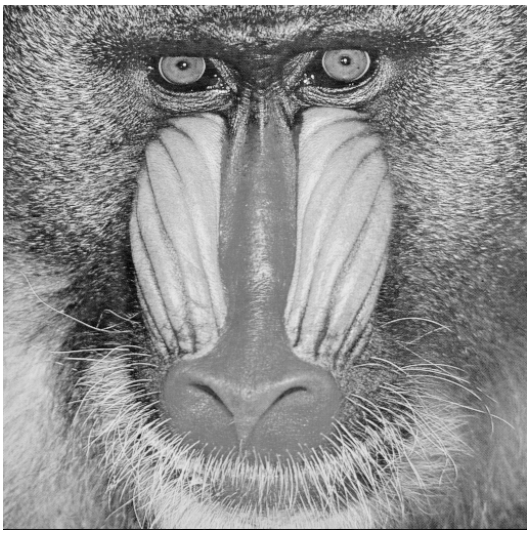
(c) Sobel Function x-Axis Derivative

(d) Sobel Function y-Axis Derivative

(e) Magnitude using xy-Functions

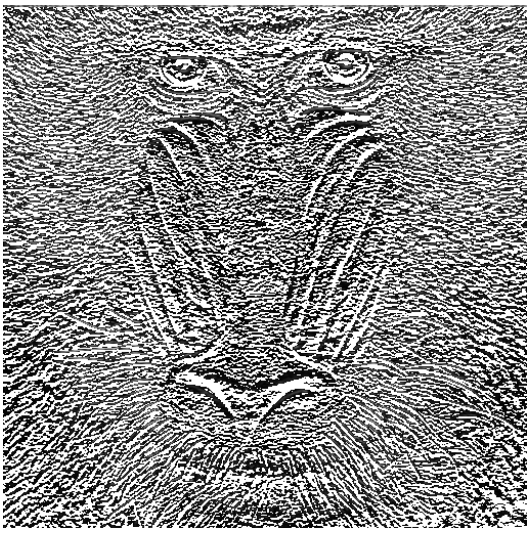
(f) GNMNT Implementation

Figure 5.10: Convolution using Sobel Filter for Edge Detection using Lena Image with ONMNT

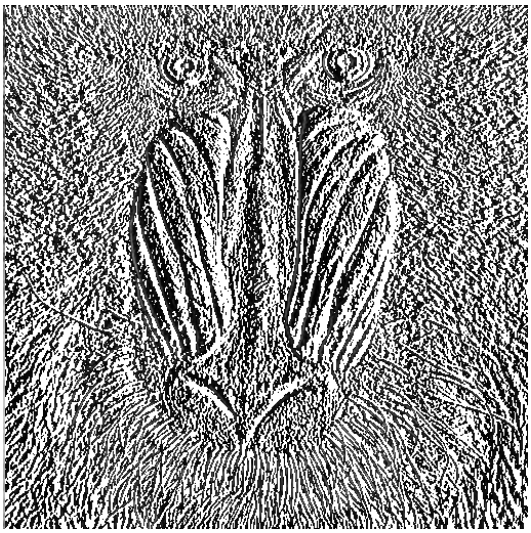


(a) Original Image

(b) MATLAB Implementation



(c) Sobel Function x-Axis Derivative



(d) Sobel Function y-Axis Derivative

(e) Magnitude using xy-Functions

(f) GNMNT Implementation

Figure 5.11: Convolution using Sobel Filter for Edge Detection using Baboon Image with O^2NMNT

across the x- and y-axes produces

$$G_x = H \circledast Y \tag{5.72}$$

and

$$G_y = H' \circledast Y. \tag{5.73}$$

The results of (5.72) and (5.73) using the NMNT are shown in Figures 5.9(c) and 5.9(d) respectively. The gradient map in Figure 5.9(e) is obtained by combining the two direction sources by

$$G = \sqrt{G_{x^2} + G_{y^2}} \tag{5.74}$$

The process concludes by defining a threshold level, which in this case was accomplished by calculating the mean of the gradient map

$$T_G = \overline{G} \tag{5.75}$$

and applying this threshold to define the boundaries for G yields Figure 5.9(f). It appears that MATLAB may use additional processing techniques such as line thinning for example [141]. Figures 5.10 and 5.11 depict the same processes using the ONMNT and O^{2}NMNT respectively, thus validating the implementations.

Table 5.1: Effective Lengths of 1D and 2D GNMNTs

	1D Block and Key Size					2D Block and Key Size				
	p					p				
N	5	7	13	17	19	5	7	13	17	19
4	20	28	52	68	76	80	112	208	272	304
8	40	56	104	136	152	320	448	832	1088	1216
16	80*	112	208	272	304	1280*	1792	3328	4352	4864
32	160#	224	416	544	608	5120#	7168	13312	17408	19456
64	-	448*	832	1088	1216	-	28672*	53248	69632	77824
128	-	896#	1664	2176	2432	-	114688#	212992	278528	311296
256	-	-	3328	4352	4864	-	-	851968	1114112	1245184
512	-	-	6656	8704	9728	-	-	3407872	4456448	4980736
1024	-	-	13312	17408	19456	-	-	13631488	17825792	19922944

* NMNT and ONMNT, # NMNT Only

5.7 Encryption Applications

An application for using the 2D algorithm of the GNMNT for encryption is where two people have the same image. This can be combined with a password for an implementation of the two-factor authentication (2FA) protocol that is applied to encryption: something you know and something you have [142]. Selecting a random area of the image and transferring the coordinates produces an ephemeral-type key, which can be communicated to the receiving party using similar techniques to those used to transmit the password. However, in order to safeguard the integrity of the encryption, it would be advisable to transfer the image attributes and the regular password using two distinct public key techniques. Such techniques enhance security by adding an additional layer of security. There are a number of methods used to provide 2FA including documentation, text message, email, hardware token or even software [143], which have become highly adopted by many industries, particularly by the finance sector.

This section will provide examples on this methodology by incorporating an encryption system that uses a 2D key from an image that both parties are known to have. The method of image selection, image key could well be communicated using a public key encryption system such as RSA or ECC. Once an image and image

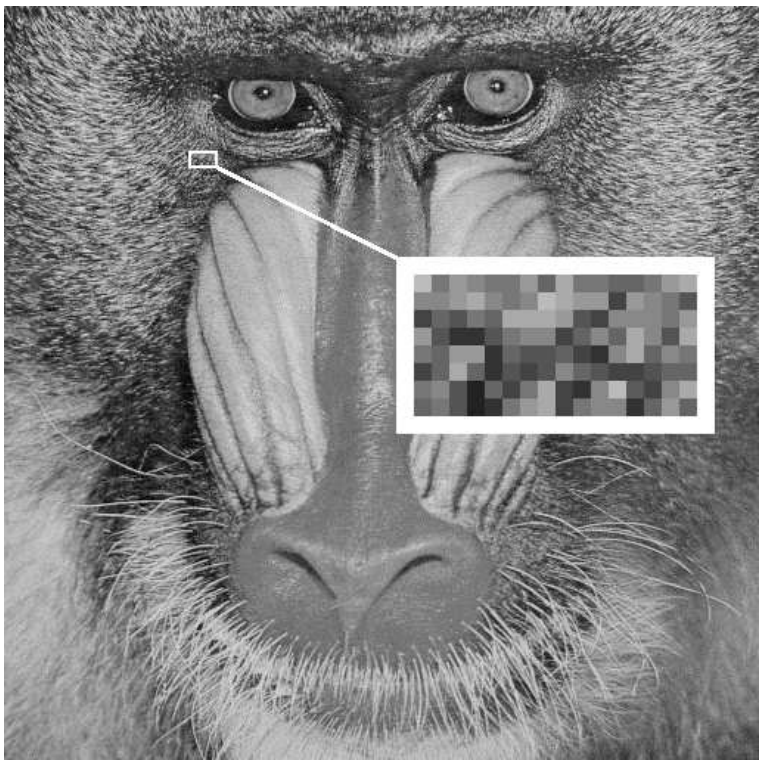


Figure 5.12: Image from which the selected 2D key is obtained

Table 5.2: Pixel Values of 2D Key without Concatenation

0xb1	0x7e	0x9e	0x87	0x71	0x80	0x95	0x63	0xb0	0x76	0x7b	0x68	0x6a	0x7a	0x97	0xa0
0x90	0x80	0x9d	0x9f	0x8e	0x79	0x8a	0xbe	0xad	0x92	0x90	0x4b	0x9a	0x8d	0x78	0x5f
0x42	0x6f	0x5f	0x75	0x6d	0xa5	0xa8	0xaf	0x95	0x58	0x4e	0x85	0x82	0x86	0x73	0x9c
0x94	0x51	0x37	0x39	0x4e	0x7f	0x82	0x86	0x44	0x5f	0x88	0x64	0xa9	0x87	0x5e	0xa1
0x78	0x69	0x94	0x9b	0x36	0x62	0x54	0x57	0x72	0x45	0x40	0x8b	0xa6	0x6d	0x87	0x77
0x44	0x64	0xa6	0x6b	0x32	0x5a	0x47	0x4f	0x87	0x68	0x5b	0x3c	0x49	0x55	0x6b	0x6b
0x72	0x91	0x55	0x1e	0x68	0x4e	0x78	0xac	0x48	0x39	0x86	0xb1	0x5b	0x3f	0x8d	0xa4
0x67	0x76	0x59	0x1e	0x38	0x74	0x9c	0xa0	0x46	0x72	0x80	0x87	0x3a	0x64	0x99	0x61

Table 5.3: Pixel Values of 2D Key with Concatenation

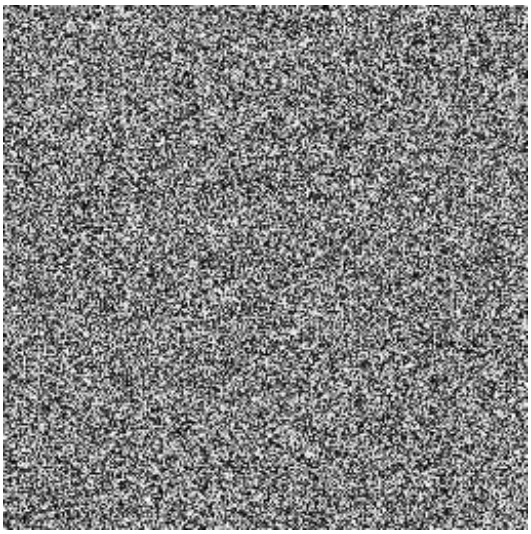
0xb17e	0x9e87	0x7180	0x9563	0xb076	0x7b68	0x6a7a	0x97a0
0x9080	0x9d9f	0x8e79	0x8abe	0xad92	0x904b	0x9a8d	0x785f
0x426f	0x5f75	0x6da5	0xa8af	0x9558	0x4e85	0x8286	0x739c
0x9451	0x3739	0x4e7f	0x8286	0x445f	0x8864	0xa987	0x5ea1
0x7869	0x949b	0x3662	0x5457	0x7245	0x408b	0xa66d	0x8777
0x4464	0xa66b	0x325a	0x474f	0x8768	0x5b3c	0x4955	0x6b6b
0x7291	0x551e	0x684e	0x78ac	0x4839	0x86b1	0x5b3f	0x8da4
0x6776	0x591e	0x3874	0x9ca0	0x4672	0x8087	0x3a64	0x9961

position have been selected, this will then act as a private session key between the two parties.

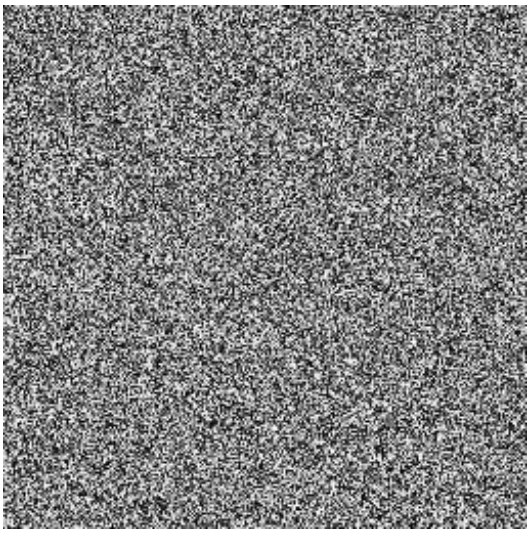
5.7.1 RC-GNMNT Implementations

One of the advantages of the NMNT and GNMNT transforms are their highly adaptable lengths [7, 8]. However, irrespective of these lengths, limitations are still put in place with respect to the prime that is defined by p. Using a 2D algorithm allows even greater versatility by essentially squaring the lengths over the same value of p, thereby providing significant increases in these lengths as shown in Table 5.1. In the following examples, a key of size based on an image area of 16×8 in size has been selected from the image shown in Figure 5.12. This key is shown numerically in Table 5.2 as it was extracted from the greyscale image and again in Table 5.3 after the concatenation process has taken place. Data within the key was concatenated in pairs to reduce the size from 16 columns to 8 so that the length $N = 8$ could be assigned to a square area. A value of $p = 17$ was selected to derive a Mersenne prime number of 131071, which completes the configuration of the encryption system. The system uses a similar procedure that was defined in

(a) Original



(b) Cipher



(c) Incorrect Key

(d) Correct Key

Figure 5.13: Encrypted Cameraman using RC-NMNT, $N = 8 \times 8$ and $Mp = 131071$

(3.81)-(3.91) adapted as a 2D system using a 2D key for all examples, where only the transform type was changed according to the appropriate subsection.

5.7.1.1 RC-NMNT

The results of encrypting the cameraman image using the RC-NMNT and a 2D key are shown in Figure 5.13. As anticipated from previous results using 1D-implementations, it can be seen that the NMNT has performed at a comparable level. The decryption using a different key has rendered the resultant deciphering operation completely worthless with no information apparently being released. Moreover, it would appear that there may not be a necessary requirement in computing the separable algorithm in order to completely process the image using

(a) Original (b) Cipher

(c) Incorrect Key (d) Correct Key

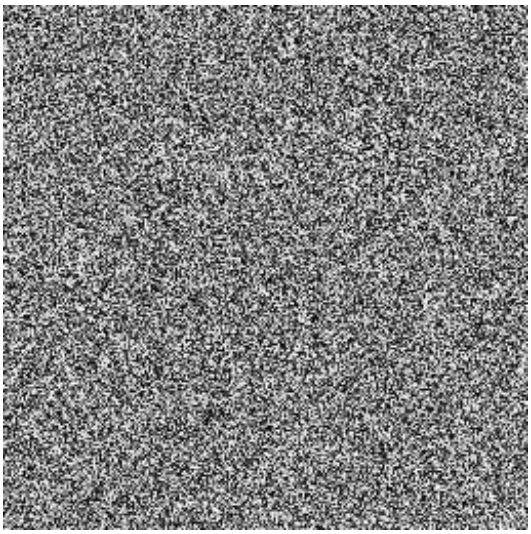
Figure 5.14: Encrypted Camerman using RC-ONMNT, $N = 8 \times 8$ and $Mp = 131071$

the RC method.

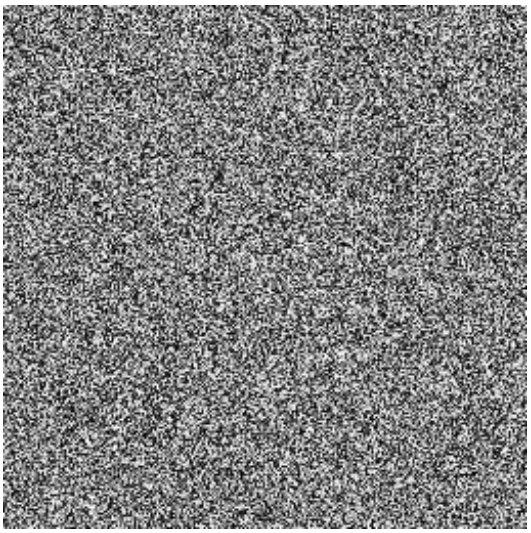
5.7.1.2 RC-ONMNT

Using the first of the two new transforms for RC encryption suggests that there are very promising results. Deciphering the image has rendered it worthless, similar to the results obtained from the NMNT. After applying the correct key, it is shown in Figure 5.14 that the system correctly deciphered the image, suggesting that encryption system is working as expected.

(a) Original



(b) Cipher



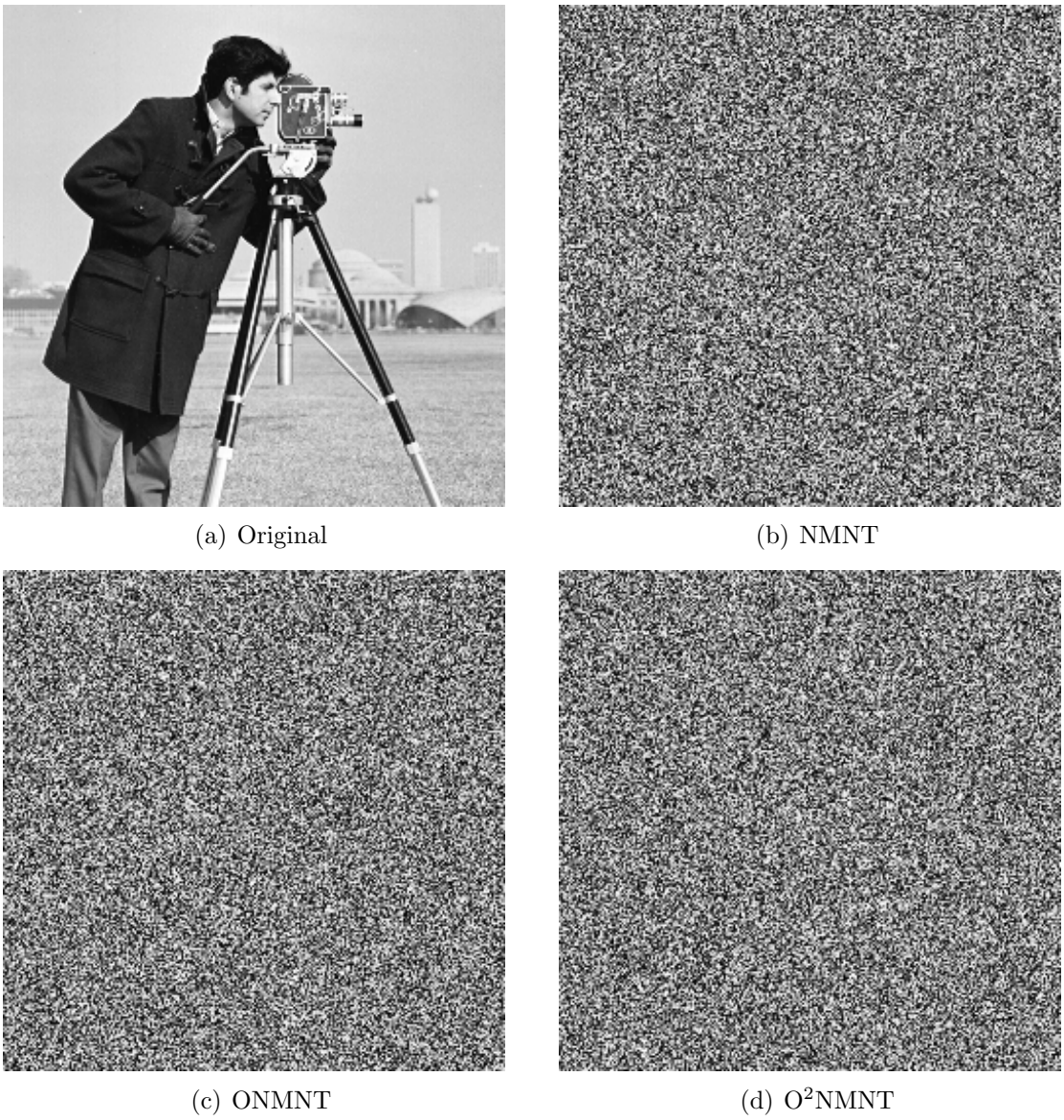
(c) Incorrect Key

(d) Correct Key

Figure 5.15: Encrypted Camerman using RC-O^{2}NMNT, $N = 8 \times 8$ and $Mp = 131071$

5.7.1.3 RC-O^{2}NMNT

The second of the two new transforms applied for encryption using the RC method is the O^{2}NMNT. Like its counterparts, the O^{2}NMNT has successfully encrypted and decrypted the image using the correct key, while failing to decipher the image using the incorrect key as shown in Figure 5.15. Again, this would initially suggest that the system using the O^{2}NMNT would be applicable for use in the field of cryptography.



(a) Original (b) NMNT

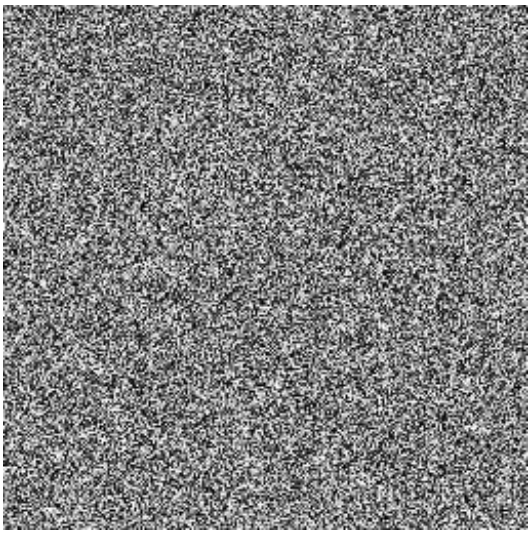
(c) ONMNT (d) O^2NMNT

Figure 5.16: Decrypted Cameraman Error using RC-GNMNT, $N = 8 \times 8$ and $Mp = 131071$

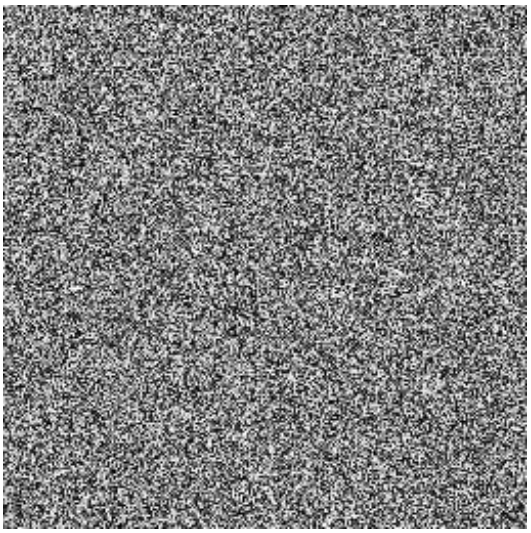
5.7.1.4 Comparison of RC-Encryption Characteristics

The different variations of the enciphered images using the same key but different transforms is depicted in Figure 5.16. Whilst there is nothing to be gained directly from observing this image, it is presented to demonstrate that the results from using each transform to decrypt the image using the same alternate key and also that they have provided different results according to each transform. Each of the images was previously shown to correctly encrypt and decrypt the image when using the same key.

