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 ($O^2$NMNT).

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 $O^2$NMNT 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

<table>
<thead>
<tr>
<th>Nomenclature</th>
<th>xxvi</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Symbols</td>
<td>xxv</td>
</tr>
</tbody>
</table>

### 1 Introduction

| 1.1 Motivation                | 1    |
| 1.2 Aims and Objectives of the Thesis | 3    |
| 1.3 Contributions            | 4    |
| 1.4 Publications Arising From This Research | 5    |
| 1.5 Thesis Outline           | 5    |

### 2 Background

| 2.1 Introduction              | 7    |
| 2.2 Encryption Applications   | 7    |
| 2.3 Encryption Types          | 9    |
| 2.3.1 Symmetric Keys          | 9    |
| 2.3.2 Asymmetric Keys         | 10   |
| 2.3.2.1 Factorisation Problem | 11   |
| 2.3.2.2 Discrete Logarithm Problem | 12   |
| 2.3.2.3 Diffie-Hellman Key Exchange | 13   |
| 2.3.2.4 ElGamal Encryption   | 14   |
| 2.3.2.5 Elliptic Curve Discrete Logarithm Problem | 14   |
| 2.4 Cipher Types              | 20   |
| 2.4.1 Stream Ciphers          | 20   |
| 2.4.1.1 A5/1                 | 21   |
| 2.4.2 Block Ciphers           | 22   |
| 2.4.2.1 Substitution Boxes   | 22   |
### CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.8.3 The O²NMNT Kernel Components</td>
<td>69</td>
</tr>
<tr>
<td>3.8.4 Proving the Detriments of Zero-Elements within the GNMNT</td>
<td>73</td>
</tr>
<tr>
<td>3.8.5 Demonstration of the Proof</td>
<td>76</td>
</tr>
<tr>
<td>3.9 Conclusion</td>
<td>77</td>
</tr>
</tbody>
</table>

#### 4 Fast Algorithms of the GNMNT

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Introduction</td>
<td>79</td>
</tr>
<tr>
<td>4.2 Radix-2 ONMNT</td>
<td>79</td>
</tr>
<tr>
<td>4.2.1 Radix-2 DIT</td>
<td>80</td>
</tr>
<tr>
<td>4.2.2 Radix-2 DIF</td>
<td>83</td>
</tr>
<tr>
<td>4.3 Radix-4 ONMNT</td>
<td>85</td>
</tr>
<tr>
<td>4.3.1 Radix-4 DIT</td>
<td>86</td>
</tr>
<tr>
<td>4.3.2 Radix-4 DIF</td>
<td>90</td>
</tr>
<tr>
<td>4.4 Split-Radix ONMNT</td>
<td>96</td>
</tr>
<tr>
<td>4.4.1 Split-Radix DIT</td>
<td>97</td>
</tr>
<tr>
<td>4.4.2 Split-Radix DIF</td>
<td>99</td>
</tr>
<tr>
<td>4.5 Complexity Analysis</td>
<td>102</td>
</tr>
<tr>
<td>4.5.1 Higher Radices of the GNMNT</td>
<td>108</td>
</tr>
<tr>
<td>4.6 Performance Analysis of the One-Dimensional Derivations</td>
<td>109</td>
</tr>
<tr>
<td>4.7 Conclusion</td>
<td>111</td>
</tr>
</tbody>
</table>