(a) Original



(b) Cipher



(c) Incorrect Key

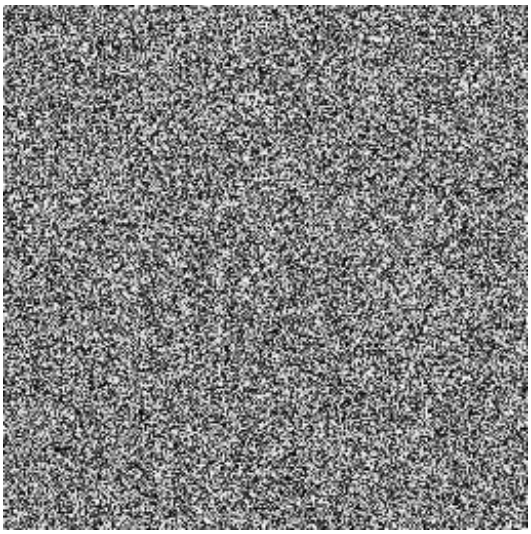
(d) Correct Key

Figure 5.17: Encrypted Cameraman using 2D-NMNT, $N = 8 \times 8$ and $Mp = 131071$

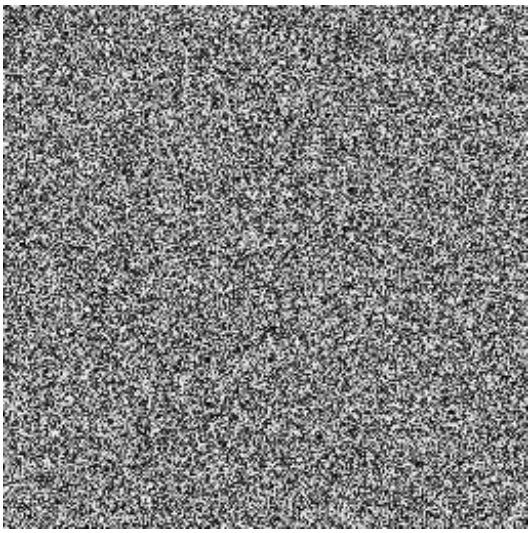
5.7.2 2D-GNMNT Implementations

This section is similar to the RC-GNMNT except that the additional separable algorithm has been applied to provide a true 2D transformation of the image in each system. While no further processing has been done in respect to 2D signal- or image-processing, the underlying implementation has been developed and tested to ensure compatibility throughout these transforms. The development of true 2D algorithms in cryptography opens many opportunities to further incorporate different methods such as filtering to enhance the framework of a developed encryption system.

(a) Original



(b) Cipher



(c) Incorrect Key

(d) Correct Key

Figure 5.18: Encrypted Camerman using 2D-ONMNT, $N = 8 \times 8$ and $Mp = 131071$

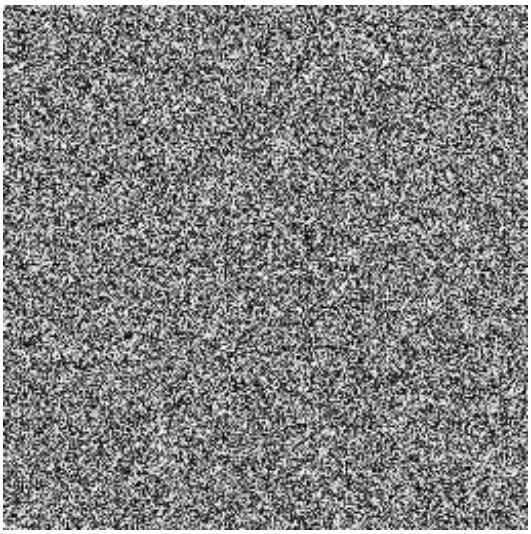
5.7.2.1 2D-NMNT

Like the RC-NMNT, the 2D-NMNT has correctly encrypted and decrypted the image using the same key. When using the alternate key to decrypt, again like the RC-GNMNT, the results yielded an unusable image where no information of the original contents appears to be obtainable, as shown in Figure 5.17.

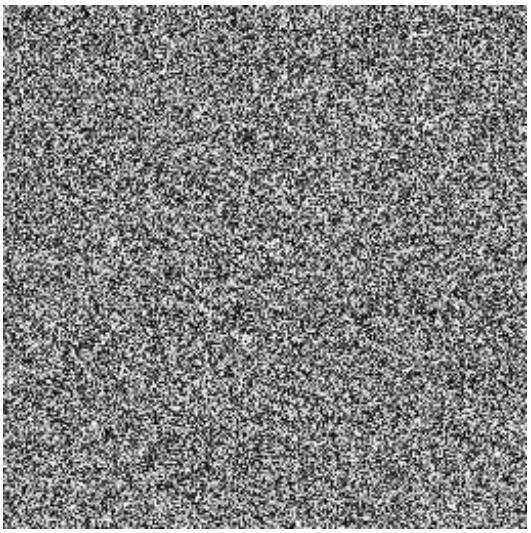
5.7.2.2 2D-ONMNT

Using the first of the new transforms and the separable algorithm has again yielded favourable results as shown in Figure 5.18. The results that were obtained from the result of the RC-ONMNT appear to be comparable to the 2D variant using the separable algorithm.

(a) Original



(b) Cipher



(c) Incorrect Key

(d) Correct Key

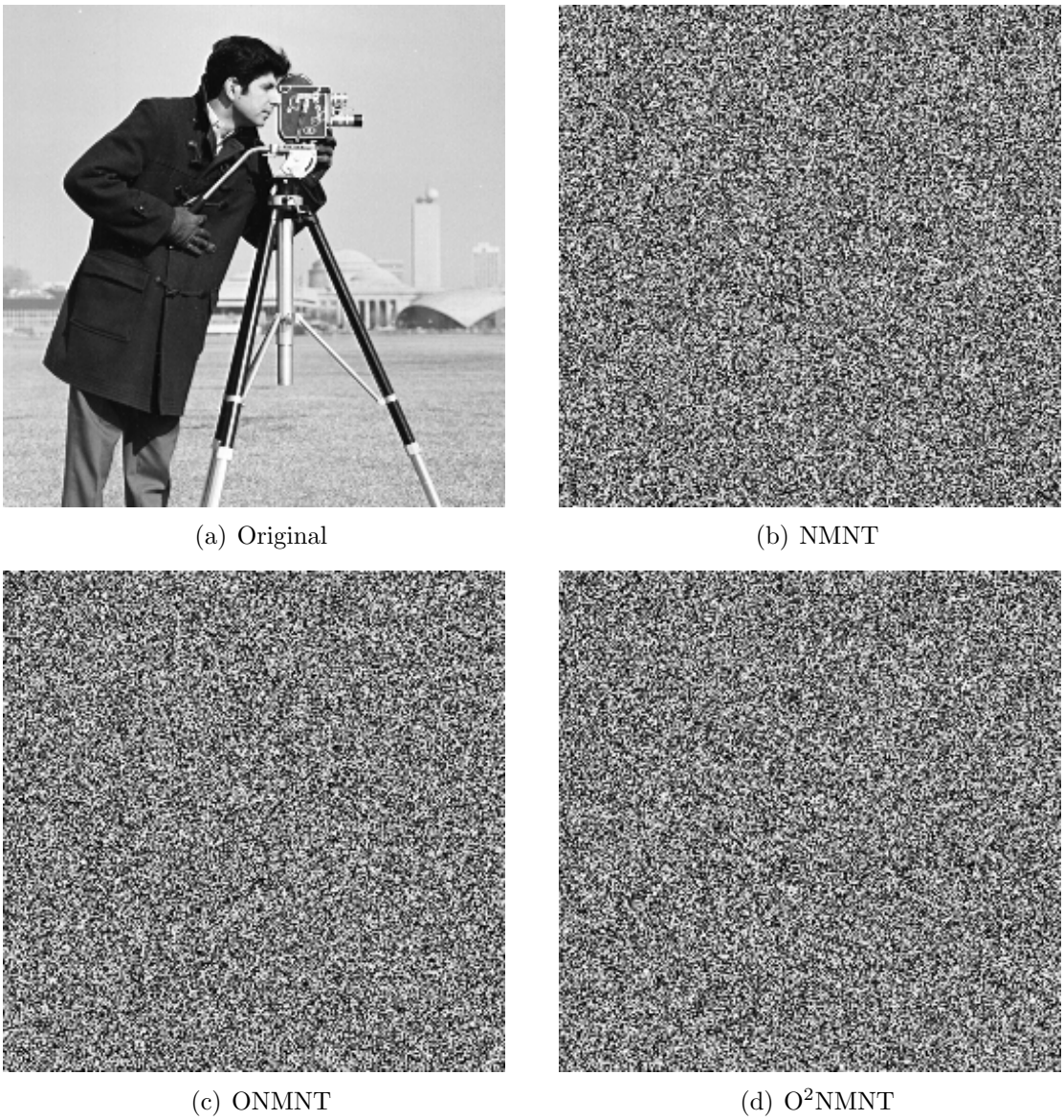
Figure 5.19: Encrypted Camerman using 2D-O^{2}NMNT, $N = 8{\times}8$ and $Mp = 131071$

5.7.2.3 2D-O^{2}NMNT

The final example of 2D encryption shows in Figure 5.19 that the second new transform has also demonstrated favourable results. The encryption and decryption process using the same key to obtain the original image and using different keys to show that recovering the original image is unobtainable has reached expectations and, like the other implementations of the GNMNT in these simulations, the O^{2}NMNT has performed comparably.

5.7.2.4 Comparison of 2D-Encryption Characteristics

The final image in these simulations again shows in Figure 5.20 the comparison of the resulting images using the 2D transforms where the same alternate



(a) Original (b) NMNT

(c) ONMNT (d) O^2NMNT

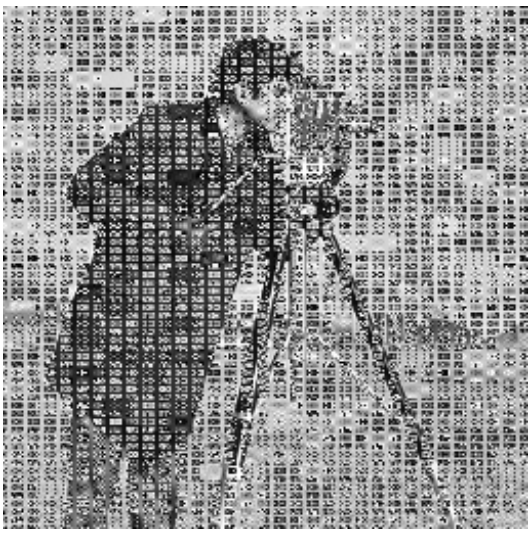
Figure 5.20: Decrypted Cameraman Error using 2D-GNMNT, $N = 8 \times 8$ and $Mp = 131071$

decrypting keys were used instead of the same encrypting keys. Similar to the RC implementations, there is no direct information to be obtained from these images other than to show that by using the same key to encrypt and using the same incorrect key to decrypt, each of the transforms used has produced different results.

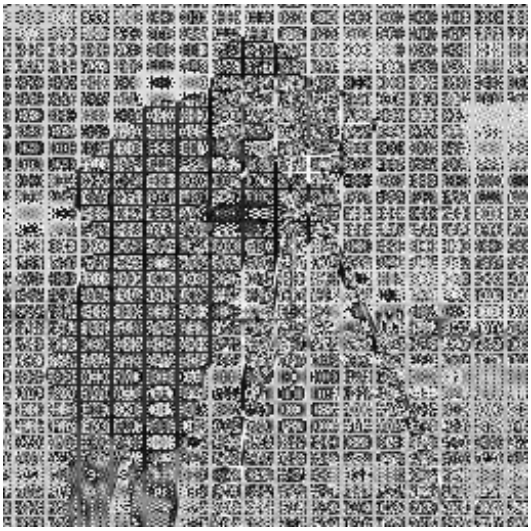
5.7.3 Imposing a Single-Bit Error using the RC- and 2D-GNMNT

With previous respect to Section 3.8.1, it would be remiss to ignore the results of changing a single bit of the GNMNT using the RC method to assess whether there

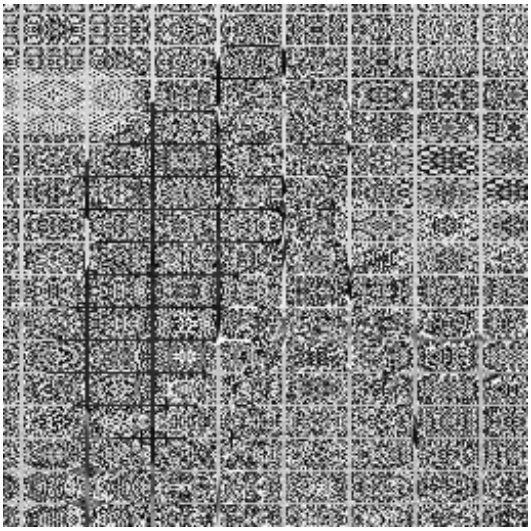
(a) Original Image



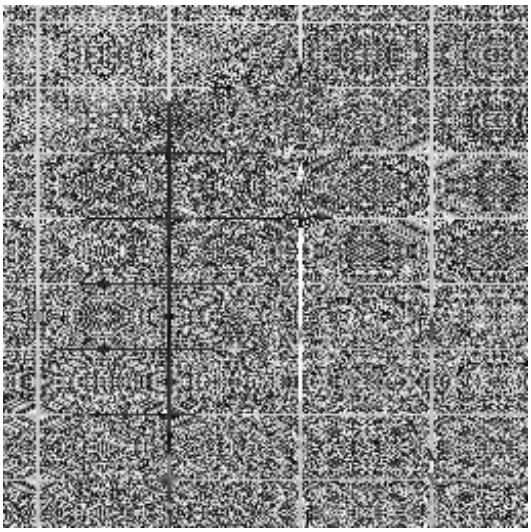
(b) RC-NMNT Bit Error $N = 8$



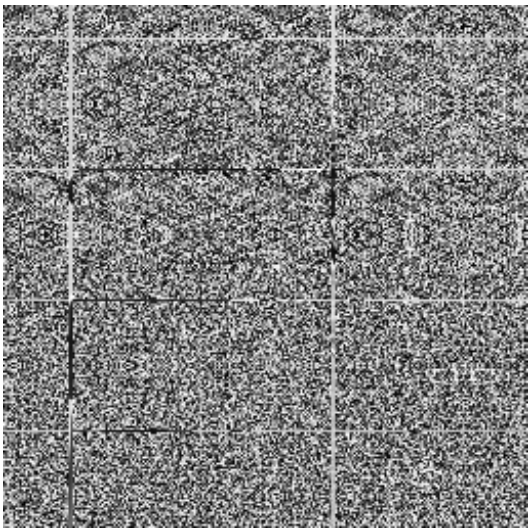
(c) RC-NMNT Bit Error $N = 16$



(d) RC-NMNT Bit Error $N = 32$



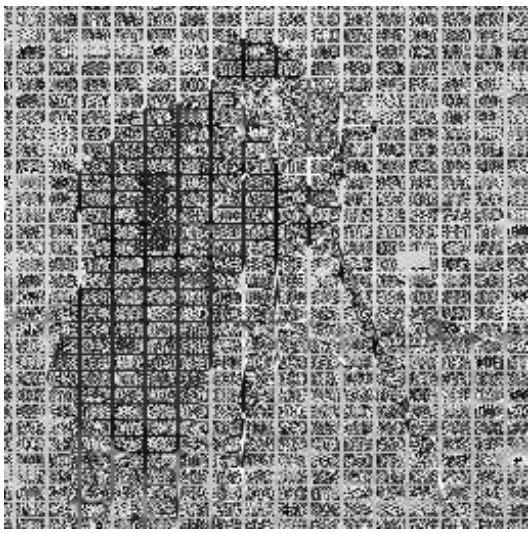
(e) RC-NMNT Bit Error $N = 64$



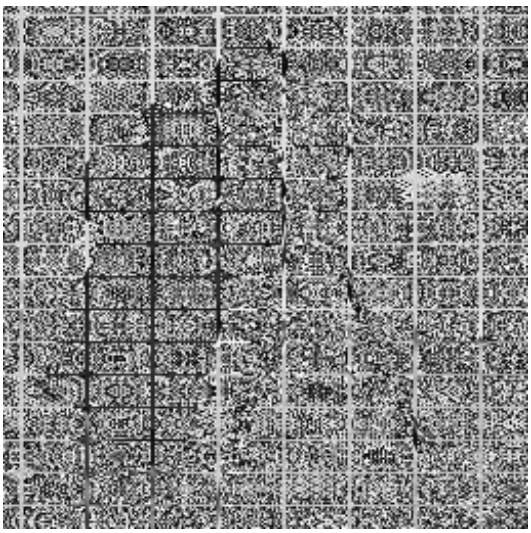
(f) RC-NMNT Bit Error $N = 128$

Figure 5.21: Decrypted Bit Error using RC-NMNT with $N = 8, 16, 32, 64, 128$ and $Mp = 131071$

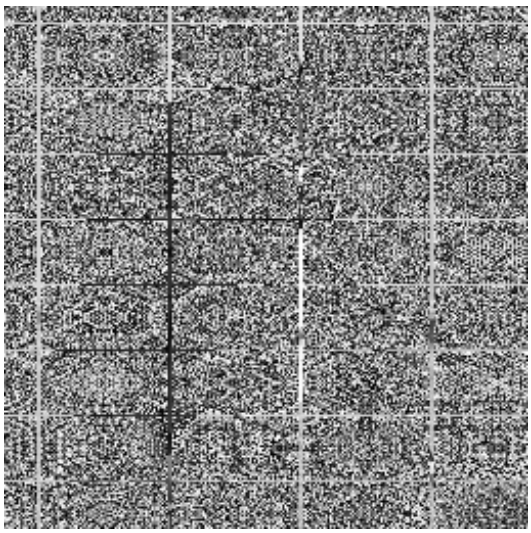
(a) Original Image



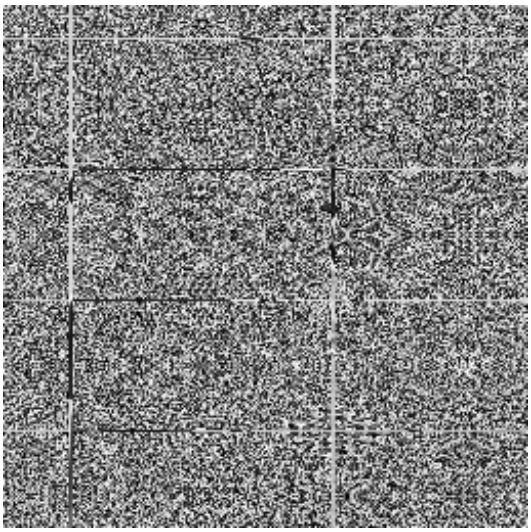
(b) RC-ONMNT Bit Error $N = 8$



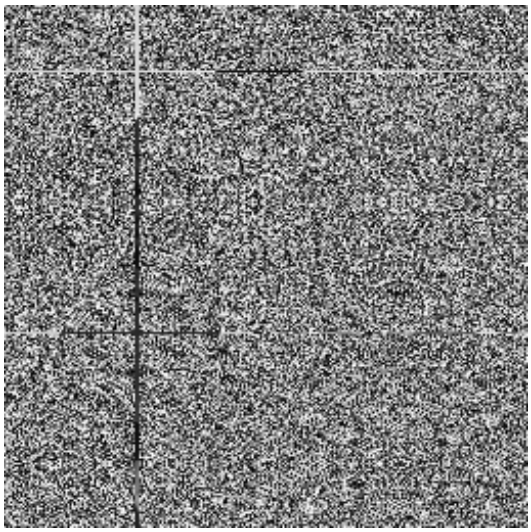
(c) RC-ONMNT Bit Error $N = 16$



(d) RC-ONMNT Bit Error $N = 32$



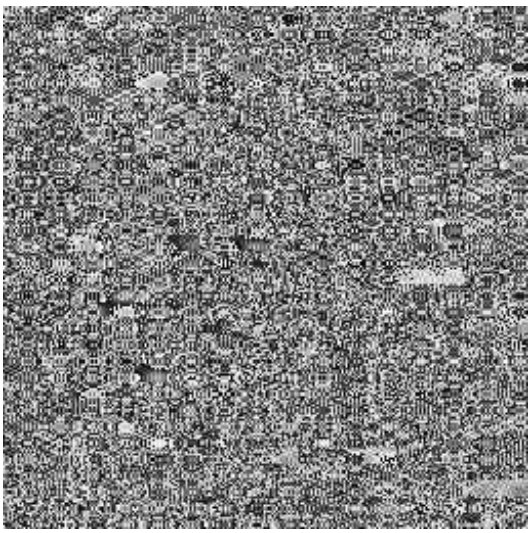
(e) RC-ONMNT Bit Error $N = 64$



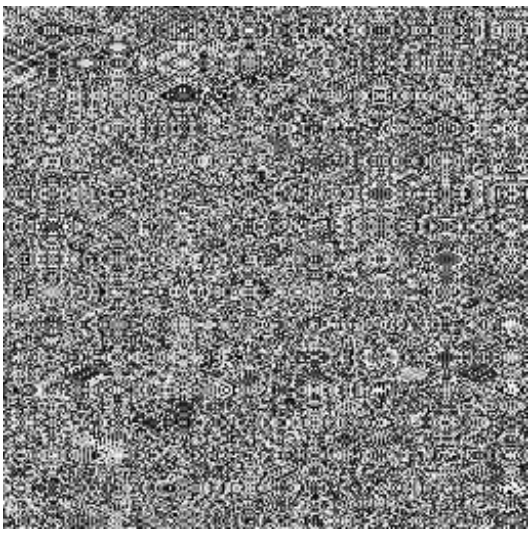
(f) RC-ONMNT Bit Error $N = 128$

Figure 5.22: Decrypted Bit Error using RC-ONMNT with $N = 8, 16, 32, 64, 128$ and $Mp = 131071$