#### 5 The Row-Column GNMNT

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Introduction</td>
<td>113</td>
</tr>
<tr>
<td>5.2 RC-NMNT</td>
<td>114</td>
</tr>
<tr>
<td>5.3 RC-ONMNT</td>
<td>118</td>
</tr>
<tr>
<td>5.4 RC-O²NMNT</td>
<td>123</td>
</tr>
<tr>
<td>5.5 Complexity Analysis</td>
<td>125</td>
</tr>
<tr>
<td>5.6 The 2D Cyclic Convolution for the GNMNT</td>
<td>129</td>
</tr>
<tr>
<td>5.6.1 Cyclic Convolution for the 2D-NMNT</td>
<td>129</td>
</tr>
<tr>
<td>5.6.2 Cyclic Convolution for the 2D-ONMNT</td>
<td>130</td>
</tr>
<tr>
<td>5.6.3 Cyclic Convolution for the 2D-O²NMNT</td>
<td>131</td>
</tr>
<tr>
<td>5.6.4 Verification using Cyclic Convolution</td>
<td>131</td>
</tr>
<tr>
<td>5.7 Encryption Applications</td>
<td>136</td>
</tr>
<tr>
<td>5.7.1 RC-GNMNT Implementations</td>
<td>137</td>
</tr>
</tbody>
</table>
5.7.1.1 RC-NMNT ........................................... 138
5.7.1.2 RC-ONMNT ........................................ 139
5.7.1.3 RC-O\(^2\)NMNT .................................. 140
5.7.1.4 Comparison of RC-Encryption Characteristics ........ 141
5.7.2 2D-GNMNT Implementations ............................. 142
5.7.2.1 2D-NMNT ........................................... 143
5.7.2.2 2D-ONMNT ....................................... 143
5.7.2.3 2D-O\(^2\)NMNT ................................... 144
5.7.2.4 Comparison of 2D-Encryption Characteristics ........ 144
5.7.3 Imposing a Single-Bit Error using the RC- and 2D-GNMNT . 145
5.8 Conclusion ................................................. 152

6 The Avalanche Effect of the GNMNT 155
6.1 Introduction ............................................. 155
6.2 The Strict Avalanche Criterion ............................ 155
6.3 Methodology .............................................. 156
  6.3.1 Applying and Testing the SAC ....................... 156
  6.3.2 AES Implementation .................................. 160
  6.3.3 GNMNT Implementation ............................. 161
6.4 Assessing the Default Configuration ....................... 161
  6.4.1 Assessing the AES .................................... 163
  6.4.2 Assessing the NMNT .................................. 165
  6.4.3 Assessing the ONMNT ................................ 166
  6.4.4 Assessing the O\(^2\)NMNT ............................. 166
6.5 Assessing the Modified Configuration ...................... 167
  6.5.1 Assessing the NMNT (Shuffled) ...................... 167
  6.5.2 Assessing the ONMNT (Shuffled) .................... 168
  6.5.3 Assessing the O\(^2\)NMNT (Shuffled) ................ 168
  6.5.4 Exhaustive Analysis of the GNMNT .................. 169
6.6 Conclusion ................................................. 174

7 Conclusion and Future Work 177

A Radix-2 GNMNT Algorithms 181
List of Figures

1.1 Taxonomy of the GNMNT ........................................... 4
2.1 Addition and Doubling Operations in the Elliptic Curve .......... 16
2.2 The Field of Elliptic Curve Points for $p = 19, a = 2, b = 2$ .... 17
2.3 A5/1 Cipher Scheme ............................................. 21
2.4 Selection of Feistel Networks .................................... 23
2.5 The DES Algorithm ............................................... 25
2.6 Initial and Inverse Permutate Functions .......................... 26
2.7 The F-Box in DES Feistel Network with Round Key ............. 27
2.8 An S-Box from DES .............................................. 28
2.9 The Five Confidentiality Encryption Modes Ratified for AES .... 31
3.1 Convolution Process Structure for the NMNT .................... 53
3.2 Convolution Process Structure for the ONMNT ................... 56
3.3 Convolution Process Structure for the $O^2$NMNT ................ 59
3.4 Encrypted Cameraman using NMNT, $N = 8$ and $M_p = 131071$ . 63
3.5 Encrypted Cameraman using ONMNT, $N = 8$ and $M_p = 131071$ . 64
3.6 Encrypted Cameraman using $O^2$NMNT, $N = 8$ and $M_p = 131071$ . 65
3.7 Decrypted Cameraman Error using GNMNT, $N = 8$ and $M_p = 131071$ 66
3.8 Consistency of Trivial Elements in the NMNT Kernel ............ 67
3.9 Consistency of Trivial Elements in the ONMNT Kernel .......... 68
3.10 Decrypted 256 × 256 Cameraman Bit Error using NMNT with
      $N = 8, 16, 32, 64, 128, M_p = 131071$ ......................... 70
3.11 Decrypted 256 × 256 Cameraman Bit Error using ONMNT with
      $N = 8, 16, 32, 64, 128, M_p = 131071$ ......................... 71
3.12 Decrypted 256 × 256 Cameraman Bit Error using $O^2$NMNT with
      $N = 8, 16, 32, 64, 128, M_p = 131071$ ......................... 72
LIST OF FIGURES