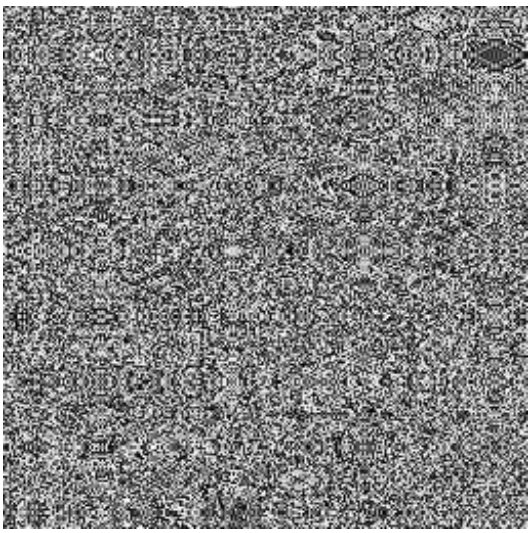
(a) Original Image



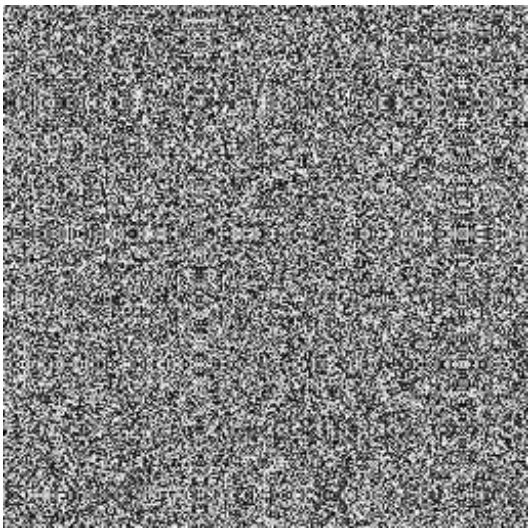
(b) RC-O^{2}NMNT Bit Error $N = 8$



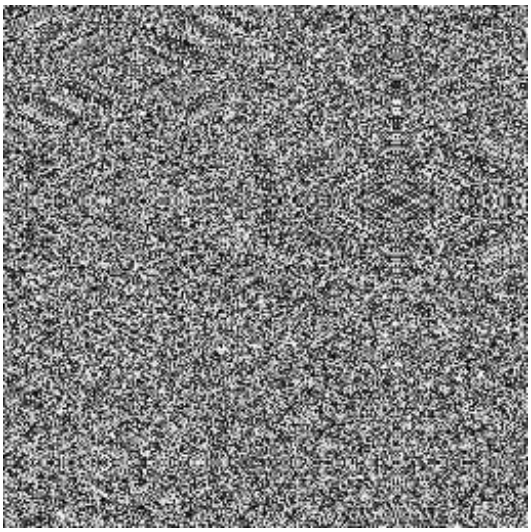
(c) RC-O^{2}NMNT Bit Error $N = 16$



(d) RC-O^{2}NMNT Bit Error $N = 32$



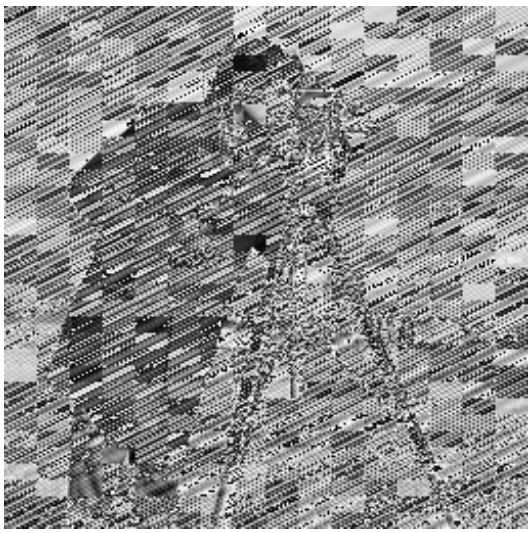
(e) RC-O^{2}NMNT Bit Error $N = 64$



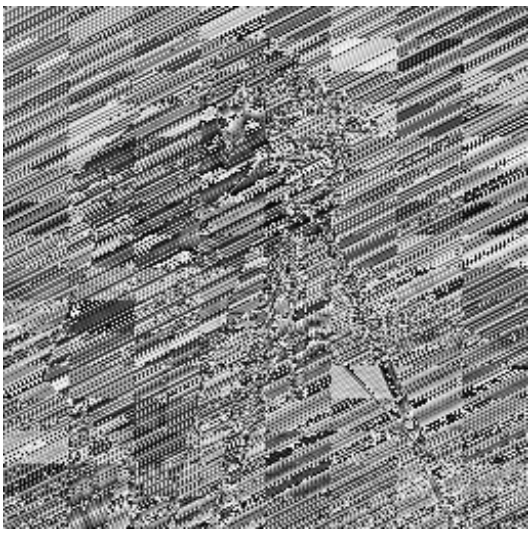
(f) RC-O^{2}NMNT Bit Error $N = 128$

Figure 5.23: Decrypted Bit Error using RC-O^{2}NMNT with $N = 8, 16, 32, 64, 128$ and $Mp = 131071$

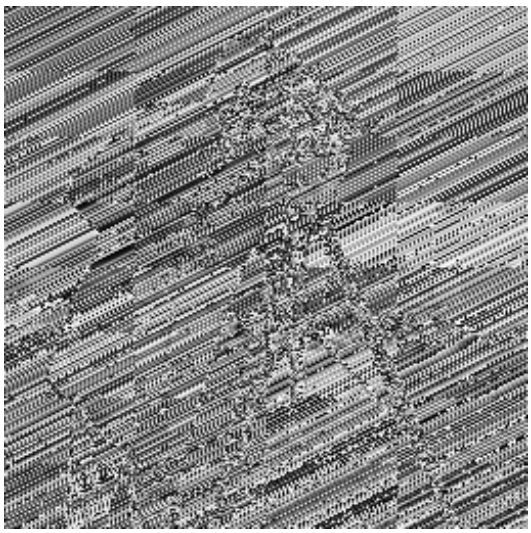
(a) Original Image



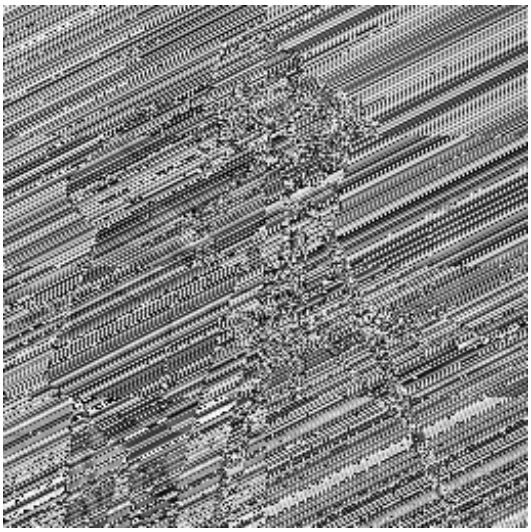
(b) 2D-NMNT Bit Error $N = 8$



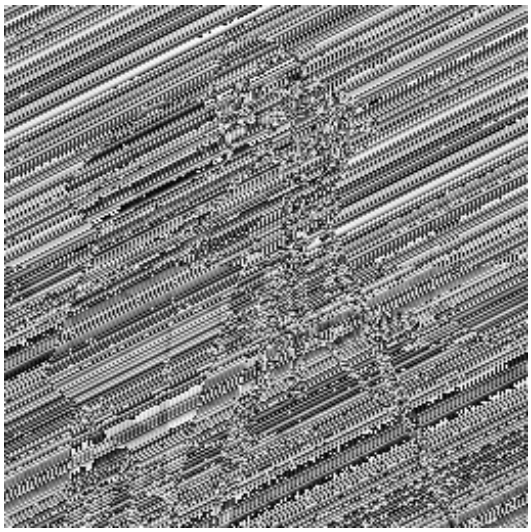
(c) 2D-NMNT Bit Error $N = 16$



(d) 2D-NMNT Bit Error $N = 32$



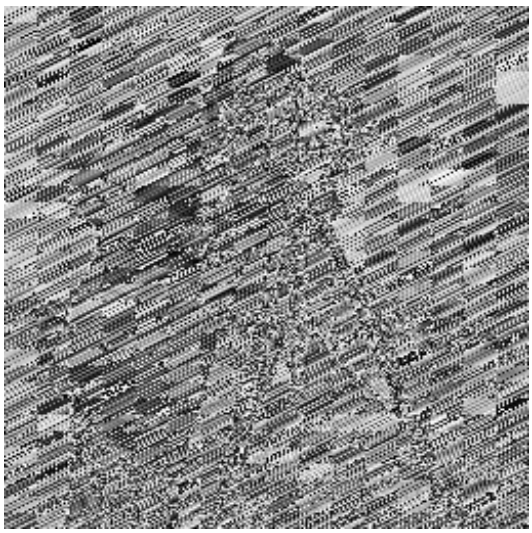
(e) 2D-NMNT Bit Error $N = 64$



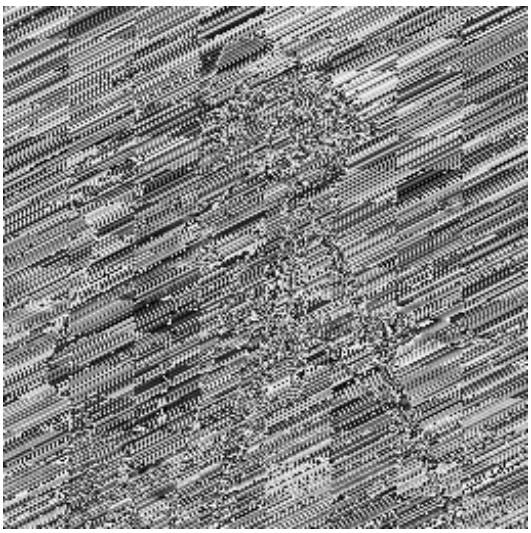
(f) 2D-NMNT Bit Error $N = 128$

Figure 5.24: Decrypted Bit Error using 2D-NMNT with $N = 8, 16, 32, 64, 128$ and $Mp = 131071$

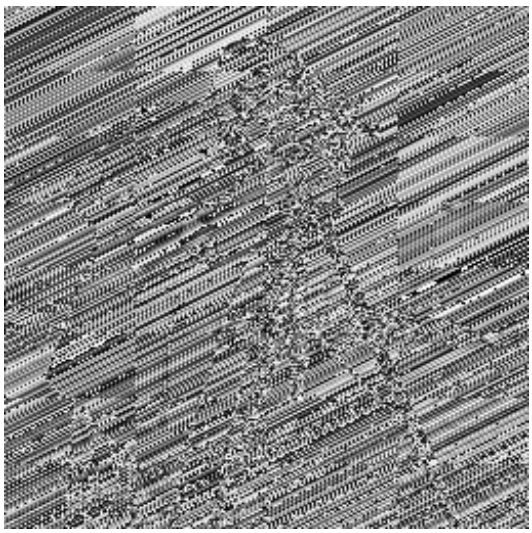
(a) Original Image



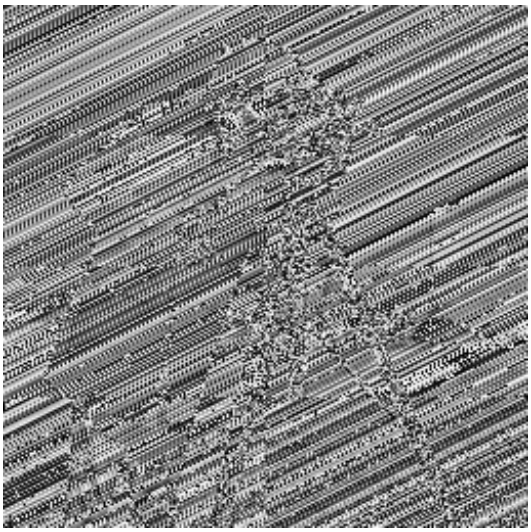
(b) 2D-ONMNT Bit Error $N = 8$



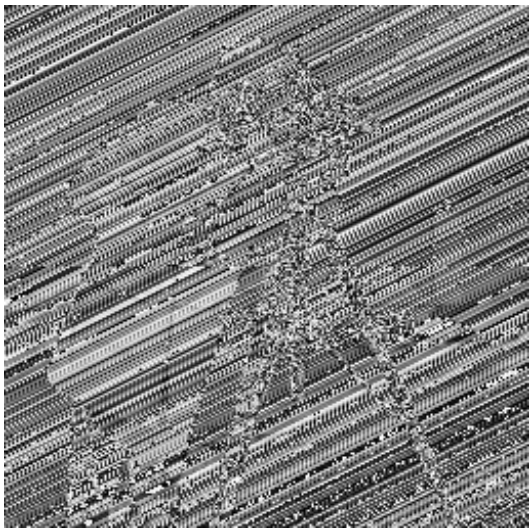
(c) 2D-ONMNT Bit Error $N = 16$



(d) 2D-ONMNT Bit Error $N = 32$



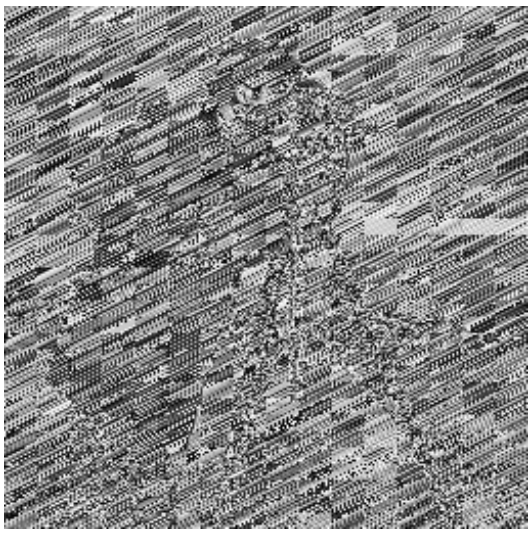
(e) 2D-ONMNT Bit Error $N = 64$



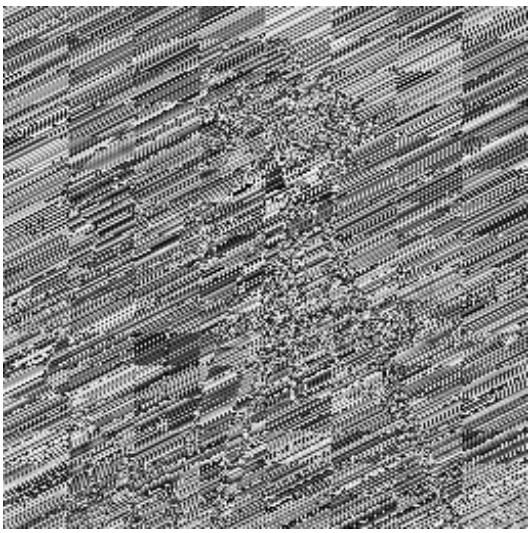
(f) 2D-ONMNT Bit Error $N = 128$

Figure 5.25: Decrypted Bit Error using 2D-ONMNT with $N = 8, 16, 32, 64, 128$ and $Mp = 131071$

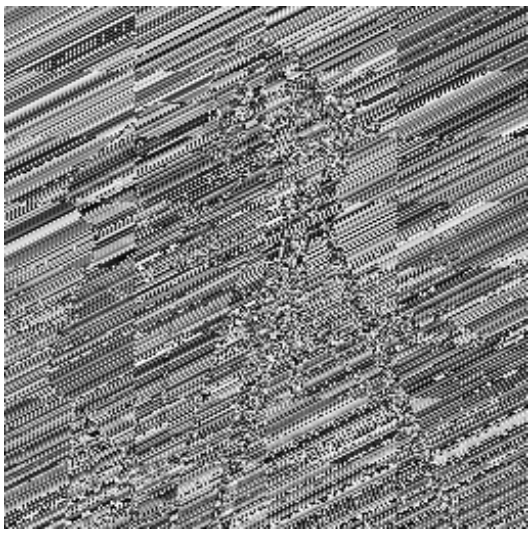
(a) Original Image



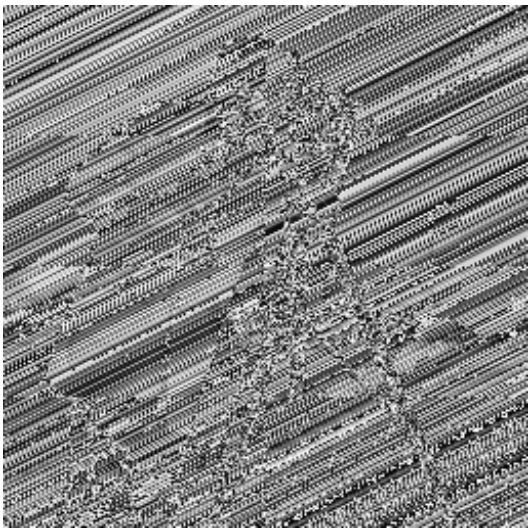
(b) 2D-O^{2}NMNT Bit Error $N = 8$



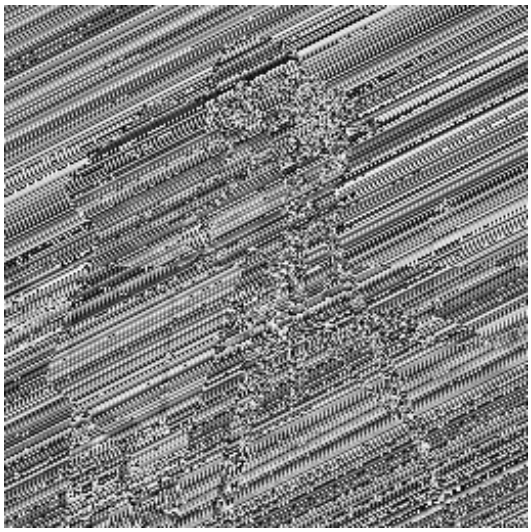
(c) 2D-O^{2}NMNT Bit Error $N = 16$



(d) 2D-O^{2}NMNT Bit Error $N = 32$



(e) 2D-O^{2}NMNT Bit Error $N = 64$



(f) 2D-O^{2}NMNT Bit Error $N = 128$

Figure 5.26: Decrypted Bit Error using 2D-O^{2}NMNT with $N = 8, 16, 32, 64, 128$ and $Mp = 131071$

are similar characteristics that were observed with the NMNT and ONMNT. In doing so, a random key is this time selected and concatenated and again, the last bit of the final element is changed. The results for the bit error RC-GNMNT are shown in Figures 5.21-5.23 where it can be observed that the effects are greatly exacerbated. The reason for this relates to the nature of the processing as the method essentially utilises N keys rather than an $N \times N$ key because of the way the RC-GNMNT is processed. As such, more of the image can still be recovered as clearly observed, although this diminishes as the transform length and subsequent key length increases. Interestingly, the O^2NMNT result in Figure 5.23 appears to be defining the processing blocks rather than producing a random collection of pixels. There should therefore be caution in implementing a technique using the RC-O^2NMNT, ensuring that any schemes employing this transform contain multiple rounds and non-linear operations.

Addressing this technique with the 2D-GNMNT using a random key for encryption with subsequent bit error in the key for decryption produces results that can be observed in Figures 5.24-5.26. There is again evidence within the respective images from each technique that information has been recovered, albeit somewhat skewed, which again diminishes as the transform length and subsequent key length increases. The skewing artefacts are no doubt a result of the separable algorithm being applied to the RC method, which would emphasise how the transform is being processed. As the implementation has been proven already using the convolution techniques, there is no doubt that this skewing condition is confined to the application of the RC and separable algorithm methods. Moreover, the outline of the cameraman can be observed in Figure 5.26, indicating that even the O^2NMNT is not infallible with respect in this instance.

5.8 Conclusion

This chapter has developed and demonstrated the RC-GNMNT and its respective separable algorithms, specifically the ONMNT, IONMNT and O^2NMNT for developing 2D algorithms. Examples using the cyclic convolution have been shown and demonstrated using the Sobel filter to provide edge detection through 2D convolution, thereby verifying its implementation. Example RC and 2D core encryption systems have been developed and presented to demonstrate

the capabilities of these new transforms. Moreover, the simulations of the encryption system show that it is not necessarily required to complete the full 2D transformation in order to achieve favourable results, but maximising the transform length and subsequent key length would certainly be advantageous. However, there is a definite emphasis from the results that further steps are mandated when implementing the GNMNT within an encryption system, particularly shuffling, non-linear operations and multiple rounds. Whilst there were comparable results between the RC and 2D methods and the RC method may serve as a faster implementation, there may well be further benefits in completing the full 2D process as other opportunities are presented that could incorporate additional functionality to an encryption system. For example, part of an encryption system may consist of filtering parts of the data either as a pre- or post-operation to the underlying encryption system.

Chapter 6

The Avalanche Effect of the GNMNT

6.1 Introduction

The continual pace of technological development has accelerated the need for stronger and more resilient encryption schemes. These schemes should offer flexibility with respect to scalability, depending on needs and requirements. The current standard [144] is currently halfway through its second decade of ratification and while it has shown significant resilience so far, that is not to say it will continue to do so. Reflecting how DES has been continually attacked with the production of faster and more powerful GPUs highlighting the need for development of similar systems that can be developed like deep crack, but using only commercial off-the-shelf (COTS) components.

6.2 The Strict Avalanche Criterion

The avalanche effect is primarily influenced by diffusion that was noted by Shannon when he first suggested that the two core concepts of security are confusion and diffusion [12]. The key principle behind the avalanche model is that should a single bit in the cipher text be out of place when applied to the decryption algorithm, then it will yield a catastrophic effect upon the decoded plaintext. This theory was developed further by Feistel and the ideas of confusion and diffusion were more commonly referred to and implemented as networks incorporating substitution and

permutation respectively [13]. Webster and Travers developed the avalanche effect further still by introducing the SAC [14]. The SAC states that when a single bit is changed in the cipher text, the effect of this change should influence the resultant bits after the decryption process ideally by a probability of 0.5. We can therefore measure the resultant plaintext of such an event and compare it with the original plaintext where we should ideally observe that 50% of the bits between the two plaintexts differ.

6.3 Methodology

There were a number of configurations and variations used for the development of this simulation that include the simulation algorithm to apply conditions to test and measure the SAC, the AES algorithm and the GNMNT transforms. The simulations involved 10^6 iterations for each of the GNMNT transforms and the AES algorithm, where either 136 or 128 bits were manipulated respectively during each iteration. The implementation for each process will be described in greater detail in Sections 6.3.2 and 6.3.3 for AES and GNMNT respectively.

6.3.1 Applying and Testing the SAC

The method used to undertake these measurements was designed to probe each and every bit of the cipher text to identify not only consistency but also areas that potentially exhibit undesirable characteristics: these may manifest as results that contain excessively low or high metrics or exhibit consistent patterns. The process used to accomplish this process was previously described in [119], beginning with the obtaining of a cipher text c by applying an encryption operation $\mathcal{E}(\cdot)$ to a plaintext t by

$$c = \mathcal{E}(t). \tag{6.1}$$

We can then change a single bit in c by selecting a bit position n and performing

$$\hat{c}_m = c \oplus 2^n \text{ for } 0 \leq n < pN \tag{6.2}$$

where $\hat{c}_m$ is the indexed state from the original cipher text where the bit that has been modified $n = m$, $\oplus$ denotes a bitwise XOR operation, p is the size of each element and N represents the number of elements in the cipher. Using n to select

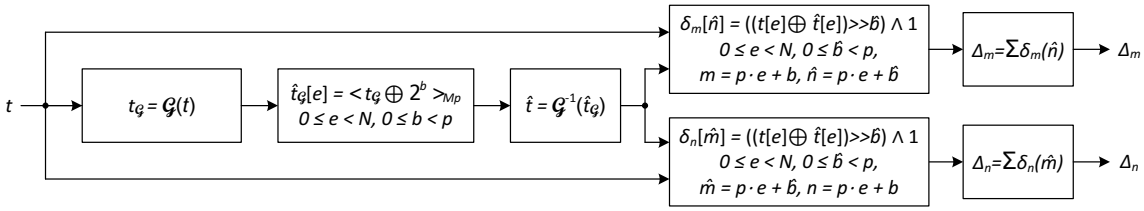


Figure 6.1: Avalanche Assessment Process

a bit and treating the vector as if it were an N size element, it can be better visualised as

$$n = pe + b, \text{ for } 0 \le e < N \text{ and } 0 \le b < p, \tag{6.3}$$

which converts n into an element that is index by e and the associated bit of that element referenced by 2^b. We then decrypt the altered cipher text $\hat{c}_m$ to obtain an alternate plaintext $\hat{t}_m$ by

$$\hat{t}_m = \mathcal{D}(\hat{c}_m) \tag{6.4}$$

where $\mathcal{D}(\cdot)$ refers to the corresponding decryption algorithm from (6.1). The bits that have changed can be easily detected by developing a small function

$$\delta_m(n) = \begin{cases} 1, \text{ if } (t \oplus \hat{t}_m) \wedge 2^n \neq 0 \\ 0 \end{cases} \tag{6.5}$$

using $\wedge$ to denote a bitwise AND operation to process each bit in $\hat{t}_m$. We can then represent the total change in the resultant vector $\hat{t}_m$ by

$$\Delta_m = \left[\frac{1}{pN} \sum_{n=0}^{pN-1} \delta_m(n) \right] \times 100. \tag{6.6}$$

As (6.5) and (6.6) provide a methodology to measure the differential change in the vector, so we can also measure the differential bit positions from the resultant vector by applying a transpose. This will provide

$$\delta_m(n) = \begin{cases} 1, \text{ if } (t \oplus \hat{t}_n) \wedge 2^m \neq 0 \\ 0 \end{cases} \tag{6.7}$$

and

$$\Delta_n = \left[\frac{1}{pN} \sum_{n=0}^{pN-1} \delta_n(m) \right] \times 100. \tag{6.8}$$

This process is further illustrated in Figure 6.1.