4.1 Radix-2 1D-ONMNT In-Place DIT Butterfly .............................. 81
4.2 Radix-2 1D-ONMNT In-Place DIF Butterfly .............................. 85
4.3 Radix-4 1D-ONMNT In-Place DIT Butterfly .............................. 89
4.4 Radix-4 In-Place DIT Flow Diagram, $N = 16$ ............................ 91
4.5 Radix-4 1D-ONMNT In-Place DIF Butterfly .............................. 94
4.6 Split-Radix DIT Structure .................................................. 97
4.7 Split-Radix 1D-ONMNT In-Place DIT Butterfly ......................... 98
4.8 Split-Radix 1D-ONMNT In-Place DIF Butterfly ......................... 100
4.9 Split-Radix In-Place DIF Flow Diagram, $N = 16$ ...................... 101
4.10 Complexity of Different Radices by Total Operations ................. 105
4.11 Processing Time for Vectors of Length $2^N$ ............................ 106
4.12 Time to Process $\approx 10^{10}$ bits Using Vectors of Length $2^N$ .... 107

5.1 Example of 2D-NMNT using Row-Column Method ....................... 117
5.2 Example of 2D-ONMNT using Row-Column Method ....................... 122
5.3 Example of 2D-O$^2$NMNT using Row-Column Method ................. 124
5.4 Complexity of Different Radices for Row-Column by Total Operations 127
5.5 Complexity of Different Radices for Row-Column with Separable
    Algorithm by Total Operations .......................................... 128
5.6 Convolution Process Structure for the 2D-NMNT ....................... 129
5.7 Convolution Process Structure for the 2D-ONMNT ....................... 130
5.8 Convolution Process Structure for the 2D-O$^2$NMNT .................. 131
5.9 Convolution using Sobel Filter for Edge Detection using Cameraman
    Image with NMNT .......................................................... 132
5.10 Convolution using Sobel Filter for Edge Detection using Lena Image
    with ONMNT .............................................................. 133
5.11 Convolution using Sobel Filter for Edge Detection using Baboon
    Image with O$^2$NMNT .................................................. 134
5.12 Image from which the selected 2D key is obtained .................... 136
5.13 Encrypted Cameraman using RC-NMNT, $N = 8 \times 8$ and $Mp = 131071138$
5.14 Encrypted Cameraman using RC-ONMNT, $N = 8 \times 8$ and $Mp = 131071139$
5.15 Encrypted Cameraman using RC-O$^2$NMNT, $N = 8 \times 8$ and $Mp = 131071140$
5.16 Decrypting Cameraman Error using RC-GNMNT, $N = 8 \times 8$ and
    $Mp = 131071$ ......................................................... 141
5.17 Encrypted Cameraman using 2D-NMNT, $N = 8 \times 8$ and $Mp = 131071$
5.18 Encrypted Cameraman using 2D-ONMNT, $N = 8 \times 8$ and $Mp = 131071$
5.19 Encrypted Cameraman using 2D-O$^2$NMNT, $N = 8 \times 8$ and $Mp = 131071$
5.20 Decrypted Cameraman Error using 2D-GNMNT, $N = 8 \times 8$ and $Mp = 131071$
5.21 Decrypted Bit Error using RC-NMNT with $N = 8, 16, 32, 64, 128$ and $Mp = 131071$
5.22 Decrypted Bit Error using RC-ONMNT with $N = 8, 16, 32, 64, 128$ and $Mp = 131071$
5.23 Decrypted Bit Error using RC-O$^2$NMNT with $N = 8, 16, 32, 64, 128$ and $Mp = 131071$
5.24 Decrypted Bit Error using 2D-NMNT with $N = 8, 16, 32, 64, 128$ and $Mp = 131071$
5.25 Decrypted Bit Error using 2D-ONMNT with $N = 8, 16, 32, 64, 128$ and $Mp = 131071$
5.26 Decrypted Bit Error using 2D-O$^2$NMNT with $N = 8, 16, 32, 64, 128$ and $Mp = 131071$

6.1 Avalanche Assessment Process ............... 157
6.2 AES Analysis ................................... 162
6.3 NMNT Analysis ................................ 163
6.4 ONMNT Analysis .............................. 164
6.5 O$^2$NMNT Analysis ........................... 165
6.6 NMNT Analysis (Shuffled) .................... 168
6.7 ONMNT Analysis (Shuffled) ............... 169
6.8 O$^2$NMNT Analysis (Shuffled) ............ 170
6.9 GNMNT Exhaustive Analysis: $p = 5, N = 8$ 171
6.10 GNMNT Vector Analysis with Multiple Configurations ........... 172
6.11 GNMNT Bit Position Analysis with Multiple Configurations ...... 173
# List of Tables

2.1 Letter Indexing for Caesar Cipher with Key = 3 ............................................. 9  
2.2 Encryption Using Vigenère Cipher ................................................................. 10  
2.3 Decryption Using Vigenère Cipher ................................................................. 10  
2.4 NIST Recommended Key Sizes .......................................................................... 11  
2.5 Elliptic Curve Multiplication $(13 \times P)$ using Double and Add .................... 18  
2.6 NIST Prime Elliptic Curve P-192 ..................................................................... 18  
2.7 LFSR Clocking Behaviour for A5/1 ................................................................. 21  
2.8 Votes for AES Selection ..................................................................................... 29  

3.1 The NMNT Kernel for $N = 16$ and $Mp = 127$ ............................................. 41  
3.2 The ONMNT Kernel for $N = 16$ and $Mp = 127$ ............................................. 42  
3.3 The IONMNT Matrix for $N = 16$ and $Mp = 127$ ........................................... 43  
3.4 The $O^2$NMNT Kernel for $N = 16$ and $Mp = 127$ ....................................... 45  
3.5 Values of $\alpha_1$ and $\alpha_2$ According to $p$ for NMNT ............................... 45  
3.6 Values of $\alpha_1$ and $\alpha_2$ According to $p$ and $N$ for NMNT ....................... 46  
3.7 Values of $\alpha_1$ and $\alpha_2$ According to $p$ and $N$ for ONMNT ...................... 47  
3.8 Values of $\alpha_1$ and $\alpha_2$ According to $p$ and $N$ for $O^2$NMNT ................. 48  
3.9 GNMNT Transform Selection .......................................................................... 49  

4.1 ONMNT Radix-2 Complexity ............................................................................. 102  
4.2 ONMNT Radix-4 Complexity ............................................................................ 103  
4.3 ONMNT Split-Radix Complexity ...................................................................... 103  
4.4 ONMNT Fused-Split-Radix Complexity .......................................................... 104  
4.5 Available Lengths $N = r^s$ According to Radix ............................................. 108  

5.1 Effective Lengths of 1D and 2D GNMNTs ....................................................... 135  
5.2 Pixel Values of 2D Key without Concatenation ............................................... 137  
5.3 Pixel Values of 2D Key with Concatenation ..................................................... 137
6.1 Modification of $\hat{t}_3$ for $p = 3, N = 4$ ........................................ 158
6.2 Resultant Values of $\hat{t}$ for $p = 3, N = 4$ ........................................ 158
6.3 Differences Between $t$ and $\hat{t}$ for $p = 3, N = 4$ ........................... 159
6.4 AES Metrics - Vector / Bit Position ................................................. 162
6.5 NMNT Metrics - Vector / Bit Position ............................................. 163
6.6 ONMNT Metrics - Vector / Bit Position ........................................... 164
6.7 $O^2$NMNT Metrics - Vector / Bit Position ....................................... 165
6.8 NMNT Metrics - Vector / Bit Position (Shuffled) ............................... 168
6.9 ONMNT Metrics - Vector / Bit Position (Shuffled) ............................. 169
6.10 $O^2$NMNT Metrics - Vector / Bit Position (Shuffled) ......................... 170
6.11 GNMNT Exhaustive Analysis: $p = 5, N = 8$ .................................... 171
6.12 Multiple Configuration Analysis of the GNMNT ................................. 172
## Nomenclature