We can demonstrate a small exhaustive example using $p = 3$ and $N = 4$ and begin by selecting four numbers at random for t producing

$$t = \{3, 2, 1, 2\}. \tag{6.9}$$

Table 6.1: Modification of $\hat{t}_{\mathcal{G}}$ for $p = 3$, $N = 4$

e	b	$\hat{t}_{\mathcal{G}}[3]$	$\hat{t}_{\mathcal{G}}[2]$	$\hat{t}_{\mathcal{G}}[1]$	$\hat{t}_{\mathcal{G}}[0]$
0	0	001	010	000	011
0	1	001	010	000	000
0	2	001	010	000	110
1	0	001	010	001	010
1	1	001	010	010	010
1	2	001	010	100	010
2	0	001	011	000	010
2	1	001	000	000	010
2	2	001	110	000	010
3	0	000	010	000	010
3	1	011	010	000	010
3	2	101	010	000	010

Table 6.2: Resultant Values of $\hat{t}$ for $p = 3$, $N = 4$

$pe + b$	$\hat{t}[3]$	$\hat{t}[2]$	$\hat{t}[1]$	$\hat{t}[0]$
0	101	100	110	000
1	110	101	101	110
2	100	011	000	001
3	101	000	011	000
4	000	101	101	101
5	100	001	010	001
6	101	000	110	100
7	110	110	101	101
8	100	001	000	011
9	001	000	110	000
10	000	110	101	110
11	100	011	010	011

However, for the sake of clarity in this example we shall instead represent values in binary, which results in

$$t = \{011_2, 010_2, 001_2, 010_2\}. \tag{6.10}$$

Transforming this vector using the NMNT produces the vector

$$T_{\mathcal{G}} = \{001_2, 010_2, 000_2, 010_2\}. \tag{6.11}$$

Now we can produce a loop using e and b, which were defined in (6.3) to change each bit of $T_{\mathcal{G}}$ in sequence resulting in pN new transformed vectors denoted by $\hat{T}_{\mathcal{G}}$. These altered values of $T_{\mathcal{G}}$ are shown in Table 6.1. Applying the inverse NMNT transform to each of the $\hat{T}_{\mathcal{G}}$ and record them in $\hat{t}_m$, where m was defined in (6.4). These resultant values of $\hat{t}$ are shown in Table 6.2. Finally, applying (6.5) and (6.6) using t and each $\hat{t}$ produces values for δ_m and Δ_m respectively. While this method assess the bits that have changed in each vector and can be seen as a row-by-row approach, using the resultant vectors we can also examine each column. Applying the similar (6.7) and (6.8) produces values for δ_n and Δ_n respectively. The results

Table 6.3: Differences Between t and $\hat{t}$ for $p = 3$, $N = 4$

	δn												Δ_m	m
δ_m	1	1	0	1	1	0	1	1	1	0	1	0	8	0
	1	0	1	1	1	1	1	0	0	1	0	0	7	1
	1	1	1	0	0	1	0	0	1	0	1	1	7	2
	1	1	0	0	1	0	0	1	0	0	1	0	5	3
	0	1	1	1	1	1	1	0	0	1	1	1	9	4
	1	1	1	0	1	1	0	1	1	0	1	1	9	5
	1	1	0	0	1	0	1	1	1	1	1	0	8	6
	1	0	1	1	0	0	1	0	0	1	1	1	7	7
	1	1	1	0	1	1	0	0	1	0	0	1	7	8
	0	1	0	0	1	0	1	1	1	0	1	0	6	9
	0	1	1	1	0	0	1	0	0	1	0	0	5	10
	1	1	1	0	0	1	0	1	1	0	0	1	7	11
Δ_n	9	10	8	5	8	6	7	6	7	5	8	6	$\wp$	$\sum \Delta_n$
n	11	10	9	8	7	6	5	4	3	2	1	0	$\sum \Delta_m$	$\sum \Delta_m = \sum \Delta_n$

of the XOR operation between t and each $\hat{t}$ that produces the respective δ and Δ values that can be seen in Table 6.3. This table also shows the sums of the two sequences of Δ, which naturally should be equal. Calculating the mean $\bar{x}$ can be achieved by using either Δ_m or Δ_n for $0 \leq m, n < N$ as both yield the same value. In fact, this is a measure to verify that the results have been both recorded and processed correctly. Measuring the standard deviation is again a straight forward process as

$$\sigma = \sqrt{\frac{1}{m} \sum_{m=0}^{N-1} (\Delta_m - \bar{x})^2}. \tag{6.12}$$

As the data has been processed to provide a percentage for the results, the standard deviation is also expressed as a percentage. To reflect the impact this has on the actual block, a small adjustment must be made to change from a percentage to the relevant number of bits. We denote this new measurement as σ_τ and calculate it thus:

$$\sigma_\tau = (pN\sigma) \times \frac{1}{100}. \tag{6.13}$$

6.3.2 AES Implementation

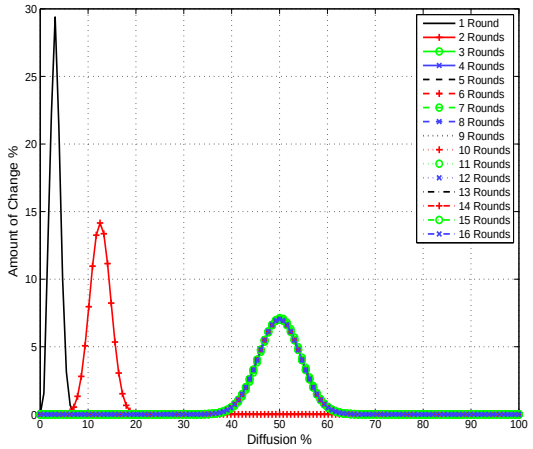
The AES implementation uses a slightly stripped-down version of the algorithm that has the password functionality removed so as to better compare the algorithm with the GNMNT where any influence of the passwords is removed; this is essential for measuring the raw diffusion characteristics. To fully exploit the main implementation characteristics of AES, which was designed to run on eight-bit processors, all of the functions have been replaced with lookup tables. This has a minimal impact in terms of storage as only eight tables of 256 bytes each are required to accomplish this implementation: SubBytes, inverse SubBytes, $2n$, $3n$, $9n$, $11n$, $13n$ and $14n$. This results in a total of 2,048 bytes of storage in total with only 16 lookups / round for the SubBytes function and 64 lookups with 48 XOR operations that combine the ShiftRows and MixColumns functions for each round. Finally, when simulations were undertaken using the AES algorithm, they ran from between one and 16 rounds, the total number of elements in the cipher block. This was so that as well as the common 10, 12 and 14 rounds that are usually applied, further configurations could be applied and analysed.

6.3.3 GNMNT Implementation

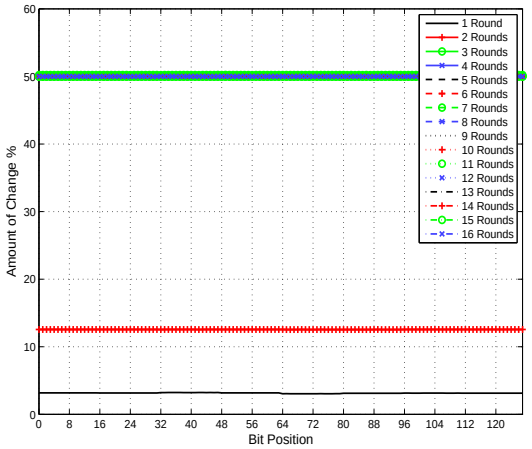
The configuration for the GNMNT simulations was selected as $p = 17$ and $N = 8$, which provided a block size of 136 bits that was as close as possible to the 128-bit block size of AES. The same vector that was derived for the AES was used but each element contained two elements from AES that were concatenated. Assessments using the GNMNT implementation were initially accomplished using the relevant straightforward transforms with no further operations applied. These simulations ran from one to a maximum of eight rounds, the number of elements in the GNMNT cipher block. In the second set of simulations, the elements were rotated one place in between each round during the forward encryption phase. During the inverse decryption phase, the elements were rotated in the opposite direction in between each round. This rationale for doing this will be explained further in Section 6.5 where its relevance can be better appreciated.

6.4 Assessing the Default Configuration

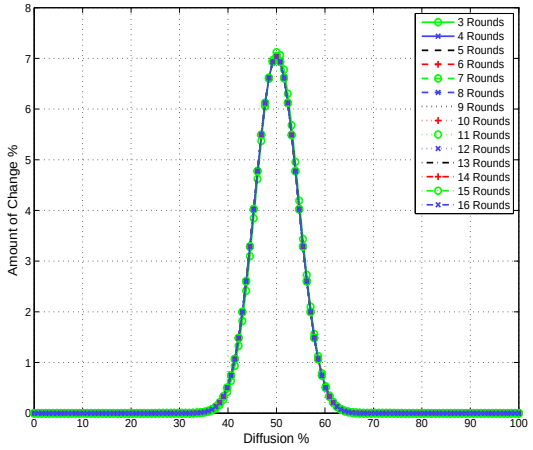
The results from each simulation are shown both graphically and statistically. Whilst it is more common to disseminate such results statistically, the graphical representation provides significantly more information that otherwise would be missing in the statistics. The metrics obtained within the statistics include the minimum, maximum and the range of bits that were affected expressed as a percentage, the standard deviation σ that was observed to serve as a tolerance, also as a percentage and the standard deviation σ_τ that was expressed as the affected number of bits. The average number of bits affected was also the same which, as expected, was consistent whether observing the diffusion over the vector or the bit position. The default results demonstrate each process with no additional interference.



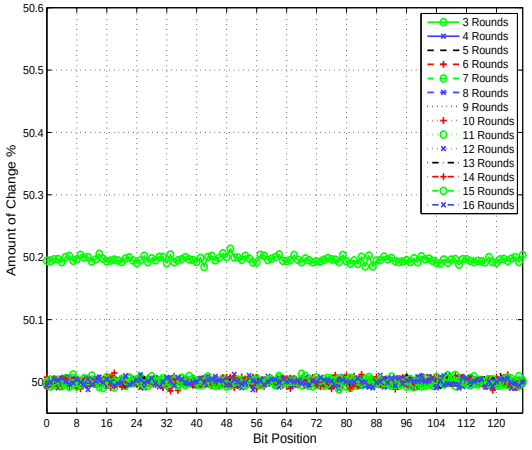
(a) Vector Analysis: Rounds 1-16



(b) Bit Position Analysis: Rounds 1-16



(c) Vector Analysis: Rounds 3-16

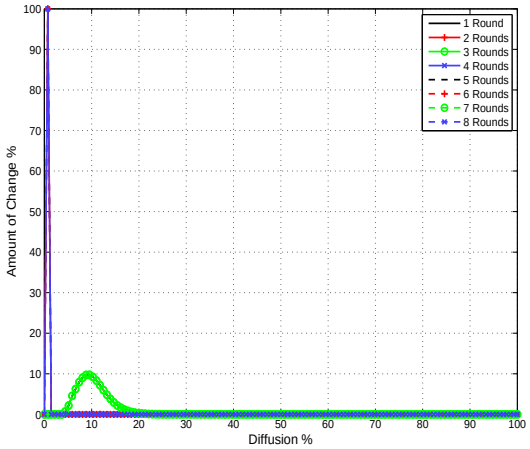


(d) Bit Position Analysis: Rounds 3-16

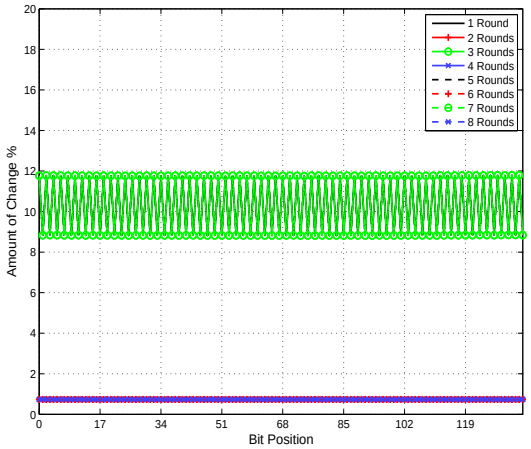
Figure 6.2: AES Analysis

Table 6.4: AES Metrics - Vector / Bit Position

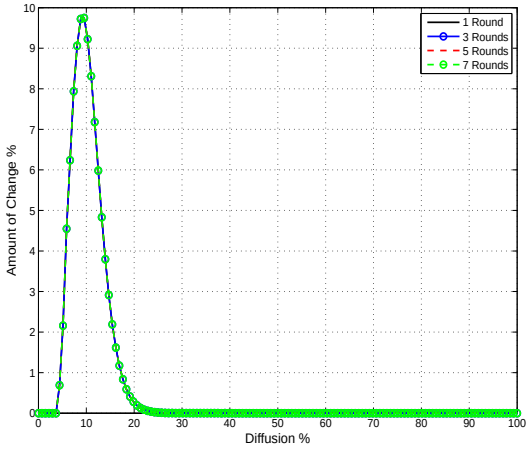
Rounds	Δ_m%				Δ_m bits	Δ_{mn}%	Δ_n bits	Δ_n%			
	Min	Max	Range	σ	σ_τ	$\bar{x}$	σ_τ	σ	Range	Max	Min
1	0.7813	6.2500	5.4688	0.1252	0.1600	3.1447	0.0030	0.0022	0.1745	3.2239	3.0495
2	3.1250	23.4375	20.3125	0.3406	0.4360	12.5521	0.0110	0.0086	0.0219	12.5613	12.5394
3	25.7813	74.2188	48.4375	0.9190	1.1760	50.1963	0.0440	0.0345	0.0293	50.2137	50.1844
4	25.0000	73.4375	48.4375	0.9085	1.1630	49.9995	0.0440	0.0344	0.0216	50.0107	49.9891
5	25.0000	73.4375	48.4375	0.9085	1.1630	49.9996	0.0440	0.0344	0.0248	50.0143	49.9895
6	25.0000	75.0000	50.0000	0.9085	1.1630	50.0002	0.0440	0.0344	0.0237	50.0095	49.9858
7	25.0000	75.7813	50.7813	0.9085	1.1630	50.0005	0.0440	0.0344	0.0240	50.0120	49.9879
8	25.0000	73.4375	48.4375	0.9086	1.1630	50.0006	0.0440	0.0344	0.0244	50.0123	49.9880
9	25.7813	75.0000	49.2188	0.9086	1.1630	49.9998	0.0440	0.0344	0.0206	50.0107	49.9901
10	26.5625	74.2188	47.6563	0.9086	1.1630	50.0004	0.0440	0.0344	0.0254	50.0119	49.9865
11	25.7813	73.4375	47.6563	0.9085	1.1630	50.0002	0.0440	0.0344	0.0197	50.0101	49.9904
12	25.0000	76.5625	51.5625	0.9085	1.1630	49.9996	0.0440	0.0344	0.0242	50.0117	49.9875
13	25.0000	75.0000	50.0000	0.9086	1.1630	49.9999	0.0440	0.0344	0.0226	50.0119	49.9893
14	25.7813	75.0000	49.2188	0.9085	1.1630	49.9996	0.0440	0.0344	0.0274	50.0146	49.9871
15	26.5625	74.2188	47.6563	0.9085	1.1630	49.9999	0.0440	0.0344	0.0208	50.0133	49.9926
16	24.2188	75.0000	50.7813	0.9085	1.1630	50.0006	0.0440	0.0344	0.0196	50.0103	49.9907



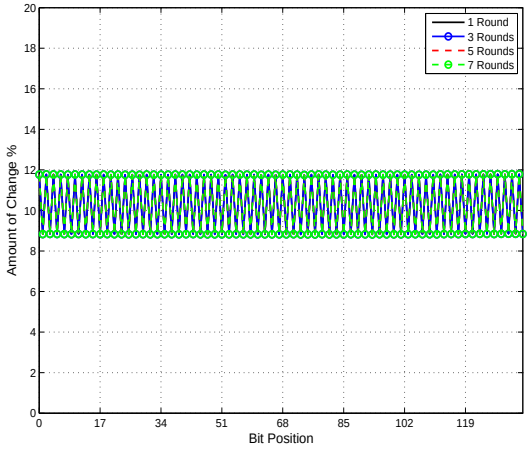
(a) Vector Analysis: Rounds 1-8



(b) Bit Position Analysis: Rounds 1-8



(c) Vector Analysis: Rounds 1,3,5 and 7



(d) Bit Position Analysis: Rounds 1,3,5 and 7

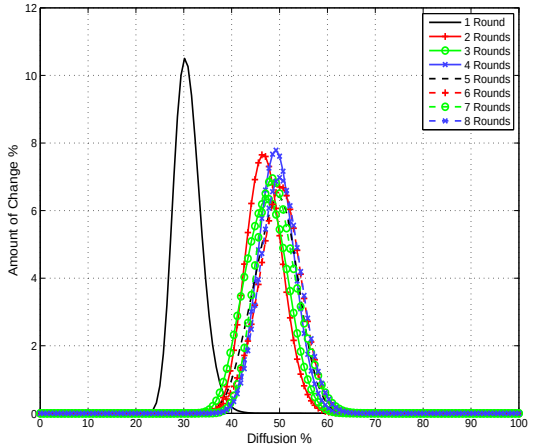
Figure 6.3: NMNT Analysis

Table 6.5: NMNT Metrics - Vector / Bit Position

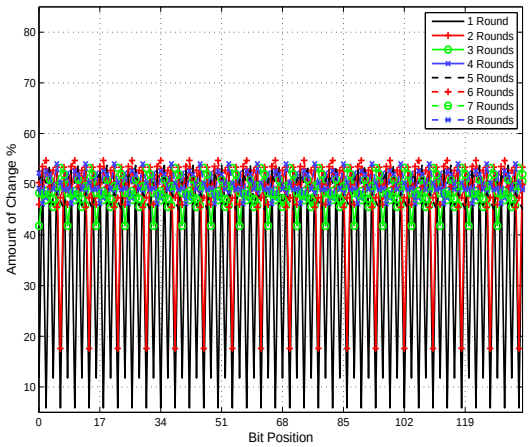
Rounds	$\Delta_m\%$				Δ_m bits	$\Delta_{mn}\%$	Δ_n bits	$\Delta_n\%$			
	Min	Max	Range	σ	σ_τ	$\bar{x}$	σ_τ	σ	Range	Max	Min
1	4.4118	38.9706	34.5588	0.2121	0.2880	10.2982	0.0170	0.0126	2.9737	11.7906	8.8169
2	0.7353	11.7647	11.0294	0.0628	0.0850	0.7353	0.0010	0.0005	0.0000	0.7353	0.7353
3	4.4118	38.9706	34.5588	0.2121	0.2880	10.2982	0.0170	0.0126	2.9737	11.7906	8.8169
4	0.7353	11.7647	11.0294	0.0628	0.0850	0.7353	0.0010	0.0005	0.0000	0.7353	0.7353
5	4.4118	38.9706	34.5588	0.2121	0.2880	10.2982	0.0170	0.0126	2.9737	11.7906	8.8169
6	0.7353	11.7647	11.0294	0.0628	0.0850	0.7353	0.0010	0.0005	0.0000	0.7353	0.7353
7	4.4118	38.9706	34.5588	0.2121	0.2880	10.2982	0.0170	0.0126	2.9737	11.7906	8.8169
8	0.7353	11.7647	11.0294	0.0628	0.0850	0.7353	0.0010	0.0005	0.0000	0.7353	0.7353

6.4.1 Assessing the AES

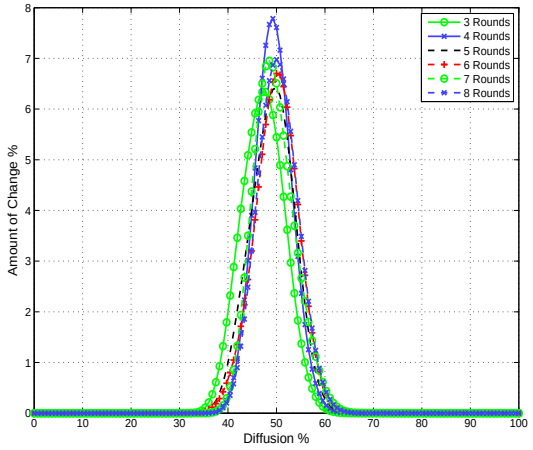
The results for the AES up to 16 rounds is shown graphically in Figures 6.2(a) and 6.2(b), which depict the diffusion rate that was obtained using the default implementation. It can be observed that it took three rounds for AES to settle to a desirable rate and four rounds to consistently reach the ideal rate. The statistics



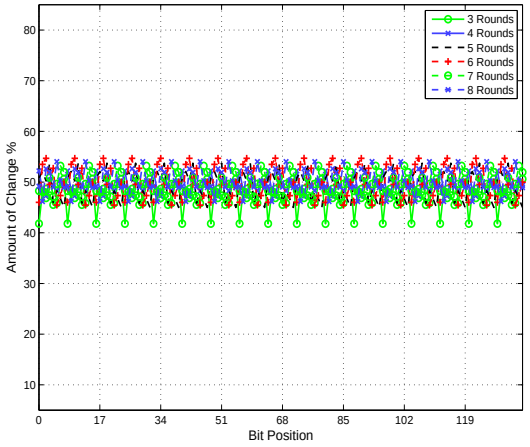
(a) Vector Analysis: Rounds 1-8



(b) Bit Position Analysis: Rounds 1-8



(c) Vector Analysis: Rounds 3-8



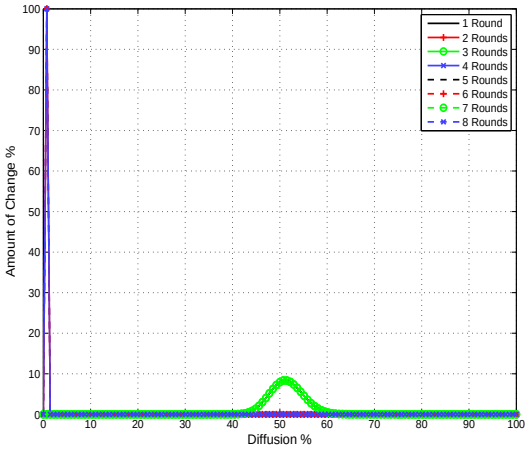
(d) Bit Position Analysis: Rounds 3-8

Figure 6.4: ONMNT Analysis

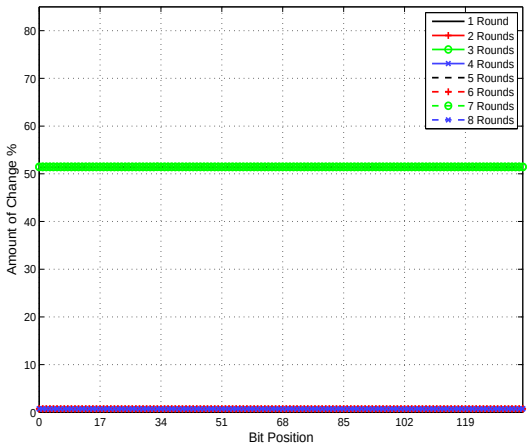
Table 6.6: ONMNT Metrics - Vector / Bit Position