### Acronyms

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

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIF</td>
<td>Decimation-in-Frequency</td>
</tr>
<tr>
<td>DIT</td>
<td>Decimation-in-Time</td>
</tr>
<tr>
<td>DLP</td>
<td>Discrete Logarithm Problem</td>
</tr>
<tr>
<td>DSA</td>
<td>Digital Signature Algorithm</td>
</tr>
<tr>
<td>EB</td>
<td>Exabyte ($10^{18}$)</td>
</tr>
<tr>
<td>ECB</td>
<td>Electronic Code Book</td>
</tr>
<tr>
<td>ECC</td>
<td>Elliptic Curve Cryptography</td>
</tr>
<tr>
<td>ECDLP</td>
<td>Elliptic Curve Discrete Logarithm Problem</td>
</tr>
<tr>
<td>ECDSA</td>
<td>Elliptic Curve Digital Signature Algorithm</td>
</tr>
<tr>
<td>EFF</td>
<td>Electronic Frontier Foundation</td>
</tr>
<tr>
<td>EiB</td>
<td>Exbabyte ($2^{60}$)</td>
</tr>
<tr>
<td>FIPS</td>
<td>Federal Information Processing Standards</td>
</tr>
<tr>
<td>FNT</td>
<td>Fermat Number Transform</td>
</tr>
<tr>
<td>GB</td>
<td>Gigabyte ($10^9$)</td>
</tr>
<tr>
<td>GF</td>
<td>Galois Field</td>
</tr>
<tr>
<td>GFLOPS</td>
<td>$10^9$ Floating Point Operations per Second</td>
</tr>
<tr>
<td>GiB</td>
<td>Gibibyte ($2^{30}$)</td>
</tr>
<tr>
<td>GNMNT</td>
<td>Generalised NMNT</td>
</tr>
<tr>
<td>GPGPU</td>
<td>General-Purpose computing on Graphics Processing Units</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
</tbody>
</table>
| GSM          | Global System for Mobile Communications  
formally Groupe Spécial Mobile |
<p>| IoT          | Internet of Things |
| IV           | Initialisation Vector |</p>
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>KB</td>
<td>Kilobyte (10^3)</td>
</tr>
<tr>
<td>KiB</td>
<td>Kibibyte (2^{10})</td>
</tr>
<tr>
<td>KPA</td>
<td>Known Plaintext Attack</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>LC</td>
<td>Linear Cryptanalysis</td>
</tr>
<tr>
<td>LFSR</td>
<td>Linear-Feedback Shift Register</td>
</tr>
<tr>
<td>MAC</td>
<td>Message Authentication Code</td>
</tr>
<tr>
<td>MB</td>
<td>Megabyte (10^6)</td>
</tr>
<tr>
<td>MiB</td>
<td>Mebibyte (2^{20})</td>
</tr>
<tr>
<td>MIC</td>
<td>Many Integrated Core</td>
</tr>
<tr>
<td>MMX</td>
<td>Multimedia Extensions</td>
</tr>
<tr>
<td>MNT</td>
<td>Mersenne Number Transform</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology</td>
</tr>
<tr>
<td>NMNT</td>
<td>New Mersenne Number Transform</td>
</tr>
<tr>
<td>NTT</td>
<td>Number Theoretic Transform</td>
</tr>
<tr>
<td>O$^2$NMNT</td>
<td>Odd-Squared NMNT</td>
</tr>
<tr>
<td>OFB</td>
<td>Output Feedback</td>
</tr>
<tr>
<td>ONMNT</td>
<td>Odd NMNT</td>
</tr>
<tr>
<td>OpenCL</td>
<td>Open Compute Language</td>
</tr>
<tr>
<td>OTP</td>
<td>One-Time Pad</td>
</tr>
<tr>
<td>PAN</td>
<td>Personal Area Network</td>
</tr>
<tr>
<td>PB</td>
<td>Petabyte (10^{15})</td>
</tr>
<tr>
<td>PHT</td>
<td>Pseudo-Hadamard Transform</td>
</tr>
<tr>
<td>PiB</td>
<td>Pebibyte (2^{50})</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>RC</td>
<td>Row-Column</td>
</tr>
<tr>
<td>RSA</td>
<td>Rivest-Shamir-Adleman Cryptosystem</td>
</tr>
<tr>
<td>SAC</td>
<td>Strict Avalanche Criterion</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SIMD</td>
<td>Single-Instruction Multiple-Data</td>
</tr>
<tr>
<td>SPN</td>
<td>Substitution-Permutation Network</td>
</tr>
<tr>
<td>TB</td>
<td>Terabyte ($10^{12}$)</td>
</tr>
<tr>
<td>TDEA</td>
<td>Triple Data Encryption Algorithm</td>
</tr>
<tr>
<td>TDES</td>
<td>Triple DES</td>
</tr>
<tr>
<td>TiB</td>
<td>Tebibyte ($2^{40}$)</td>
</tr>
<tr>
<td>WAN</td>
<td>Wide Area Network</td>
</tr>
<tr>
<td>WHT</td>
<td>Walsh-Hadamard transform</td>
</tr>
<tr>
<td>XEX</td>
<td>XOR Encrypt XOR</td>
</tr>
<tr>
<td>XOR</td>
<td>Exclusive-OR</td>
</tr>
<tr>
<td>XTS</td>
<td>XEX Tweakable Block Cipher with Ciphertext Stealing</td>
</tr>
<tr>
<td>ZB</td>
<td>Zettabyte ($10^{21}$)</td>
</tr>
<tr>
<td>ZiB</td>
<td>Zebibyte ($2^{70}$)</td>
</tr>
</tbody>
</table>
List of Symbols

\( \alpha \) Generator for \( \beta \)
\( \beta \) Kernel generator used in deriving GNMNTs
\( \delta \) Difference between vectors expressed in bits
\( \Delta \) Difference between vectors expressed as a percentage
\( \lambda \) Normalisation factor, mults to adds
\( \theta \) Representation of beta quantities, \( \frac{2k+1}{2} \)
\( \sigma \) Standard deviation
\( \tau \) Normalised standard deviation expressed in bits
\( \phi \) Representation of beta quantities, \( \frac{2n+1}{2} \)
\( \varphi \) 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
\( C_5 \) Cipher vector in GNMNT domain
\( E \) Set of elliptic curve points
\( GF \) Galois field
\( Mp \) Mersenne prime
\( k \) \( k \)th element in the GNMNT domain
\( K \) Encryption key
\( n \) \( n \)th element in the number domain
### Nomenclature

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>Length of a vector</td>
</tr>
<tr>
<td>$p$</td>
<td>generator for $M_p$, $M_p = 2^p - 1$</td>
</tr>
<tr>
<td>$r$</td>
<td>Recovered plaintext vector</td>
</tr>
<tr>
<td>$\hat{r}$</td>
<td>Alternate recovered plaintext vector</td>
</tr>
<tr>
<td>$R_g$</td>
<td>Recovered plaintext vector in GNMNT domain</td>
</tr>
<tr>
<td>$t$</td>
<td>Plaintext vector</td>
</tr>
<tr>
<td>$\hat{t}$</td>
<td>Alternate plaintext vector</td>
</tr>
<tr>
<td>$T_g$</td>
<td>Plaintext vector in GNMNT domain</td>
</tr>
<tr>
<td>$x$</td>
<td>Vector in the number domain</td>
</tr>
<tr>
<td>$\bar{x}$</td>
<td>Mean value</td>
</tr>
<tr>
<td>$X$</td>
<td>Vector in the NMNT domain</td>
</tr>
<tr>
<td>$X_O$</td>
<td>Vector in the ONMNT domain</td>
</tr>
<tr>
<td>$X_{O^2}$</td>
<td>Vector in the $O^2$NMNT domain</td>
</tr>
<tr>
<td>$\land$</td>
<td>Bitwise AND operation</td>
</tr>
<tr>
<td>$\oplus$</td>
<td>Bitwise Exclusive-Or (XOR) operation</td>
</tr>
<tr>
<td>$\otimes$</td>
<td>Point-by-point multiplication</td>
</tr>
<tr>
<td>$\ast$</td>
<td>Convolution operator</td>
</tr>
<tr>
<td>$\circ$</td>
<td>Convolution operator using the GNMNT</td>
</tr>
<tr>
<td>$&lt; . &gt;_{M_p}$</td>
<td>Modulo $M_p$</td>
</tr>
</tbody>
</table>
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
1. INTRODUCTION