Rounds	Δ_m%				Δ_m bits	Δ_{mn}%	Δ_n bits	Δ_n%			
	Min	Max	Range	σ	σ_τ	$\bar{x}$	σ_τ	σ	Range	Max	Min
1	21.3235	54.4118	33.0882	0.6781	0.9220	30.9032	0.2230	0.1642	47.1186	52.9952	5.8766
2	31.6176	72.7941	41.1765	0.8713	1.1850	46.9111	0.1190	0.0877	35.7798	53.3721	17.5924
3	27.9412	72.7941	44.8529	0.7971	1.0840	47.2102	0.0480	0.0354	8.8330	50.6216	41.7886
4	32.3529	72.0588	39.7059	0.9238	1.2560	49.3714	0.0440	0.0327	4.9214	52.2194	47.2980
5	29.4118	75.0000	45.5882	0.8292	1.1280	49.0878	0.0510	0.0376	8.7430	53.7353	44.9923
6	29.4118	73.5294	44.1176	0.8571	1.1660	49.9016	0.0540	0.0395	9.2530	54.7078	45.4548
7	31.6176	75.0000	43.3824	0.8556	1.1640	49.5020	0.0480	0.0349	6.3323	53.2010	46.8686
8	32.3529	76.4706	44.1176	0.8882	1.2080	50.2657	0.0490	0.0360	7.8730	54.0375	46.1644

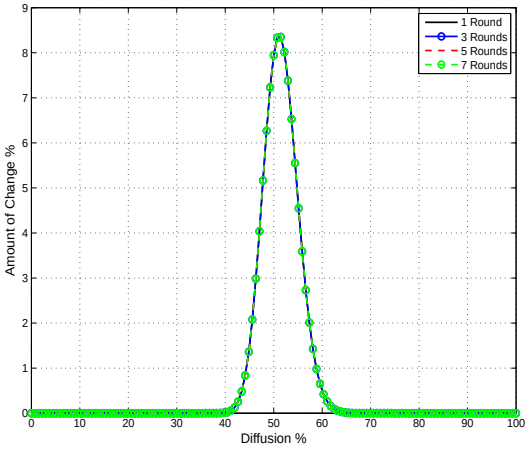
obtained for this simulation are shown in Table 6.4, which shows the total bits affected within the vectors tested on the left-hand side of the table and the bit positions affected within the vectors tested on the right-hand side. However, the statistics suggest that the best scenario would be at least four rounds to ensure consistency. It can be observed in Figure 6.2(b) that this would be the ideal starting point, where the remaining rounds consistently yield results that are both



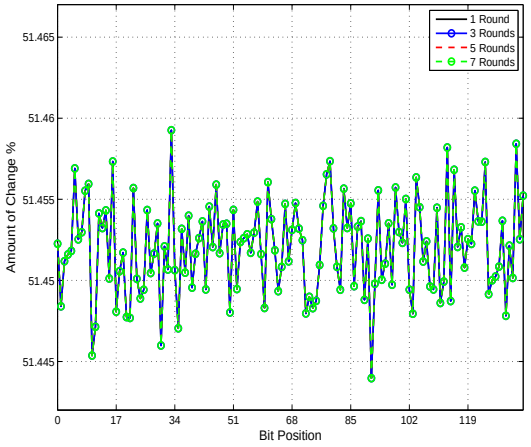
(a) Vector Analysis: Rounds 1-8



(b) Bit Position Analysis: Rounds 1-8



(c) Vector Analysis: Rounds 1,3,5 and 7



(d) Bit Position Analysis: Rounds 1,3,5 and 7

Figure 6.5: O^{2}NMNT Analysis

Table 6.7: O^{2}NMNT Metrics - Vector / Bit Position

Rounds	Δ_m%				Δ_m bits	Δ_{mn}%	Δ_n bits	Δ_n%			
	Min	Max	Range	σ	σ_τ	$\bar{x}$	σ_τ	σ	Range	Max	Min
1	36.0294	76.4706	40.4412	1.0047	1.3660	51.4521	0.0440	0.0323	0.0153	51.4593	51.4440
2	0.7353	11.7647	11.0294	0.0628	0.0850	0.7353	0.0010	0.0005	0.0000	0.7353	0.7353
3	36.0294	76.4706	40.4412	1.0047	1.3660	51.4521	0.0440	0.0323	0.0153	51.4593	51.4440
4	0.7353	11.7647	11.0294	0.0628	0.0850	0.7353	0.0010	0.0005	0.0000	0.7353	0.7353
5	36.0294	76.4706	40.4412	1.0047	1.3660	51.4521	0.0440	0.0323	0.0153	51.4593	51.4440
6	0.7353	11.7647	11.0294	0.0628	0.0850	0.7353	0.0010	0.0005	0.0000	0.7353	0.7353
7	36.0294	76.4706	40.4412	1.0047	1.3660	51.4521	0.0440	0.0323	0.0153	51.4593	51.4440
8	0.7353	11.7647	11.0294	0.0628	0.0850	0.7353	0.0010	0.0005	0.0000	0.7353	0.7353

close to one another and at the ideal 50% threshold.

6.4.2 Assessing the NMNT

Figures 6.3(a) and 6.3(b) show the results of the NMNT, which unfortunately have yielded very unfavourable results; In fact findings suggest that in some cases there was virtually no change. The statistics in Table 6.5 show that every even

round that has been processed has resulted in an extremely low performance. A quick calculation on the mean to convert the value from a percentage to bit count consistently produces a figure of 2.0425 bits on average that have been changed. Observation of the NMNT processed over the odd rounds shows only a marginal improvement, with a consistent average of 10.3

6.4.3 Assessing the ONMNT

An initial observation of Figures 6.4(a) and 6.4(b) shows that the ONMNT has produced very promising results. Although Figure 6.4(a) shows that the first round is relatively weaker than the rest and Figure 6.4(b) shows that in fact the first two rounds are potentially the weakest. Removing the first two rounds from these results produces Figures 6.4(c) and 6.4(d) where the is a marked and significant improvement. Table 6.6 shows statistically that after overcoming the first two rounds the transform produces consistently desirable results as can be seen in the range variation across the bit positions. While it is not as good as the AES algorithm, it is a significant baseline from which to develop.

6.4.4 Assessing the O^2NMNT

On first impressions, the results that are provided by the O^2NMNT transform appear to be ambiguous. While there is a significant improvement in the diffusion of both the vector total bits and over the bit positions that reach the ideal threshold, the result is similar to the NMNT where the odd rounds are similar to each other. Similarly, the even rounds are also similar to each other and, like the NMNT, exhibit extremely undesirable results. This is confirmed statistically, where all the odd and even rounds yield identical results. Regardless of the negative aspects however, we can see both statistically in Table 6.6 and graphically in Figures 6.5(c) and 6.5(d) that from the very first round, not only has the objective almost been achieved, but in some areas it yields results that exceed the AES in terms of the amount of change at peak, the difference in bit range and the reflective smaller bit tolerance. Moreover, these positive results are reflected across the bit positions, indicating that the transform produces a strong diffusion factor in respect to the total number of bits affected and shows a consistent metric across all bit positions. Aside from the initial weakness, the O^2NMNT shows very desirable characteristics

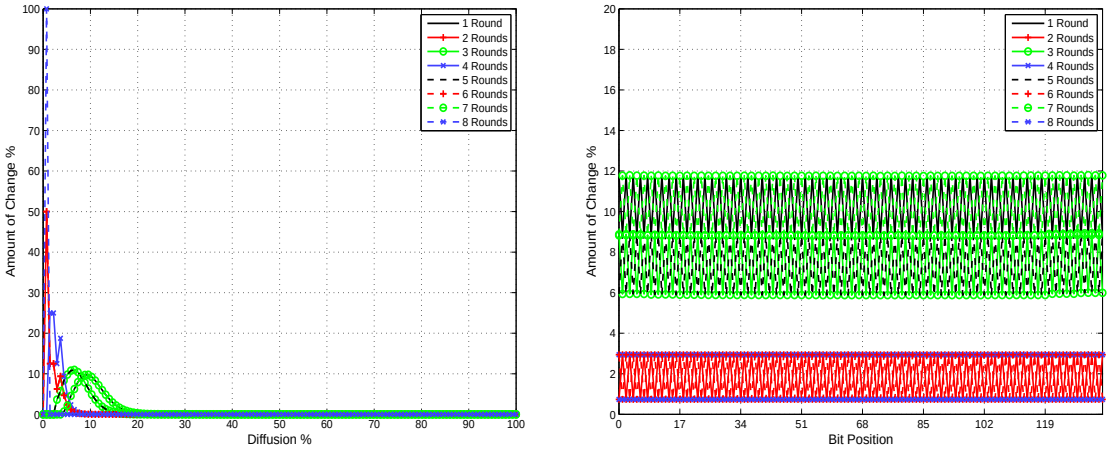
in its very first round.

6.5 Assessing the Modified Configuration

This section addresses the configurations that were first introduced in the methodology and builds upon the default results. The AES shows, from four rounds and beyond, an unfaltering convergence to the desirable characteristics to confirm its resilience and integrity in the measurement of this metric. However, while the ShiftRows and MixColumns functions are very linear in construction, it would make an interesting study to ascertain the amount of influence that the SubBytes function has with respect to diffusion. This is particularly interesting as Rijndaels S-Box, which is essentially what the SubBytes function is, can be reconfigured in many different ways during use; a new irreducible polynomial can be used as can a different affine transform. Addressing the result of the GNMNT suggests that an additional operation should be introduced that will have a minimal impact upon the transforms, yet offer enough influence to address the poor performance that was observed in successive rounds. The first round of the O^2NMNT was particularly favourable, which suggests that the modification should therefore be made as an additional process after the first round. As introduced in the methodology, this additional process is a function that rotates the elements by one place in between rounds of the forward transform and reverses the direction during the inverse transform over successive rounds. Whilst there are myriad ways that these elements can be rotated, shuffled, flipped or mirrored, it would be appropriate for the change to have as little impact to the structure as possible so as not to introduce undetectable weaknesses that would be missed through overly complex processes.

6.5.1 Assessing the NMNT (Shuffled)

The results shown in the newly modified NMNT are marginally better at best. While the original default transform alternated the results according to whether or not the current round was an even or odd iteration, the modified version now repeats every converging round. For example it can be observed in Table 6.8 that the first round is almost identical to the seventh round, the second round is almost



(a) Vector Analysis: Rounds 1-8

(b) Bit Position Analysis: Rounds 1-8

Figure 6.6: NMNT Analysis (Shuffled)

Table 6.8: NMNT Metrics - Vector / Bit Position (Shuffled)

Rounds	Δ_m%				Δ_m bits	Δ_{mn}%	Δ_n bits	Δ_n%			
	Min	Max	Range	σ	σ_τ	$\bar{x}$	σ_τ	σ	Range	Max	Min
1	4.4118	38.9706	34.5588	0.2121	0.2880	10.2982	0.0170	0.0126	2.9737	11.7906	8.8169
2	0.7353	13.9706	13.2353	0.0567	0.0770	1.8382	0.0110	0.0082	2.2089	2.9442	0.7353
3	2.9412	32.3529	29.4118	0.1627	0.2210	7.3698	0.0160	0.0118	3.0593	8.9365	5.8772
4	1.4706	13.9706	12.5000	0.0956	0.1300	2.9411	0.0030	0.0018	0.0069	2.9442	2.9373
5	2.9412	33.8235	30.8824	0.1627	0.2210	7.3696	0.0160	0.0118	3.0676	8.9425	5.8750
6	0.7353	13.9706	13.2353	0.0567	0.0770	1.8383	0.0110	0.0082	2.2086	2.9438	0.7353
7	4.4118	38.9706	34.5588	0.2121	0.2880	10.2981	0.0170	0.0126	2.9842	11.7976	8.8134
8	0.7353	11.7647	11.0294	0.0628	0.0850	0.7353	0.0010	0.0005	0.0000	0.7353	0.7353

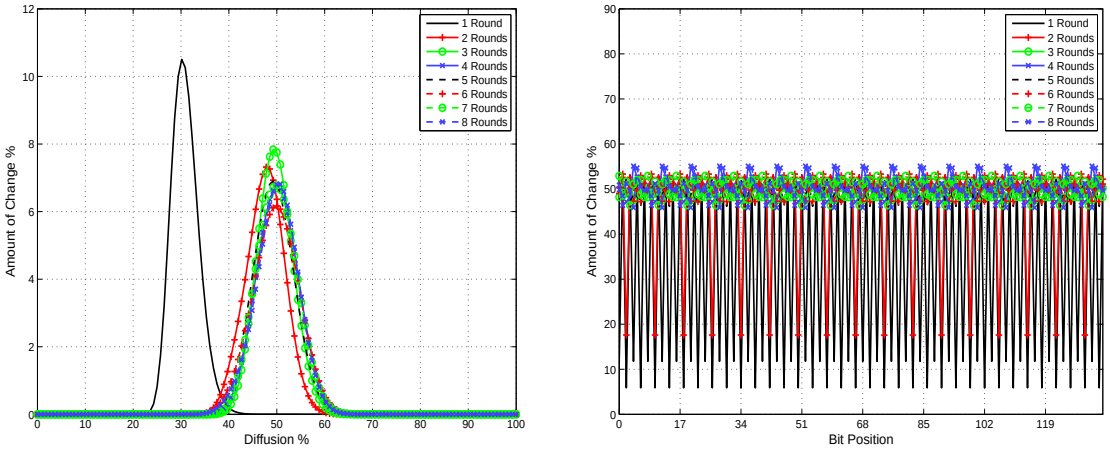
identical to the sixth round, the third with the fifth and the fourth and eight rounds show certain attributes that are either multiples of two or four in some cases.

6.5.2 Assessing the ONMNT (Shuffled)

Modifying the ONMNT does not appear to have brought any noticeable benefits. It doesnt appear to have brought any noticeable degradation either as the transform still demonstrates that a minimum of three rounds are required before the transform starts producing optimal results. In this respect, adding the shuffle function has had an overall neutral effect.

6.5.3 Assessing the O^{2}NMNT (Shuffled)

The O^{2}NMNT has probably shown the greatest benefits of applying such a simple routine. Figure 6.8(a) clearly shows the benefits that have been gained, where the characteristic that was first observed with the NMNT has now been eradicated.



(a) Vector Analysis: Rounds 1-8

(b) Bit Position Analysis: Rounds 1-8

Figure 6.7: ONMNT Analysis (Shuffled)

Table 6.9: ONMNT Metrics - Vector / Bit Position (Shuffled)

Rounds	Δ_m%				Δ_m bits	Δ_{mn}%	Δ_n bits	Δ_n%			
	Min	Max	Range	σ	σ_τ	$\bar{x}$	σ_τ	σ	Range	Max	Min
1	21.3235	54.4118	33.0882	0.6781	0.9220	30.9032	0.2230	0.1642	47.1186	52.9952	5.8766
2	30.8824	72.0588	41.1765	0.8599	1.1690	47.5942	0.1210	0.0888	35.4079	52.9872	17.5793
3	32.3529	73.5294	41.1765	0.9342	1.2710	49.6727	0.0450	0.0330	4.4809	52.0231	47.5422
4	31.6176	75.0000	43.3824	0.8695	1.1830	50.1399	0.0510	0.0373	8.7159	55.0933	46.3773
5	30.8824	72.7941	41.9118	0.8649	1.1760	49.3147	0.0460	0.0341	6.1181	52.2781	46.1601
6	29.4118	73.5294	44.1176	0.8182	1.1130	49.7799	0.0470	0.0344	6.1096	53.3335	47.2238
7	32.3529	74.2647	41.9118	0.8741	1.1890	49.9703	0.0480	0.0350	6.5114	52.9310	46.4196
8	29.4118	73.5294	44.1176	0.8664	1.1780	49.9853	0.0490	0.0357	8.8204	54.6907	45.8702

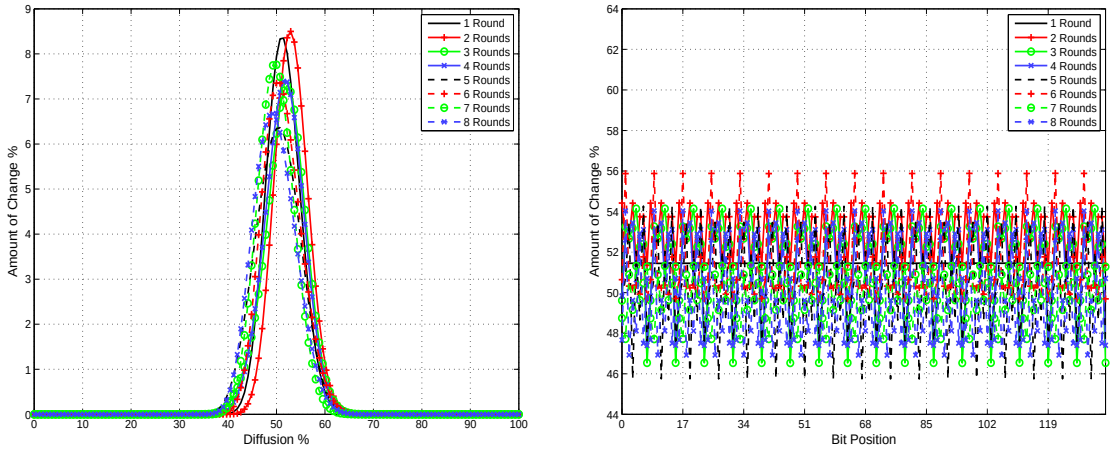
However, there is still room for improvement with respect to diffusion across the bit positions that are shown by their fairly heavy fluctuations as in Figure 6.8(b). The statistics in Table 6.10 confirm that the first round still offers the best performance of the GNMNT with noticeable improvement over the remaining seven rounds.

6.5.4 Exhaustive Analysis of the GNMNT

This section covers the scenario where every bit in a 40-bit space GNMNT configuration using $p = 5$ and $N = 8$ is permutated and manipulated as shown previously in (6.1)-(6.13). The vector space that this configuration spans

$$\begin{aligned}(2^p - 1)^N &= \left(2^5 - 1\right)^8 \\ &= 852,891,037,441 \text{ permutations.}\end{aligned} \tag{6.14}$$

In order to undertake an exhaustive analysis, we must also take $pN = 40$ bits into consideration as well as the three forward $\#\mathcal{G}$ and three inverse $\#\mathcal{G}^{-1}$ operations,



(a) Vector Analysis: Rounds 1-8

(b) Bit Position Analysis: Rounds 1-8

Figure 6.8: O^{2}NMNT Analysis (Shuffled)

Table 6.10: O^{2}NMNT Metrics - Vector / Bit Position (Shuffled)

Rounds	Δ_m%				Δ_m bits	Δ_{mn}%	Δ_n bits	Δ_n%			
	Min	Max	Range	σ	σ_τ	$\bar{x}$	σ_τ	σ	Range	Max	Min
1	36.0294	76.4706	40.4412	1.0047	1.3660	51.4521	0.0440	0.0323	0.0153	51.4593	51.4440
2	36.7647	75.0000	38.2353	1.0448	1.4210	53.0417	0.0470	0.0344	3.0756	54.4265	51.3509
3	31.6176	75.0000	43.3824	0.9280	1.2620	51.7577	0.0500	0.0367	7.6145	54.1491	46.5346
4	33.0882	73.5294	40.4412	0.9400	1.2780	51.6110	0.0490	0.0357	6.0566	53.4523	47.3958
5	30.8824	73.5294	42.6471	0.8507	1.1570	50.2736	0.0510	0.0374	8.5247	54.2704	45.7457
6	33.8235	73.5294	39.7059	0.9242	1.2570	51.1050	0.0480	0.0350	6.1917	55.8871	49.6954
7	33.0882	74.2647	41.1765	0.9354	1.2720	49.8996	0.0440	0.0326	3.6158	51.3236	47.7077
8	32.3529	74.2647	41.9118	0.8571	1.1660	49.6516	0.0490	0.0360	7.1228	54.0376	46.9148

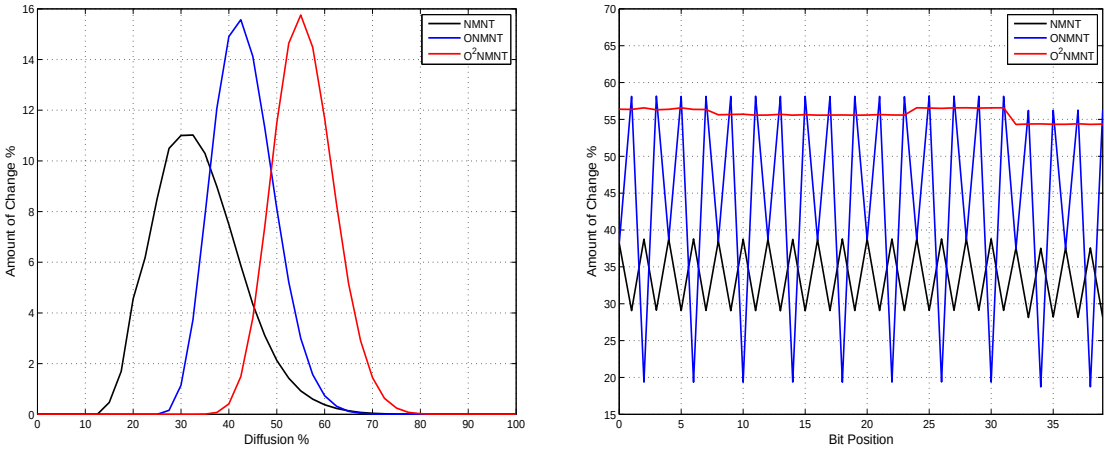
resulting in

$$\#\mathcal{G}\left\{pN\left[(2^p-1)^N\right]\right\} + \#\mathcal{G}^{-1}\left\{pN\left[(2^p-1)^N\right]\right\} = 204,693,848,985,840 \text{ operations.} \tag{6.15}$$

However, we can mitigate almost half of these operations by reusing the forward part of the algorithm so that we then have

$$\#\mathcal{G}\left[(2^p-1)^N\right\} + \#\mathcal{G}^{-1}\left\{pN\left[(2^p-1)^N\right]\right\} = 104,905,597,605,243 \text{ operations.} \tag{6.16}$$

Unfortunately, developing a test application in C running on a single core of a fairly recent 3 GHz CPU [145] yields a peak processing performance of 272,586,000 vectors / second. While this appears to be a significant processing metric, when put into context with the amount of work to be completed, it then means that the analysis will be completed in approximately 40,343 hours or slightly less than 4.61 years. The performance of this particular CPU is rated at 22.17 GFLOPs and consumes 65 Watts [146]. Over the projected time period, this CPU would likely



(a) Vector Analysis (b) Bit Position Analysis

Figure 6.9: GNMNT Exhuastive Analysis: $p = 5, N = 8$

Table 6.11: GNMNT Exhuastive Analysis: $p = 5, N = 8$

$\mathcal{G}$	$\Delta_m\%$				Δ_m bits	$\Delta_{mn}\%$	Δ_n bits	$\Delta_n\%$			
	Min	Max	Range	σ	σ_τ	$\bar{x}$	σ_τ	σ	Range	Max	Min
NMNT	15.0000	97.5000	82.5000	1.2298	0.4920	33.8703	0.0720	0.1792	9.6986	38.7218	29.0232
ONMNT	27.5000	85.0000	57.5000	2.0142	0.8060	43.5483	0.1740	0.4358	38.7181	58.0729	19.3548
O^2NMNT	35.0000	97.5000	62.5000	2.6331	1.0530	56.1284	0.0880	0.2191	0.0121	56.1350	56.1229

consume approximately 2.62 MWh of power to complete the simulation. Using the relatively new architecture of GPGPU, we can substantially reduce the processing time. As graphic cards are developed using parallel techniques inherently, a large amount of research and development has resulted in this new type of processing using consumer grade equipment. For this application, an AMD HD7970 graphics card was used containing 2,048 streams (effective cores); it runs at a clock speed of 1 GHz and is capable of a peak single precision throughput of 4,300 GFLOPs [147], consuming 218 Watts at full load [148]. From an initial viewpoint and taking into consideration of the additional processing resources and clock speed, we should expect to see an increase in performance by a factor of 682 calculated by

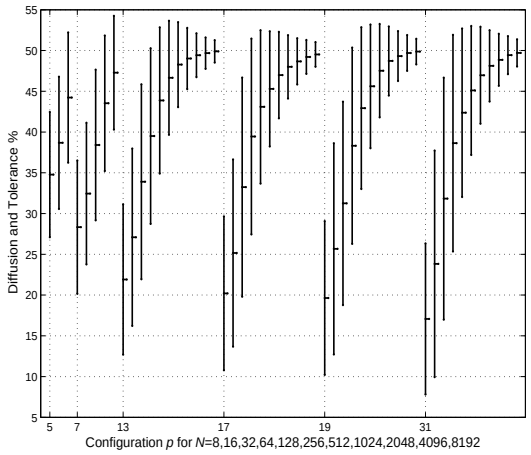
$$\text{Increase Factor} = \frac{\text{cores}_{\text{GPU}} \times \text{Clock}_{\text{GPU}}}{\text{cores}_{\text{CPU}} \times \text{Clock}_{\text{CPU}}}, \tag{6.17}$$

which should reduce the expected completion time to approximately 59 hours and 6 minutes.

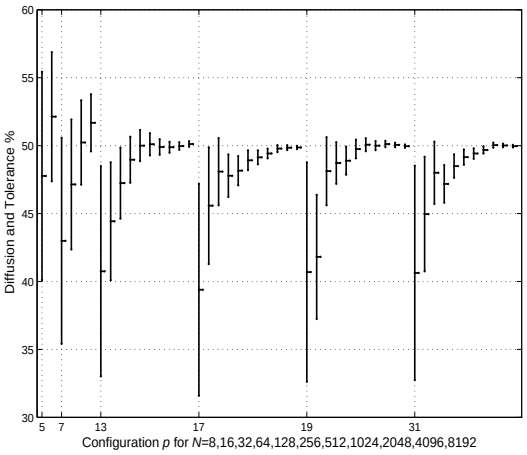
Containing commands such as the fused-multiply-add that can simultaneously perform a 24-bit multiply and a 32-bit addition using four-element vector variables in a single clock cycle, the actual time that the HD7970 took to complete the simulation was a little over 52 hours and 19 minutes. This completion time shows

Table 6.12: Multiple Configuration Analysis of the GNMNT

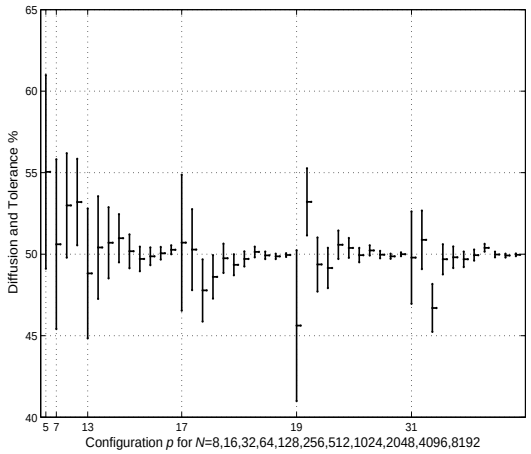
		N										
		8	16	32	64	128	256	512	1024	2048	4096	8192
p	5	1, 2, 3	1, 2	1	-	-	-	-	-	-	-	-
	7	1, 2, 3	1, 2, 3	1, 2, 3	1, 2	1	-	-	-	-	-	-
	13	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2	1
	17	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3
	19	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3
	31	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3	1, 2, 3



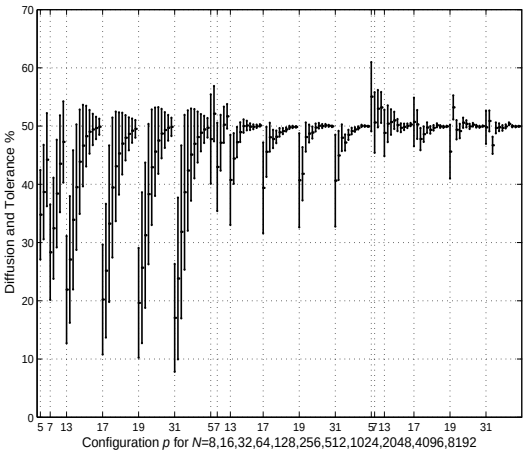
(a) NMNT



(b) ONMNT



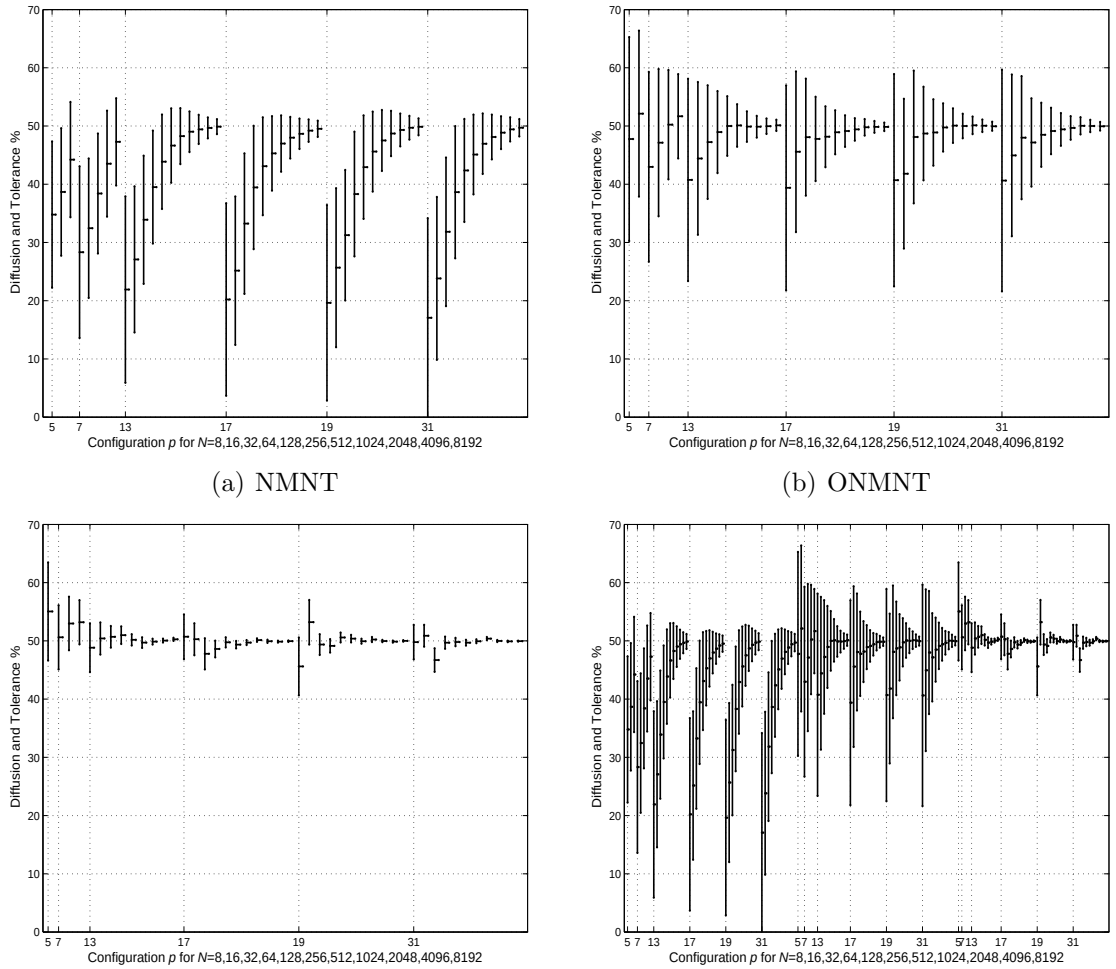
(c) O^2NMNT



(d) GNMNT - NMNT, ONMNT, O^2NMNT

Figure 6.10: GNMNT Vector Analysis with Multiple Configurations

an actual speed increase by a factor of 771. During the course of the simulation it was anticipated that the GPU had a power consumption of approximately 11.41 kWh, which is 0.49% of the power that the CPU would have required. This could possibly be further improved upon as the implementation was designed to process the metrics using atomic operations that naturally introduce bottlenecks in the processing when memory accesses run into contention. The design of the



(a) NMNT

(b) ONMNT

(c) O^2NMNT

(d) GNMNT - NMNT, ONMNT, O^2NMNT

Figure 6.11: GNMNT Bit Position Analysis with Multiple Configurations

implementation could be rearranged so that the program minimised the number of atomic operations. However, this would certainly imply the use of slower memory in order to cope with the volume of data that would be presented in addition to the complex data consolidation requirements. However, with the myriad different ways to approach a problem then obviously more time could be spent searching for superior solutions than to disseminate the actual results already obtained. The results of this simulation are depicted in Figures 6.9(a) and 6.9(b). Looking closer at the statistics that have been produced in Table 6.11, it is interesting to note that this configuration has produced notably different results: the NMNT has increased in performance to almost 34%, the ONMNT has dropped in performance to 43.5% and the O^2NMNT has increased to a little over 56%. In fact, there is a very simple explanation for this profound shift in performance, which is that the configuration has maximised the transform length in retrospect to the selected Mersenne prime for which the transform has been configured. As such, there is a stronger pull for

all transforms to converge to the ideal 50% threshold, although simulating every possible vector permutation doesnt necessarily indicate this. However, this effect can be shown by defining similar vectors across different transform parameters. Using values for p and N that define a more manageable vector range then a more accurate performance of the GNMNT can be measured. For the final configuration, a multiple set of configurations is defined and shown in Table 6.12, which refers to the following transforms: 1 - NMNT, 2 - ONMNT and 3 - O^2NMNT. Each simulation is executed four times using various different vector values that provide a wide range of values across the entire vector. The results that are obtained are then combined to produce an average set of results. Using these results, various plots are produced as shown in Figure group 6.10 that depicts the diffusion effect across the vectors and 6.11 that depicts the diffusion effect down the bit positions. The plots show the mean value that was observed and the application of the standard deviation to show the upper and lower boundaries. It can be seen in these plots, particularly with values of $p = 13$ and greater, that as the length N approaches the maximum length according to p then typically the ideal characteristics are obtained. This would appear to collaborate the results that were depicted in Figures 3.10-3.12, 5.21-5.23 and 5.24-5.26 for the 1D, RC and 2D methods respectively, where it was observed that the longer the transform length the better the transform appeared to perform.

6.6 Conclusion

This chapter has introduced methodologies to apply existing techniques to the GNMNT, which have enabled such a diverse and thorough examination of these transforms and where they have also been compared against the industry standard. The AES has repeatedly shown that it has extremely robust characteristics with respect to diffusion, where it manages to obtain peak performance after only four rounds. The transform from the GNMNT that characteristically better matched to AES was the O^2NMNT, which was able do so in only one round. However, a number of potential weaknesses were exposed within the GNMNT, including the O^2NMNT that show that it is not a transform that can be used solely for encryption purposes. Whilst it is able to perform to the ideal specifications, it is not able to do so consistently without any further intervention. However, the addition of a simple

function did make a difference to the O^2NMNT in such a way that coupled with a round key generator, it could be possible to derive a fairly crude yet fairly robust encryption system without too much effort. An exhaustive analysis was undertaken on the GNMNT using a comparatively conservative configuration. Whilst this configuration was certainly not enough to withstand the rigor of resources that are currently available, the availability of a new type of computing was able to provide an insight into the performance of the GNMNT over all possible vector permutations that were previously out of reach. Moreover, while the configuration used was different to the simulations that were run against the AES, the results brought additional insight that suggested the value of further investigation into a broader configuration range. The final simulation set encompassed a configuration range that spanned common usable ranges and beyond, where it was shown that there are significant advantages to configure p so that the longest value of N can be used. Finally, this study has shown that while there may initially be negative results using particular transforms and configurations, they should not be dismissed without exploring the potential to change to more suitable configurations and applying additional functionality.

Chapter 7

Conclusion and Future Work

It can be seen that the rate of progress with respect to both the development and defeat of methodologies is perpetual and fuelled by theoretical and technological advances. The development of security algorithms was presented in Chapter 2 where it was shown that provided with enough impetus, the amounts of resources that organisations/hackers are prepared to invest to address/defeat security issues could potentially be limitless. Technological advances will also mean that hardware becomes very cheap very quickly and also more reusable.

Newly developed transforms have been implemented and applied as part of an encryption algorithm, where both random keys and similar keys were used to see how effective the raw algorithm was. Unwanted effects manifested from the NMNT and ONMNT algorithms using similar keys that differed by a single bit and proofs were provided to validate the implementation. Initial results suggested that there was particular interest with the O^2NMNT transform as it appeared to resist the similar key attack very well.

The GNMNT has also been fully realised as fast algorithms for one-dimensional applications. Complexity analysis has shown that there are substantial improvements to be gained, especially in consideration of other architectures that possess emerging instructions and techniques to merge operations into a single command and employ parallelism through vector operations. However, depending on the size of the work required, particularly the length when relating to fast algorithms for transforms, it may prove more fortuitous to revert back to direct methods when considering parallel techniques.

The development of separable algorithms for the row-column approach of the

newly developed GNMNT has empowered this suite by not requiring fully developed two-dimensional algorithms. Implementations were verified using the Sobel filter on a number of images before conducting encryption assessments using RC and 2D techniques. Results appeared to lean towards caution upon application of the similar key attack. However, it is well known that encryption systems employ many rounds and techniques and in particular, non-linear operations.

The GNMNT algorithms were implemented using parallel techniques on a consumer based graphics card. In doing so, simulations that were previously out of reach in terms of years were accomplished in hours. The focus on security continues to grow, especially with the development of new architectures and advances in cryptanalysis.

The incorporation of GPGPU computing in this research facilitated the in-depth analysis of the SAC, which has been performed and assessed alongside the algorithm used in the AES. The O^2NMNT was shown to yield comparable results to the AES and suggests that there is potential scope to use it within an encryption system. While the exhaustive analysis of the diffusion aspects for the NMNT and in part the ONMNT are perhaps not as good as expected, consideration must be taken into account where these results were obtained without any confusion techniques. Moreover, obtaining an initial assessment into how well the GNMNT performs within RC and 2D applications, it is clear to see that such an encryption system would benefit from non-linear operations and shuffling, as well as multiple rounds.

Future Work

The development and subsequent assessment of the GNMNT has demonstrated that there is sufficient scope to develop the algorithm further. Areas where this could be feasible and advantageous are:

- further research into the GNMNT for development of new techniques and algorithms that are based upon one- or two-dimensions; examples of using two-dimensional techniques have demonstrated the circular convolution properties of these new transforms, making these transforms accessible to other disciplines such as image-, video- and signal-processing to name but a few

- develop a method of encryption that can incorporate additional functionality; it has previously been demonstrated the strengths of the GNMNT in convolutions, therefore it may well be that an encryption system that would envelop additional convolution-based functionality should also be feasible

- development of adaptable and configurable S-Boxes for encryption systems based upon the GNMNT that are comparable to the S-Boxes used with AES so as to i) add additional and reconfigurable security and ii) derive a secure methodology that can addresses the potential weaknesses that were discovered in Chapter 6

- research new techniques for medical applications to piece and process images from different sources; the advent of mobile technology has fuelled the desire to incorporate many different types of sensors that can measure all sorts of attributes including pulse, heat and humidity and piece together an analysis to how healthy the user may be - such techniques could well be applicable to other sensors and disciplines that may have medical benefits.

Appendix A

Radix-2 GNMNT Algorithms

Listing A.1: Beta Generation

```
// Generate beta1 and beta2 for transforms
void betaGen(unsigned long long *B1, unsigned long long *B2,
             unsigned long long *C1, unsigned long long *C2,
             unsigned long long *E1, unsigned long long *E2,
             unsigned int p, unsigned int log2N)
{

    unsigned long long d, *A1, *A2;
    long long a1, a2, t1, t2;

    A1 = (unsigned long long*)malloc(sizeof(unsigned long long) * fN);
    A2 = (unsigned long long*)malloc(sizeof(unsigned long long) * fN);

    a1 = 2;
    a2 = 3;

    d = p - log2N - 1;
    if (d == 0)
        d = 1;

    for (loop = 0; loop < (p - 2); loop++)
    {
        a1 = mods(a1 * a1, p);
        a2 = mods(a2 * a2, p);
```

```
    }

    for (loop = 0; loop < d; loop++)
    {
        t1 = mods((a1 * a1) - (a2 * a2), p);
        t2 = mods(2 * a1 * a2, p);
        a1 = t1;
        a2 = t2;
    }

    A1[0] = 1;
    A2[0] = 0;
    for (loop = 0; loop < fNmm; loop++)
    {
        A1[loop + 1] = mods((a1 * A1[loop]) - (a2 * A2[loop]), p);
        A2[loop + 1] = mods((a1 * A2[loop]) + (a2 * A1[loop]), p);
    }

    for (loop = 0; loop < N; loop++)
    {
        B1[loop] = A1[loop << 2];
        B2[loop] = A2[loop << 2];
        C1[loop] = A1[loop << 1];
        C2[loop] = A2[loop << 1];
        E1[loop] = A1[(loop << 1) + 1];
        E2[loop] = A2[(loop << 1) + 1];
    }

    free (A2);
    free (A1);
}
```

Listing A.2: Bit Reversal

```
// Perform Bit Reverse Order to a vector
// Derived from: http://graphics.stanford.edu/~seander/bithacks.html#ReverseParallel

void bitrevorder(unsigned long long *vec, unsigned char log2N)
{
    unsigned long long newpos, temp;

    for(loop = 0; loop < log2N; loop ++)
    {
        newpos = loop;
        newpos = ((newpos >> 1) & 0x55555555) | ((newpos & 0x55555555) << 1);
        newpos = ((newpos >> 2) & 0x33333333) | ((newpos & 0x33333333) << 2);
        newpos = ((newpos >> 4) & 0x0F0F0F0F) | ((newpos & 0x0F0F0F0F) << 4);
        newpos = ((newpos >> 8) & 0x00FF00FF) | ((newpos & 0x00FF00FF) << 8);
        newpos >>= (16 - log2N);
        if(loop < newpos)
        {
            temp = vec[loop];
            vec[loop] = vec[newpos];
            vec[newpos] = temp;
        }
    }
}
```

Listing A.3: Radix-2 NMNT (DIT)

```
// Forward and Inverse Radix−2 NMNT (DIT)
void nmnt(unsigned long long *oldvec, unsigned long long *vec,
          unsigned long long *b1, unsigned long long *b2,
          unsigned long long inverse)
{
    unsigned int i, j, k, n, io2, ts, tmask, line1, line2, betaz, offset;
        unsigned int log2N, p;
    long long t1, t2;

    i = 2;
    io2 = 1;
    ts = N >> 1;
    tmask = ts - 1;

    for(n = 0; n < N; n++)
        vec[n] = oldvec[n];

    bitrevorder(vec, log2N);

    for(j = 1; j <= log2N; j++)
    {
        for(k = 0; k < (ts & tmask); k++)
        {
            line1 = (1 << (j - 1)) + (k << j) + 1;
            line2 = (1 << j) + (k << j) - 1;
            offset = (((N >> (ts >> 1)) * (k + 1)) - 1);
            offset = ((k + 1) << (log2N - (ts >> 1))) - 1;
            for(n = 0; n < ((1 << (j - 2))); n++)
            {
                betaz = n << (ts >> 1);
                t1 = mods((vec[line1] * b1[betaz]) + (vec[line2] * b2[betaz]), p);
                t2 = mods((vec[line1] * b2[betaz]) - (vec[line2] * b1[betaz]), p);
                vec[line1] = t1;
                vec[line2] = t2;
                line1++;
```

```
                line2 = offset - n;
            }
        }
        for(k = 0; k < N; k += i)
            for(n = 0; n < io2; n++)
            {
                t1 = vec[k + n];
                t2 = vec[k + n + io2];
                vec[k + n] = mods(t1 + t2, p);
                vec[k + n + io2] = mods(t1 - t2, p);
            }
        io2 = i;
        i <<= 1;
        ts >>= 1;
    }
    for(j = 0; j < N; j++)
        vec[j] = mods(vec[j] * inverse, p);
}
```

Listing A.4: Radix-2 ONMNT (DIT)

```
// Forward Radix−2 ONMNT (DIT)
void onmnt(unsigned long long *oldvec, unsigned long long *vec,
                    unsigned long long *b1, unsigned long long *b2)
{
    unsigned int i, j, k, n, io2, ts, tmask, line1, line2, betaz, step;
    unsigned int log2N, p;
    long long t1, t2;

    i = 2;
    io2 = 1;
    ts = N >> 1;
    tmask = ts - 1;
    betaz = N >> 1;
    line1 = 0;
    line2 = 1;

    for(n = 0; n < N; n++)
        vec[n] = oldvec[n];

    bitrevorder(vec, log2N);

    for(j = 0; j < log2N; j++)
    {
        for(k = j; k < 1; k++)
            for(n = 1; n < N; n += 2)
                vec[n] = mods(vec[n] * b2[betaz], p);
        for(k = 0; k < (ts & tmask); k++)
        {
            line1 = (1 << j) + (k << (j + 1));
            line2 = line1 + (1 << j) - 1;
            step = N >> j;
            betaz = step >> 1;
            for(n = 0; n < (unsigned int)(1 << (j - 1)); n++)
            {
                t1 = mods((vec[line1] * b1[betaz]) + (vec[line2] * b2[betaz]), p);
```

```
                t2 = mods((vec[line1] * b2[betaz]) - (vec[line2] * b1[betaz]), p);
                vec[line1++] = t1;
                vec[line2--] = t2;
                betaz += step;
            }
        }
        for (k = 0; k < N; k += i)
            for (n = 0; n < io2; n++)
            {
                t1 = vec[k + n];
                t2 = vec[k + n + io2];
                vec[k + n] = mods(t1 + t2, p);
                vec[k + n + io2] = mods(t1 - t2, p);
            }
        io2 = i;
        i <<= 1;
        ts >>= 1;
    }
}
```

Listing A.5: Radix-2 ONMNT (DIF)

```
// Inverse Radix-2 ONMNT (DIF)
void ionmnt(unsigned long long *oldvec, unsigned long long *vec,
                        unsigned long long *b1, unsigned long long *b2)
{
    unsigned int i, j, k, n, io2, ts, tmask, line1, line2, betaz, step;
    unsigned int log2N, p;
    long long t1, t2;

    i = N;
    io2 = i >> 1;
    ts = 1;
    tmask = (N >> 1) - 1;
    line1 = 0;
    line2 = 1;

    for(n = 0; n < N; n++)
        vec[n] = oldvec[n];

    for(j = log2N; j > 0; j--)
    {
        for(k = 0; k < N; k += i)
            for(n = 0; n < io2; n++)
            {
                t1 = vec[k + n];
                t2 = vec[k + n + io2];
                vec[k + n] = mods(t1 + t2, p);
                vec[k + n + io2] = mods(t1 - t2, p);
            }
        for(k = 0; k < (ts & tmask); k++)
        {
            line1 = (1 << (j - 1)) + (k << j);
            line2 = line1 + (1 << (j - 1)) - 1;
            step = N >> (j - 1);
            betaz = step >> 1;
            for(n = 0; n < (unsigned int)(1 << (j - 2)); n++)
```

```
            {
                t1 = mods((vec[line1] * b1[betaz]) + (vec[line2] * b2[betaz]), p);
                t2 = mods((vec[line1] * b2[betaz]) - (vec[line2] * b1[betaz]), p);
                vec[line1++] = t1;
                vec[line2--] = t2;
                betaz += step;
            }
        }
        for(k = j; k < 2; k++)
        {
            betaz = N >> 1;
            for(n = 1; n < N; n += 2)
                vec[n] = mods(vec[n] * b2[betaz], p);
        }
        i = io2;
        io2 >>= 1;
        ts <<= 1;
    }


    for(j = 0; j < N; j++)
        vec[j] = mods(vec[j] * sn, p);

    bitrevorder(vec, log2N);
}
```