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 use 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 (O²NMNT), 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
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


1.5 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
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 substitution 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.

| 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 |

Table 2.1: Letter Indexing for Caesar Cipher with Key = 3
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 today’s 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>
<thead>
<tr>
<th>Table 2.2: Encryption Using Vigenère Cipher</th>
</tr>
</thead>
<tbody>
<tr>
<td>h i s i s t o p s e c r e t</td>
</tr>
<tr>
<td>P A S S W O R D P A S S W O R +</td>
</tr>
<tr>
<td>i h a k e g k r e s w u n s k</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2.3: Decryption Using Vigenère Cipher</th>
</tr>
</thead>
<tbody>
<tr>
<td>i h a k e g k r e s w u n s k</td>
</tr>
<tr>
<td>P A S S W O R D P A S S W O R −</td>
</tr>
<tr>
<td>t h i s i s t o p s e c r e t</td>
</tr>
</tbody>
</table>
2.3 Encryption Types

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

\[ n = pq. \quad (2.1) \]

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

\[ \varphi = (p - 1)(q - 1). \quad (2.2) \]

A public key \( e \) is selected such that \( e < n \) and the greatest common divisor of \( e \) and \( \varphi \) is equal to 1 such that

\[ \gcd(e, \varphi) = 1. \quad (2.3) \]

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

\[ d = e^{-1} \pmod{\varphi}. \quad (2.4) \]

Encryption is performed using

\[ c = m^e \pmod{n} \quad (2.5) \]

and similarly, decryption is performed by

\[ m = c^d \pmod{n}. \quad (2.6) \]

Table 2.4: NIST Recommended Key Sizes

<table>
<thead>
<tr>
<th>Key Size (bits)</th>
<th>Symmetric</th>
<th>RSA and Diffie-Hellman</th>
<th>Elliptic Curve</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>1024</td>
<td>160</td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>2048</td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>3072</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>192</td>
<td>7680</td>
<td>384</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>15360</td>
<td>521</td>
<td></td>
</tr>
</tbody>
</table>
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

\[ \alpha^x \equiv \beta \mod p \]  

for \( x \) such that

\[ x = \log_\alpha \beta \mod p \]

having previously disclosed \( \alpha, \beta \) 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

\[ (\mathbb{Z}/p\mathbb{Z})^* = \{1, 2, \ldots, p-1\} \].

This field is computed from \( \alpha^0 \) to \( \alpha^{p-2} \) providing \( p - 1 \) elements using generator \( \alpha \) as

\[ (\mathbb{Z}/p\mathbb{Z})^* = \{\alpha^0, \alpha^1, \ldots, \alpha^{p-2}\} \].

The field is calculated up to \( p - 2 \) because

\[ \alpha^{p-1} \equiv 1 \mod p. \]
2.3 Encryption Types

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

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

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 \( 7^{11} \equiv 14 \mod 17 \). The problem then lies within solving \( x \) with knowledge of \( \alpha, \beta \) and \( p \) such that

\[x = \log_\alpha \mod p. \tag{2.14}\]

However, as \( p \) increases it becomes extremely difficult to derive \( x \) from \( \alpha \) and \( \beta \).

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, \( \alpha \). Each party will derive an ephemeral or temporary key in the range \( 1 < k < p - 1 \) and apply it to the generator as

\[K_A = \alpha^{k_a} \mod p \text{ for } 0 < k_a < p - 1 \tag{2.15}\]

and

\[K_B = \alpha^{k_b} \mod p \text{ for } 0 < k_b < p - 1. \tag{2.16}\]