Listing A.6: Radix-2 O^2NMNT (DIT)

```
// Forward and Inverse Radix−2 O2NMNT (DIT)
void o2nmnt(unsigned long long *oldvec, unsigned long long *vec,
                        unsigned long long *b1, unsigned long long *b2,
                        unsigned long long *e1, unsigned long long *e2,
                        unsigned long long inverse)
{
    unsigned int i, j, k, n, io2, ts, tmask, line1, line2, betaz, step;
    unsigned int log2N, p;
    long long t1, t2;

    i = 2;
    io2 = 1;
    ts = N >> 1;
    tmask = ts - 1;
    betaz = N >> 1;
    line1 = 0;
    line2 = 1;

    for(n = 0; n < N; n++)
        vec[n] = oldvec[n];

    bitrevorder(vec, log2N);

    for(j = 1; j <= log2N; j++)
    {
        for(k = j; k < 2; k++)
            for(n = 1; n < N; n += 2)
            {
                vec[n] = mods(vec[n] * b2[betaz], p);
            }
        for(k = 0; k < (ts & tmask); k++)
        {
            line1 = (1 << (j - 1)) + (k << j);
            line2 = line1 + (1 << (j - 1)) - 1;
            step = N >> (j - 1);
```

```
            betaz = step >> 1;
            for (n = 0; n < ((1 << (j - 2))); n++)
            {
                t1 = mods((vec[line1] * b1[betaz]) + (vec[line2] * b2[betaz]), p);
                t2 = mods((vec[line1] * b2[betaz]) - (vec[line2] * b1[betaz]), p);
                vec[line1] = t1;
                vec[line2] = t2;
                line1++;
                line2--;
                betaz += step;
            }
        }
        for (k = 0; k < N; k += i)
            for (n = 0; n < io2; n++)
            {
                t1 = vec[k + n];
                t2 = vec[k + n + io2];
                vec[k + n] = mods(t1 + t2, p);
                vec[k + n + io2] = mods(t1 - t2, p);
            }
        io2 = i;
        i <<= 1;
        ts >>= 1;
    }

    line1 = 0;
    line2 = N - 1;
    for (n = 0; n < (N >> 1); n++)
    {
        t1 = mods((vec[line1] * e1[line1]) + (vec[line2] * e2[line1]), p);
        t2 = mods((vec[line2] * e1[line2]) + (vec[line1] * e2[line2]), p);
        vec[line1] = t1;
        vec[line2] = t2;
        line1++;
        line2--;
    }
```

```
    for(j = 0; j < N; j++)
        vec[j] = mods(vec[j] * inverse, p);
}
```

Appendix B

Parallel GNMNT Algorithms

Listing B.1: OpenCL GPGPU Code for Exhaustive GNMNT Assessment

```
#pragma OPENCL EXTENSION cl_khr_local_int32_base_atomics : enable

#define LOCK(a) atom_cmpxchg(a, 0, 1)
#define UNLOCK(a) atom_xchg(a, 0)

#define DATA_SIZE 144
#define STORE DATA_SIZE * 6
#define MP 31
#define BITS 5
#define SCALE 4

#define b18x50 1
#define b18x51 27
#define b18x52 0
#define b18x53 4
#define b28x50 0
#define b28x51 4
#define b28x52 30
#define b28x53 4

#define b1O8x50 1
#define b1O8x51 18
#define b1O8x52 27
#define b1O8x53 24
```

```
#define b2O8x50 0
#define b2O8x51 7
#define b2O8x52 4
#define b2O8x53 13

#define b1OS8x50 5
#define b1OS8x51 20
#define b1OS8x52 2
#define b1OS8x53 21
#define b2OS8x50 10
#define b2OS8x51 29
#define b2OS8x52 11
#define b2OS8x53 26

uint     mod1(uint input)
{
    __private  uint     output;

    output = input & MP;
    output = (output != MP) ? output : 0;
    output += ((input >> BITS) & MP);
    return((output + ((output < MP) ? 0: 1)) & MP);
}

uint2     mod2(uint2 input)
{
    __private  uint2     output;

    output = input & MP;
    output = (output != MP) ? output : 0;
    output += ((input >> BITS) & MP);
    return  ((output + ((output < MP) ? (uint)0 : (uint)1)) & MP);
}

uint3     mod3(uint3 input)
{
```

```
    return (uint3)(mod2(input.s01), mod1(input.s2));
}

uint4     mod4(uint4 input)
{
    return (uint4)(mod2(input.s01), mod2(input.s23));
}

uint8     mod8(uint8 input)
{
    return (uint8)(mod2(input.s01), mod2(input.s23), mod2(input.s45), mod2(input.s67));
}

uint8     nmnt(uint8 stage)
{
    __private uint4     temp;
    __private uint2     temp2;

    stage = stage.s04261537;
    // Stage 1
    temp = stage.s1357;
    stage.s1357 = mod4(((uint4)stage.s0246 + (uint4)MP - (uint4)temp));
    stage.s0246 = mod4(((uint4)stage.s0246 + (uint4)temp));
    // Stage 2a
    temp2 = mod2(mul24((uint2)stage.s37, (uint2)(MP - b18x52, MP - b18x52)));
    stage.s37 = mod2(mad24((uint2)stage.s37, (uint2)(b28x52), (uint2)temp2));
    // Stage 2b
    temp = (uint4)stage.s2367;
    stage.s2367 = mod4(((uint4)stage.s0145 + (uint4)MP - (uint4)temp));
    stage.s0145 = mod4(((uint4)stage.s0145 + (uint4)temp));
    // Stage 3a
    stage.s567 = mod3(mad24((uint3)stage.s765, (uint3)(b28x51, b28x52, b28x53), \
                    mod3(mul24((uint3)stage.s567, \
                    (uint3)(b18x51, MP - b18x52, b18x53)))));
    // Stage 3b
    temp = (uint4)stage.s4567;
```

```
    stage.s4567 = mod4(((uint4)stage.s0123 + (uint4)MP - (uint4)temp));
    stage.s0123 = mod4(((uint4)stage.s0123 + (uint4)temp));
    return stage;
}

uint8    inmnt(uint8 stage)
{
    stage = nmnt(stage);
    stage = mod8((stage * (uint8)(SCALE)));
    return stage;
}

uint8    onmnt(uint8 stage)
{
    __private  uint4    temp;

    // ONMNT (DIT)
    // Bit Reverse
    stage = stage.s04261537;
    // Stage 1a
    stage.s1357 = mod4(mul24(stage.s1357, (uint4)(30)));
    // Stage 1b
    temp = stage.s1357;
    stage.s1357 = mod4((stage.s0246 - temp + MP));
    stage.s0246 = mod4((stage.s0246 + temp));
    // Stage 2a
    temp = mod4(mul24(stage.s2367, (uint4)(b1O8x52, MP - b1O8x52, \
                                           b1O8x52, MP - b1O8x52)));
    stage.s2367 = mod4(mad24( stage.s3276, (uint4)(b2O8x52), temp));
    // Stage 2b
    temp = stage.s2367;
    stage.s2367 = mod4((stage.s0145 - temp + MP));
    stage.s0145 = mod4((stage.s0145 + temp));
    // Stage 3a
    temp = mod4(mul24(stage.s4567, (uint4)(b1O8x51, b1O8x53, MP - b1O8x53, \
                                                    MP - b1O8x51)));
```

```
    stage.s4567 = mod4(mad24(stage.s7654, \
                             (uint4)(b2O8x51, b2O8x53, b2O8x53, b2O8x51), temp));
    // Stage 3b
    temp = stage.s4567;
    stage.s4567 = mod4((stage.s0123 - temp + MP));
    stage.s0123 = mod4((stage.s0123 + temp));
    return stage;
}

uint8     ionmnt(uint8 stage)
{
    __private  uint4     temp;

    // iONMNT (DIF)
    // Stage 3b
    temp = stage.s4567;
    stage.s4567 = mod4((stage.s0123 - temp + MP));
    stage.s0123 = mod4((stage.s0123 + temp));
    // Stage 3a
    temp = mod4(mul24(stage.s4567, (uint4)(b1O8x51, b1O8x53, \
                                           MP - b1O8x53, MP - b1O8x51)));
    stage.s4567 = mod4(mad24(stage.s7654, \
                             (uint4)(b2O8x51, b2O8x53, b2O8x53, b2O8x51), temp));
    // Stage 2b
    temp = stage.s2367;
    stage.s2367 = mod4((stage.s0145 - temp + MP));
    stage.s0145 = mod4((stage.s0145 + temp));
    // Stage 2a
    temp = mod4(mul24(stage.s2367, (uint4)(b1O8x52, MP - b1O8x52, \
                                           b1O8x52, MP - b1O8x52)));
    stage.s2367 = mod4(mad24(stage.s3276, (uint4)(b2O8x52), temp));
    // Stage 1b
    temp = stage.s1357;
    stage.s1357 = mod4((stage.s0246 - temp + MP));
    stage.s0246 = mod4((stage.s0246 + temp));
    // Stage 1a
```

```
    stage.s1357 = mod4(mul24(stage.s1357, (uint4)(30)));
    // Bit Reverse
    stage = stage.s04261537;
    // Scale
    stage = mod8((stage * (uint8)(SCALE)));
    return stage;
}

uint8    osnmnt(uint8 stage)
{
    __private  uint4    temp, tempa, tempb;

    // OSNMNT
    // Bit Reverse
    stage = stage.s04261537;
    // Stage 1a
    stage.s1357 = mod4(mul24(stage.s1357, (uint4)(30)));
    // Stage 1b
    temp = stage.s1357;
    stage.s1357 = mod4((stage.s0246 - temp + MP));
    stage.s0246 = mod4((stage.s0246 + temp));
    // Stage 2a
    temp = mod4(mul24(stage.s2367, (uint4)(b1O8x52, MP - b1O8x52, \
                                           b1O8x52, MP - b1O8x52)));
    stage.s2367 = mod4(mad24(stage.s3276, (uint4)(b2O8x52), temp));
    // Stage 2b
    temp = stage.s2367;
    stage.s2367 = mod4((stage.s0145 - temp + MP));
    stage.s0145 = mod4((stage.s0145 + temp));
    // Stage 3a
    temp = mod4(mul24(stage.s4567, (uint4)(b1O8x51, b1O8x53, \
                                    MP - b1O8x53, MP - b1O8x51)));
    stage.s4567 = mod4(mad24(stage.s7654, (uint4)(b2O8x51, b2O8x53, \
                                                  b2O8x53, b2O8x51), temp));
    // Stage 3b
    temp = stage.s4567;
```

```
    stage.s4567 = mod4((stage.s0123 - temp + MP));
    stage.s0123 = mod4((stage.s0123 + temp));
    // Convert ONMNT to OSNMNT
    temp = stage.s7654;
    tempa = mod4(mul24(stage.s0123, (uint4)(b1OS8x50, b1OS8x51, \
                                            b1OS8x52, b1OS8x53)));
    tempb = mod4(mul24(stage.s7654, (uint4)(MP - b1OS8x50, MP - b1OS8x51, \
                                            MP - b1OS8x52, MP - b1OS8x53)));
    stage.s7654 = mod4(mad24(stage.s0123, (uint4)(b2OS8x50, b2OS8x51, \
                                                  b2OS8x52, b2OS8x53), tempb));
    stage.s0123 = mod4(mad24(temp, (uint4)(b2OS8x50, b2OS8x51, \
                                            b2OS8x52, b2OS8x53), tempa));
    return stage;
}

uint8     iosnmnt(uint8 stage)
{
    stage = osnmnt(stage);
    stage = mod8((stage * (uint8)(SCALE)));
    return stage;
}

void     latInc( __local  uint *source, uint data)
{
    if (data & 16 == 16)
        atomic_inc(source);
    if (data & 8 == 8)
        atomic_inc(source + 1);
    if (data & 4 == 4)
        atomic_inc(source + 2);
    if (data & 2 == 2)
        atomic_inc(source + 3);
    if (data & 1 == 1)
        atomic_inc(source + 4);
}
```

```
void     atomicStat(__local uint *source, uint8 data, uint8 diffAbs, uint data2)
{
    atomic_add(source, data.s0);
    atomic_add(source + 1, data.s1);
    atomic_add(source + 2, data.s2);
    atomic_add(source + 3, data.s3);
    atomic_add(source + 4, data.s4);
    atomic_add(source + 5, data.s5);
    atomic_add(source + 6, data.s6);
    atomic_add(source + 7, data.s7);

    atomic_inc(source + 8 + data.s0);
    atomic_inc(source + 14 + data.s1);
    atomic_inc(source + 20 + data.s2);
    atomic_inc(source + 26 + data.s3);
    atomic_inc(source + 32 + data.s4);
    atomic_inc(source + 38 + data.s5);
    atomic_inc(source + 44 + data.s6);
    atomic_inc(source + 50 + data.s7);

    atomic_inc(source + 56 + data2);

    latInc(source + 97, diffAbs.s0);
    latInc(source + 102, diffAbs.s1);
    latInc(source + 107, diffAbs.s2);
    latInc(source + 112, diffAbs.s3);
    latInc(source + 117, diffAbs.s4);
    latInc(source + 122, diffAbs.s5);
    latInc(source + 127, diffAbs.s6);
    latInc(source + 132, diffAbs.s7);
}

__kernel void gnmntAvN8p5(__constant uint *inVec,
                          __global uint *outBuffer,
                          __local uint *dataStore,
                          __local uint *eVal,
```

```
                                __local  uint *bitVal)
{
    __private  uint8     in, stage, domain, diffAbs, diff;

    __private  uint      lid = get_local_id(0);
    __private  uint      gid = get_group_id(0);
    __private  uint      gbl = get_global_id(0);
    __private  uint      y;
    __private  uint      transform; // 0 - NMNT, 1 - ONMNT, 2 - OSNMNT
    __private  uint      e0, e1;

    __private  uchar      ill ;
    __private  uchar     length;
    __private  uchar     doInv;


    if(lid == 0)    // Setup Local Variables
    {
        for(y = 0; y < STORE; y++)
            dataStore[y] = 0;
        for(y = 0; y < 40; y++)
        {
            eVal[y] = y / 5;
            bitVal[y] = 1 << (4 - (y % 5));
        }
    }

    if((gid < 744) || (lid < 31))
    {

        barrier(CLK_LOCAL_MEM_FENCE);

        doInv = 0;

        in = vload8(0, inVec);
```

```
// Translate vector course
// range 0 to 961 or 10 bits worth

in.s2 = mod1(gbl);
gbl = (gbl - in.s2) / MP;
in.s3 = mod1(gbl);
gbl = (gbl - in.s3) / MP;
in.s4 = mod1(gbl);

e1 = 0;

while(e1 < 31)
{
    // Translate vector medium
    in.s1 = e1;
    e0 = 0;
    while(e0 < 31)
    {
        // Translate vector fine
        in.s0 = e0;

        transform = 3;
        while(transform > 0)
        {
            transform--;
            stage = in;
            switch((transform << 1) | doInv)
            {
            case 0:
                // NMNT
                stage = nmnt(stage);
                break;
            case 1:
                // iNMNT
                stage = inmnt(stage);
                break;
```

```
case 2:
    // ONMNT (DIT)
    stage = onmnt(stage);
    break;
case 3:
    // iONMNT (DIF)
    stage = ionmnt(stage);
    break;
case 4:
    // OSNMNT
    stage = osnmnt(stage);
    break;
case 5:
    // iOSNMNT
    stage = iosnmnt(stage);
    break;
default:
    break;
}

domain = stage;
length = 40;
while(length > 0)
{
    length--;
    stage = domain;
    switch(eVal[length])
    {
    case 7:
        stage.s0 ^= bitVal[length];
        ill = 1 - (stage.s0 != MP);
        break;
    case 6:
        stage.s1 ^= bitVal[length];
        ill = 1 - (stage.s1 != MP);
        break;
```

```
case 5:
    stage.s2 ^= bitVal[length];
     ill  = 1 - (stage.s2 != MP);
    break;
case 4:
    stage.s3 ^= bitVal[length];
     ill  = 1 - (stage.s3 != MP);
    break;
case 3:
    stage.s4 ^= bitVal[length];
     ill  = 1 - (stage.s4 != MP);
    break;
case 2:
    stage.s5 ^= bitVal[length];
     ill  = 1 - (stage.s5 != MP);
    break;
case 1:
    stage.s6 ^= bitVal[length];
     ill  = 1 - (stage.s6 != MP);
    break;
case 0:
    stage.s7 ^= bitVal[length];
     ill  = 1 - (stage.s7 != MP);
    break;
default:
    break;
}

switch((transform << 1) | doInv)
{
case 0:
    // iNMNT
    stage = inmnt(stage);
    break;
case 1:
    // NMNT
```

```
                    stage = nmnt(stage);
                    break;
                case 2:
                    // iONMNT (DIF)
                    stage = ionmnt(stage);
                    break;
                case 3:
                    // ONMNT (DIT)
                    stage = onmnt(stage);
                    break;
                case 4:
                    // iOSNMNT
                    stage = iosnmnt(stage);
                    break;
                case 5:
                    // OSNMNT
                    stage = osnmnt(stage);
                    break;
                default:
                    break;
                }

                diffAbs = stage ^ in;
                diff = popcount(diffAbs);

                atomicStat(&dataStore[(((transform << 1) | ill) * \
                        DATA_SIZE)], diff, diffAbs, diff.s0 + diff.s1 + \
                                diff.s2 + diff.s3 + diff.s4 + \
                                diff.s5 + diff.s6 + diff.s7);
            }
        }
        e0++;
    }
    e1++;
  }
}
```

```
    barrier(CLK_LOCAL_MEM_FENCE);

    if(lid == 0)
    {
        gid *= STORE;
        for(y = 0; y < STORE; y++)
            outBuffer[gid + y] = dataStore[y];
    }
}
```

References

[1] A. Wagan, B. Mughal, and H. Hasbullah, "VANET Security Framework for Trusted Grouping Using TPM Hardware," in *Communication Software and Networks, 2010. ICCSN '10. Second International Conference on*, February 2010.

[2] Will Coldwell, The Independent, "The 10 Best Smart Watches," Internet: http://www.independent.co.uk/life-style/fashion/features/the-10-best-smart-watches-8854593.html, October 2013, [5th July 2014].

[3] Jeremy Hsu, IEEE Spectrum Magazine, "How Google Glass Can Improve ATM Banking Security," Internet: http://spectrum.ieee.org/tech-talk/consumer-electronics/gadgets/how-google-glass-can-improve-atm-banking-security, March 2014, [5th July 2014].

[4] J. Rivera and R. van der Meulen, "Gartner Says the Internet of Things Installed Base Will Grow to 26 Billion Units By 2020," Internet: http://www.gartner.com/newsroom/id/2636073, December 2013, [5th July 2014].

[5] *Announcing the Advanced Encryption Standard (AES)*, National Institue of Standards and Technology, 2001.

[6] 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*, vol. 1, Sep 1994, pp. I/547–I/553 vol.1.

[7] S. Boussakta and A. Holt, "New Transform Using the Mersenne Numbers," *Vision, Image and Signal Processing, IEE Proceedings*, vol. 142, no. 6, pp. 381–388, Dec 1995.

[8] S. Boussakta, M. Hamood, and N. Rutter, "Generalized New Mersenne Number Transforms," *Signal Processing, IEEE Transactions on*, vol. 60, no. 5, pp. 2640–2647, May 2012.

[9] R. Agarwal and C. Burrus, "Fast Convolution Using Fermat Number Transforms with Applications to Digital Filtering," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 22, no. 2, pp. 87–97, 1974.

[10] C. M. Rader, "Discrete Convolutions via Mersenne Transforms," *IEEE Trans. Comput.*, vol. C-21, pp. 1269–1273, 1972.

[11] M.T. Hamood, "Development of Efficient Algorithms for Fast Computation of Discrete Transforms," Ph.D. Thesis, School of Electrical, Electronic and Computer Engineering, Newcastle University, 2012.

[12] C. E. Shannon, "Communication Theory of Secrecy Systems," *Bell System Technical Journal*, vol. 28, no. 4, pp. 656–715, 1949.

[13] H. Feistel, "Cryptography and Computer Privacy," *Scientific American*, vol. 228, no. 5, pp. 15–23, 1973.

[14] A. F. Webster and S. E. Tavares, "On the Design of S-Boxes," in *Advances in Cryptology CRYPTO 2003*, ser. Lecture Notes in Computer Science, vol. 218. London, UK, UK: Springer-Verlag, 1986, pp. 523–534.

[15] Advanced Micro Devices, Inc., "OpenCL Zone, AMD Developer Central," Internet: http://developer.amd.com/tools-and-sdks/opencl-zone/, April 2014, [18th April 2014].

[16] NVIDIA Corporation, "CUDA Parallel Computing, GPU Computing on the CUDA Architecture," Internet: http://www.nvidia.co.uk/object/cuda-parallel-computing-uk.html, April 2014, [18th April 2014].

[17] Groupe Speciale Mobile Association (GSMA), "Understanding 7 Billion: Counting Connections and People," Internet: http://www.gsma.com/newsroom/understanding-7-billion-counting-connections-and-people/, April 2014, [12th June 2014].

[18] United States Census Bureau, "World Population - Total Midyear Population for the World: 1950-2050," Internet: http://www.census.gov/population/

international/data/worldpop/table_population.php, December 2013, [20th May 2014].

[19] Samsung, "Samsung Heart Rate Monitor Band," Internet: http://www.samsung.com/uk/consumer/mobile-devices/smartphones/smartphone-accessories/EI-HH10NNBEGWW, August 2014, [6th August 2014].

[20] Symantec, "Security Response: How Safe Is Your Quantified Self? Tracking, Monitoring, and Wearable Tech." Internet: http://http://www.symantec.com/connect/blogs/how-safe-your-quantified-self-tracking-monitoring-and-wearable-tech, July 2014, [6th August 2014].

[21] Sophos, "TJ Maxx Retail Giant Admits Hackers Stole 45 Million Credit Card Details," Internet: http://www.sophos.com/pressoffice/news/articles/2007/03/tjx.html, March 2007, [19th January 2011].

[22] ——, "Millions of British Families at Risk of Identity Theft After HMRC Data Loss, Sophos offers advice," Internet: http://www.sophos.com/pressoffice/news/articles/2007/11/hmrc-id-theft.html, November 2007, [19th January 2011].

[23] H. M. Government, "Data Protection Act 1998," The Crown, Tech. Rep., 1998.

[24] Microsoft, "OneDrive," Internet: https://onedrive.live.com/about/en-gb/, August 2014, [6th August 2014].

[25] Google, "Google Drive," Internet: https://www.google.com/intl/en/drive/, August 2014, [6th August 2014].

[26] Dropbox, "Dropbox," Internet: https://www.dropbox.com/, August 2014, [6th August 2014].

[27] JustCloud, "JustCloud," Internet: http://www.justcloud.com/, August 2014, [6th August 2014].

[28] ZipCloud, "ZipCloud," Internet: http://www.zipcloud.com/, August 2014, [6th August 2014].

[29] David Selinger (Forbes), "Big Data: Getting Ready for the 2013 Big Bang," Internet: http://www.forbes.com/sites/ciocentral/2013/01/15/big-data-get-ready-for-the-2013-big-bang/, January 2013, [9th April 2014].

[30] Charles McLellan (ZDNet), "Storage in 2014: An Overview," Internet: http://www.zdnet.com/storage-in-2014-an-overview-7000024712/, January 2014, [11th May, 2014].

[31] Mike Lynch (BBC), "Data Wars: Unlocking the Information Goldmine," Internet: http://www.bbc.co.uk/news/business-17682304, April 2012, [9th April 2014].

[32] S. C. Mukhopadhyay, Ed., *Internet of Things: Challenges and Opportunities*. Springer, 2014.

[33] Cisco Systems Inc., "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update," Internet: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.pdf, February 2014, [11th May, 2014].

[34] P. Calvocoressi, *Top Secret Ultra*. Knopf Doubleday Publishing Group, 1981.

[35] BBC, originally Jim Finkle (Reuters), "Russia Gang Hacks 1.2 Billion Usernames and Passwords," Internet: http://www.bbc.co.uk/news/technology-28654613, August 2014, [6th August 2014].

[36] S. Singh, *The Code Book: How to Make it, Break it, Hack it, Crack it*. Delacorte Press, 2001.

[37] ——, *The Code Book: The Evolution of Secrecy from Mary Queen of Scots to Quantum Cryptography*. First Anchor Books, 2000.

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

[39] A. Bogdanowicz, "Cryptography Breakthrough Is 100th Milestone: Public-key cryptography receives its due," *The Institute, the IEEE News Source*, 2010.

[40] National Security Agency, "The Case for Elliptic Curve Cryptography," Internet: http://www.nsa.gov/business/programs/elliptic_curve.shtml, January 2009, [6th August 2014].

[41] L. Adleman, "A Subexponential Algorithm for the Discrete Logarithm Problem with Applications to Cryptography," in *Foundations of Computer Science, 1979., 20th Annual Symposium on*, Oct 1979, pp. 55–60.

[42] W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Trans on Inf. Theory*, vol. 22, no. 6, pp. 644–654, November 1976.

[43] R. Rivest, A. Shamir, and L. Adleman, "Cryptographic Communications System and Method," Sep. 20 1983, US Patent 4,405,829. [Online]. Available: http://www.google.com/patents/US4405829

[44] T. ElGamal, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *IEEE Trans on Inf. Theory*, vol. 31, no. 4, pp. 469–472, July 1985.

[45] V. S. Miller, "Use of Elliptic Curves in Cryptography," *Advances in Cryptology*, vol. 218, pp. 417–426, 1985.

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

[47] NIST, "Recommended Elliptic Curves for Federal Government Use," National Institue of Standards and Technology, Tech. Rep., July 1999.

[48] Glenn Greenwald (The Guardian, "Revealed: How US and UK Spy Agencies Defeat Internet Privacy and Security," Internet: http://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security, September 2013, [6th August 2014].

[49] Nicole Perlroth et al. (The New York Times), "N.S.A. Able to Foil Basic Safeguards of Privacy on Web," Internet: http://www.nytimes.com/2013/09/06/us/nsa-foils-much-internet-encryption.html?pagewanted=1&_r=1&, September 2013, [6th August 2014].

[50] Entrust, "Zero to `Dual_EC_DRBG` in 30 minutes," Internet: http://www.entrust.com/wp-content/uploads/2014/02/WP_Zero-to-Dual_EC_DRBG_April2014.pdf, April 2014, [6th August 2014].

[51] Jennifer Huergo (NIST), "NIST Removes Cryptography Algorithm from Random Number Generator Recommendations," Internet: http://www.nist.gov/itl/csd/sp800-90-042114.cfm, April 2014, [6th August 2014].

[52] Kim Zetter (WIRED, "RSA Tells Its Developer Customers: Stop Using NSA-Linked Algorithm," Internet: http://www.wired.com/2013/09/rsa-advisory-nsa-algorithm/, September 2013, [6th August 2014].

[53] Lucian Constantin (PC Workd), "Silent Circle ditches NIST cryptographic standards to thwart NSA spying," Internet: http://www.pcworld.com/article/2051380/silent-circle-moves-away-from-nist-cryptographic-standards-cites-uncertainty.html, October 2013, [6th August 2014].

[54] SafeCurves, "SafeCurves: Choosing Safe Curves for Elliptic-Curve Cryptography," Internet: http://safecurves.cr.yp.to/, January 2014, [6th August 2014].

[55] M. Briceno, I. Goldverg, and D. Wagner, "A Pedagogical Implementation of the GSM A5/1 and A5/2 Voice Privacy Encryption Algorithms," Internet: Availableonlineathttp://cryptome.org/gsm-a512.htm, October 1999, [5th July 2014].

[56] National Institute of Standards and Technology (NIST), "Announcing Draft Federal Information Processing Standard (FIPS) 46-3, Data Encryption Standard (DES), and Request for Comments," Internet: http://csrc.nist.gov/groups/STM/cavp/documents/des/fr990115.htm, January 1999, [4th July 2014].

[57] V. T. Hoang and P. Rogaway, "On Generalized Feistel Networks," in *Advances in Cryptology CRYPTO 2010*, ser. Lecture Notes in Computer Science, vol. 6223. Springer-Verlag, 2010, pp. 613–630.

[58] D. Sharmila and R. Neelaveni, "A Proposed SAFER Plus Security Algorithm Using Fast Walsh Hadamard Transform for Bluetooth Technology,"

International Journal of Wireless & Mobile Networks (IJWMN), vol. 1, no. 80-88, November 2009.

[59] J. L. Massey, G. H. Khachatrian, and M. K. Kuregian, "SAFER+ Cylink Corporations Submission for the Advanced Encryption Standard," in *First AES Candidate Conference (AES1)*, August 1998, [6th August 2014].

[60] R. L. Rivest, M. Robshaw, R. Sidney, and Y. Yin, "The RC6 Block Cipher," Internet: http://people.csail.mit.edu/rivest/pubs/RRSY98.pdf, August 1998, [4th July 2014].

[61] X. B. Yang, S. Boussakta, M. Al-Gailani, and R. Ngadiran, "A New Development of Cryptosystem Using New Mersenne Number Transform," in *Communication Systems Networks and Digital Signal Processing (CSNDSP), 2010 7th International Symposium on*, 2010, pp. 701–705.

[62] Federal Information Processing Standards, "Data Encryption Standard (DES)," Intenet: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf, October 1999, [4th July 2014].

[63] D. E. Denning, "The Data Encryption Standard - Fifteen Years of Public Scrutiny," in *Proceedings of the Sixth Annual Computer Security Applications Conference*. IEEE Computer Society, December 1990.

[64] E. Biham and A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*. London, UK, UK: Springer-Verlag, 1993.

[65] M. Matsui, "Linear Cryptanalysis Method for DES Cipher," in *Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology*, ser. EUROCRYPT '93. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1994, pp. 386–397, http://dl.acm.org/citation.cfm?id=188307.188366.

[66] ——, "The First Experimental Cryptanalysis of the Data Encryption Standard," in *Advances in Cryptology CRYPTO 94*, ser. Lecture Notes in Computer Science, vol. 839. Springer-Verlag, 1994, pp. 1–11.

[67] D. Coppersmith, "The Data Encryption Standard (DES) and its Strength Against Attacks," *IBM Journal of Research and Development*, vol. 38, no. 3, pp. 243–250, May 1994.

[68] DESCHALL (Press Release), "Internet Linked Computers Challenge Data Encryption Standard," Internet: https://archive.today/72SUJ, June 1997, [4th July 2014].

[69] Distributed.Net (Press Release), "Secure Encryption Challenged by Internet," Internet: http://www.distributed.net/images/d/da/19980223_-_PR_-_DES2-1.pdf, February 1998, [21st January 2011].

[70] Electronic Frontier Foundation (Press Release), "EFF DES Cracker Machine Brings Honesty to Crypto Debate: Electronic Frontier Foundation Proves that DES is not Secure," Internet: http://w2.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/HTML/19980716_eff_descracker_pressrel.html, July 1998, [21st January 2011].

[71] RSA Laboratories (Questions and Answers), "Historical: Cryptographic challenges: Des challenge iii," Internet: http://www.emc.com/emc-plus/rsa-labs/historical/des-challenge-iii.htm, January 1999, [4th July 2014].

[72] National Institute of Standards and Technology, "Announcing Development of a Federal Information Processing Standard for Advanced Encryption," Internet: http://csrc.nist.gov/archive/aes/pre-round1/aes_9701.txt, January 1997, [4th July 2014].

[73] J. Nechvatal, E. Barker, D. Dodson, M. Dworkin, J. Foti, and E. Roback, "Status Report on the First Round of the Development of the Advaned Encryption Standard," National Institute of Standards and Technology, Tech. Rep., 1997.

[74] C. C. Burwick, D. Coppersmith, E. DAvignon, R. Gennaro, S. Halevi, C. Jutla, S. M. M. Jr., L. OConnor, M. Peyravian, D. Safford, and N. Zunic, "MARS - a candidate cipher for AES," Internet: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.35.6084&rep=rep1&type=pdf, September 1999, [4th July 2014].

[75] J. Daemen and V. Rijmen, "AES Proposal: Rijndael (ammended)," Internet: http://csrc.nist.gov/archive/aes/rijndael/Rijndael-ammended.pdf, September 1999, [4th July 2014].

[76] R. Anderson, E. Biham, and L. Knudsen, "Serpent: A Proposal for the Advanced Encryption Standard," Internet: http://www.cl.cam.ac.uk/~rja14/Papers/serpent.pdf, June 1998, [4th July 2014].

[77] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, "Twofish: A 128-Bit Block Cipher," Internet: https://www.schneier.com/paper-twofish-paper.pdf, June 1998, [4th July 2014].

[78] National Institute of Standards and Technology, "The Third Advanced Encryption Standard Candidate Conference," in *AES A Crypto Algorithm for the 21st Century*, April 2000.

[79] Miles Smid, CygnaCom Solutions, "AES Round 2 Comments," Internet: http://csrc.nist.gov/archive/aes/round2/comments/20000523-msmid-2.pdf, May 2000, [4th July 2014].

[80] National Institute of Standards and Technology (NIST), "Federal Information Processing Standards (FIPS) Publication 197 Announcing the Advanced Encryption Standard (AES)," Internet: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, November 2001, [20th January 2011].

[81] Bruce Schneier (Schneier on Security), "Another New AES Attack," Internet: https://www.schneier.com/blog/archives/2009/07/another_new_aes.html, July 2009, [17th August 2010].

[82] Morris Dworkin (NIST), "Recommendation for Block Cipher Modes of Operation: Methods and Techniques, Special Publication 800-38A," National Institue of Standards and Technology, Tech. Rep., December 2001.

[83] ——, "Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication, Special Publication 800-38B," National Institue of Standards and Technology, Tech. Rep., May 2005.

[84] ——, "Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality, Special Publication 800-38C," National Institue of Standards and Technology, Tech. Rep., July 2007.

[85] ——, "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, Special Publication 800-38D," National Institue of Standards and Technology, Tech. Rep., November 2007.

[86] ——, "Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices, Special Publication 800-38E," National Institue of Standards and Technology, Tech. Rep., January 2010.

[87] R. Rivest, "The MD4 message digest algorithm," in *Advances in Cryptology - Crypto '90*, ser. Lecture Notes in Computer Science, vol. 537. Springer-Verlag, 1991, pp. 303–311.

[88] ——, "The MD5 Message-Digest Algorithm," Internet Engineering Task Force: Network Working Group, RFC 1321, April 1992.

[89] National Institute of Standards and Technology (NIST), "Federal Information Processing Standards (FIPS) Publication 180-4 Secure Hash Standard (SHS)," Internet: http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf, March 2012, [6th August 2014].

[90] ——, "Federal Information Processing Standards (FIPS) Publication Draft 202 SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," Internet: http://csrc.nist.gov/publications/drafts/fips-202/fips_202_draft.pdf, May 2014, [7th August 2014].

[91] NVIDIA Corporation (Press Release), "New NVIDIA Computing Architecture Enables Data Processing on the GPU for Next-Generation Commercial Applications, Technical Computing, and Advanced Gaming," Internet: http://www.nvidia.com/object/IO_37226.html, November 2006, [19th July 2014].

[92] Giorgos Vasiliadis, Spiros Antonatos, Michalis Polychronakis, Evangelos P. Markatos, and Sotiris Ioannidis, "Gnort: High Performance Network Intrusion Detection Using Graphics Processors," in *Proceedings of the 11th*

International Symposium on Recent Advances in Intrusion Detection (RAID), 2008.

[93] M. Schatz, C. Trapnell, A. Delcher, and A. Varshney, "High-Throughput Sequence Alignment Using Graphics Processing Units," *BMC Bioinformatics 8:474*, 2007.

[94] S. A. Manavski and G. Valle, "CUDA Compatible GPU Cards as Efficient Hardware Accelerators for Smith-Waterman Sequence Alignment," *BMC Bioinformatics 9*, 2008.

[95] Tony Smith (The Register), "AMD Ships Stream Chip and Coding Kit," Internet: http://www.theregister.co.uk/2006/11/14/amd_ships_ctm_stream/, November 2006, [19th July 2014].

[96] Khronos, "Khronos Launches Heterogeneous Computing Initiative," Internet: http://www.khronos.org/news/press/2008/06, June 2008, [14th December 2012].

[97] ——, "The Khronos Group Releases OpenCL 1.0 Specification," Internet: http://www.khronos.org/news/press/2008/12, December 2008, [14th December 2012].

[98] Technologies, "The PlayStation 3 for High-Performance Scientific Computing," Internet: http://icl.cs.utk.edu/news_pub/submissions/c3tech.pdf, May/June 2008, [20th August 2012].

[99] Wired, "Astrophysicist Replaces Supercomputer with Eight PlayStation 3s," Internet: http://www.wired.com/techbiz/it/news/2007/10/ps3_supercomputer, October 2007, [20th August 2012].

[100] CNET, "Say Goodbye to Linux on the PS3," Internet: http://news.cnet.com/8301-13506_3-10471356-17.html, March 2010, [20th August 2012].

[101] Dan Goodwin, ars technica, "Risk Assessment / Security and Hacktivism," Internet: http://arstechnica.com/security/2012/12/25-gpu-cluster-cracks-every-standard-windows-password-in-6-hours/, December 2012, [10th December 2012].

[102] Timothy Prickett Morgan (The Register), "Top 500 supers - The Dawning of the GPUs," Internet: http://www.theregister.co.uk/2010/05/31/top_500_supers_jun2010/, May 2010, [12th December 2012].

[103] O. Alshibami, S. Boussakta, and M. Aziz, "Fast Algorithm for the 2-D New Mersenne Number Transform," *Signal Processing*, vol. 81, no. 8, pp. 1725–1735, 2001.

[104] S. Boussakta, O. Alshibami, and A. Bouridane, "Vector radix-4x4 for Fast Calculation of the 2-D New Mersenne Number Transform," *Signal Processing*, pp. 2231–2244, 2004.

[105] M. T. Hamood and S. Boussakta, "Efficient Algorithms for Computing the New Mersenne Number Transform," *Digital Signal Processing*, vol. 25, pp. 280–288, 2014.

[106] S. Boussakta and A. G. J. Holt, "Number Theoretic Transforms and their Applications in Image Processing," *Adv. Imag. Electron Phys.*, vol. 111, pp. 1–90, 1999.

[107] S. Boussakta, O. Aziz, and A. Holt, "3-D Vector Radix Algorithm for the 3-D New Mersenne Number Transform," *Vision, Image and Signal Processing*, vol. 148, no. 2, pp. 115–125, 2001, IEE Proceedings.

[108] M. Aziz, D. McLernon, and S. Boussakta, "The Implementation of a New 3-D Parallel Filtering Algorithm on the SHARC ADSP21060 Platform," in *Visual Information Engineering, VIE 2003*, 2003, pp. 270–273.

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

[110] M. F. Al-Gailani and S. Boussakta, "Evaluation of One-Dimensional NMNT for Security Applications," in *Communication Systems Networks and Digital Signal Processing (CSNDSP), 2010 7th International Symposium on*, 2010, pp. 715–720.

[111] ——, "New Mersenne Number Transform Diffusion Power Analysis," *American Journal of Engineering and Applied Sciences*, vol. 4, pp. 461–469, 2011.

[112] O. Alshibami, S. Boussakta, M. Aziz, and D. Xu, "Split-Radix Algorithm for the New Mersenne Number Transform," in *The 7th IEEE International Conference on Electronics, Circuits and Systems*, vol. 1. ICECS, 2000, pp. 583–586.

[113] O. Nibouche, S. Boussakta, and M. Darnell, "Radix-4 Decimation-in-Frequency Algorithm for the New Mersenne Number Transform," in *10th IEEE International Conference on Electronics, Circuits and Systems*, vol. 3. ICECS, Dec 2003, pp. 1133–1136.

[114] ——, "Pipeline Architectures for Radix-2 New Mersenne Number Transform," *IEEE Trans. on Circuits and Systems*, vol. 56, no. 8, pp. 1668–1680, Aug 2009.

[115] S. Boussakta and M. T. Hamood, "Rader-Brenner Algorithm for Computing New Mersenne Number Transform," *IEEE Trans. on Circuits and Systems*, vol. 58, no. 8, pp. 532–536, Aug 2011.

[116] J. H. McClellen and C. M. Rader, *Number Theory in Digital Signal Processing*. Prentice-Hall PTR, 1979.

[117] S. A. Martucci, "Symmetric Convolution and the Discrete Sine and Cosine Transforms," *IEEE Trans. Signal Process.*, vol. 42, no. 5, pp. 1038–1051, 1994.

[118] S. Talhah, "Advanced Encryption Techniques Using New Mersenne Number Transforms," Ph.D. dissertation, School of Electronic and Electrical Engineering: Leeds University, 2005.

[119] N. Rutter, S. Boussakta, and A. Bystrov, "Assessment of the One-Dimensional Generalized New Mersenne Number Transform for Security Systems," in *Proceedings of the 77th IEEE Vehicular Technology Conference, VTC Spring 2013, Dresden, Germany, June 2-5*, 2013, pp. 1–5, http://dx.doi.org/10.1109/VTCSpring.2013.6692461.

[120] J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Math. Comp.*, vol. 19, no. 90, pp. 297–301, 1965.

[121] R. Bracewell, "The Fast Hartley Transform," *Proceedings of the IEEE*, vol. 72, no. 8, pp. 1010 – 1018, 1984.

[122] Atmel Corporation, "Atmel AVR 8-bit Instruction Set," Internet: http://www.atmel.com/images/atmel-0856-avr-instruction-set-manual.pdf, July 2014, [30th April 2015].

[123] AMD Developer Central, "AMD Accelerated Parallel Processing OpenCL Programming Guide (rev 2.3)," Internet: http://developer.amd.com/wordpress/media/2013/07/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide-rev-2.7.pdf, November 2013, [18th July 2014].

[124] R. Montoye, E. Hokenek, and S. Runyon, "Design of the IBM RISC System/6000 floating-point execution unit," *IBM Journal of Research and Development*, vol. 34, no. 1, pp. 59–70, Jan 1990.

[125] NVIDIA, "CUDA C Programming Guide (version 6)," Internet: http://docs.nvidia.com/cuda/cuda-c-programming-guide/, February 2014, [18th July 2014].

[126] Intel, "Intel 64 and IA-32 Architectures Optimization Reference Manual," Internet: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf, March 2014, [18th July 2014].

[127] Khronos OpenCL Working Group, "The OpenCL Specification (version 1.2 revision 19)," Internet: https://www.khronos.org/registry/cl/specs/opencl-1.2.pdf, November 2012, [17th August 2013].

[128] M. Flynn, "Some Computer Organizations and Their Effectiveness," *Computers, IEEE Transactions on*, vol. C-21, no. 9, pp. 948–960, Sept 1972.

[129] Intel, "Datasheet: Embedded Pentium Processor with MMX Technology," Internet: http://download.intel.com/support/processors/pentiummmx/sb/24318504.pdf, June 1997, [19th July 2014].

[130] ——, "Intel Many Integrated Core Architecture (Intel MIC Architecture) - Advanced," Internet: http://www.intel.co.uk/content/www/uk/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html, June 2014, [18th July 2014].

[131] D. Zhang, W.-K. Kong, J. You, and M. Wong, "Online Palmprint Identification," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 25, no. 9, pp. 1041–1050, Sept 2003.

[132] J.-A. Lin and C.-S. Fuh, "2D Barcode Image Decoding," *Mathematical Problems in Engineering*, vol. 2013, p. 10, 2013.

[133] A. A. Belal and M. A. Abdel-Gawad, "2D Encryption Mode," Internet: http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/2dem/2dem-spec.pdf, March 2001, [8th August 2014].

[134] M. Mohammed, S. Matilia, and L. Nozal, "Fast 2D Convolution Filter based on Look Up Table FFT," in *Industrial Electronics, 1992., Proceedings of the IEEE International Symposium on*, vol. 1, May 1992, pp. 446–449.

[135] S. Bouguezel, M. Ahmad, and M. Swamy, "A Split-Radix Algorithm for 2-D DFT," in *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, vol. 3, May 2003, pp. 698–701.

[136] J. Daugman, "Complete discrete 2-D Gabor transforms by neural networks for image analysis and compression," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 36, no. 7, pp. 1169–1179, Jul 1988.

[137] J. L. Wu and S.-c. Pei, "The Vector Split-Radix Algorithm for 2D DHT," *Signal Processing, IEEE Transactions on*, vol. 41, no. 2, pp. 960–965, Feb 1993.

[138] S. Boussakta and A. Holt, "New Separable Transform," *Vision, Image and Signal Processing, IEE Proceedings*, vol. 142, no. 1, pp. 27–30, Feb 1995.

[139] M. Hamood, "Development of Efficient Algorithms for Fast Computation of Discrete Transforms," Ph.D. dissertation, School of Electrical, Electronic and Computer Engineering, Newcastle University, 2012.

[140] S. Kay and G. Lemay, "Edge Detection Using the Linear Model," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 34, no. 5, pp. 1221–1227, Oct 1986.

[141] O. Baruch, "Line Thinning by Line Following," *Pattern Recognition Letters*, vol. 8, no. 4, pp. 271 – 276, 1988.

[142] F. Aloul, S. Zahidi, and W. El-Hajj, "Two Factor Authentication Using Mobile Phones," in *Computer Systems and Applications, 2009. AICCSA 2009. IEEE/ACS International Conference on*, May 2009, pp. 641–644.

[143] J. Davis, "Two Factor Auth (2FA)," Internet: https://twofactorauth.org/, December 2014, [4th December 2014].

[144] *FIPS197*, Advanced Encryption Standard. Federal Information Processing Standards Publication, 2001.

[145] Intel, "Intel Core2 Duo Processor E8400," Internet: http://ark.intel.com/products/33910/Intel-Core2-Duo-Processor-E8400-6M-Cache-3_00-GHz-1333-MHz-FSB, 2009, [13th December 2012].

[146] PCSTATS, "Intel Core 2 Duo E8400 3.0GHz 1333MHz FSB Processor Review," Internet: http://www.pcstats.com/articleview.cfm?articleid=2394&page=5, May 2009, [13th December 2012].

[147] Advanced Micro Devices, Inc., "AMD Radeon HD7970 GHz Edition," Internet: http://www.amd.com/us/products/desktop/graphics/7000/7970ghz/Pages/radeon-7970GHz.aspx\#/3, 2012, [31st August 2012].

[148] The Guru of 3D, "Radeon HD7970 GHz edition review," Internet: http://www.guru3d.com/articles_pages/radeon_hd_7970_ghz_edition_review,8.html, June 2012, [13th December 2012].