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

\[(K_A)^{k_b} = S_k = (K_B)^{k_a}. \tag{2.17}\]
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 \( \alpha \) 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
\[
K_A = \alpha^{k_a} \mod p
\tag{2.19}
\]
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
\[
\gamma = \alpha^{k_b} \mod p
\tag{2.20}
\]
and
\[
\delta = mK_A^{k_b} \mod p
\tag{2.21}
\]
sending \( c = (\gamma, \delta) \) to Alice. Upon receiving \( c \), Alice uses her private key \( a \) to derive
\[
\gamma^{p-1-k_a} \mod p
\tag{2.22}
\]
which because
\[
\gamma^{p-1-k_a} = \gamma^{-k_a}
\tag{2.23}
\]
this operation is equivalent to deriving \( \alpha^{-k_ak_b} \). Alice can now decode the message by
\[
m = \gamma^{-a}\delta \mod p.
\tag{2.24}
\]

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
2.3 Encryption Types

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

\[ E : y^2 ≡ x^3 + ax + b \mod p \]  \hspace{1cm} (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

\[ 4a^3 + 27b^2 \not\equiv 0 \mod p. \]  \hspace{1cm} (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

\[ Q = dG \mod p \]  \hspace{1cm} (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

\[ s = \frac{y_2 - y_1}{x_2 - x_1} \mod p \]  \hspace{1cm} (2.28)

if the point is to be added with another point or

\[ s = \frac{3x_1^2 + a}{2y_1} \mod p \]  \hspace{1cm} (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

$$x^{-1} = x^{p-2} \mod p. \quad (2.30)$$

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
derived by

\[ x_3 = s^2 - x_1 - x_2 \mod p \]  \hspace{1cm} (2.31)

and

\[ y_3 = s(x_1 - x_3) \mod p. \]  \hspace{1cm} (2.32)

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

\[ P = (x_1, y_1) \mod p \]  \hspace{1cm} (2.33)

then

\[ -P = (x_1, -y_1) \mod p \]  \hspace{1cm} (2.34)
such that
\[ Q - P = (x_2, y_2) - (x_1, y_1) \mod p = (x_2, y_2) + (-x_1, -y_1) \mod p. \] (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
\[ K_A = k_a G \mod p \] (2.36)
and
\[ K_B = k_b G \mod p \] (2.37)

Sending each other their public keys, Alice and Bob apply their private keys similar to (2.17) such that
\[ k_a K_B \mod p = S_k = k_b K_A \mod p. \] (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>
<thead>
<tr>
<th>Step</th>
<th>Operation</th>
<th>Result</th>
<th>Representative</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>( P )</td>
<td>( 1_2 P )</td>
</tr>
<tr>
<td>2a</td>
<td>Double</td>
<td>( P + P )</td>
<td>2P</td>
</tr>
<tr>
<td>2b</td>
<td>Add</td>
<td>( 2P + P )</td>
<td>3P</td>
</tr>
<tr>
<td>3</td>
<td>Double</td>
<td>( 3P + 3P )</td>
<td>6P</td>
</tr>
<tr>
<td>4a</td>
<td>Double</td>
<td>( 6P + 6P )</td>
<td>12P</td>
</tr>
<tr>
<td>4b</td>
<td>Add</td>
<td>( 12P + P )</td>
<td>13P</td>
</tr>
</tbody>
</table>

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

Table 2.6: NIST Prime Elliptic Curve P-192

\[ p_{192} = 62771017353868680763835789423207666416083908700390324961279 \]
\[ a = 62771017353868680763835789423207666416083908700390324961276 \]
\[ b = 245515546008943817740293915197451784769108058161191238065 \]
\[ G_x = 60204628237568865675821348058752611191669897663688468418 \]
\[ G_y = 174050332293622031404857552280219410364023488927386650641 \]
2.3 Encryption Types

2.6, we can perform the following example where

\[
K_A = k_a G \mod p = 354684134338427178248341788335664844735957824203528261950 \\
\times (602046282375688656758213480587526111916698976636884684818, \\
174050332293622031404855552280219410364023488927386650641) \\
= (2933140742448625203731641248407683827818026042006007265108, \\
1342160346315844524113810504268757567486830356720401950124)
\]

\[
K_B = k_b G \mod p = 1799075742360715006626509900004488067778762135129614130234431 \\
\times (602046282375688656758213480587526111916698976636884684818, \\
174050332293622031404855552280219410364023488927386650641) \\
= (414712324657986268199761933154009809426422151451232702766, \\
475414318105551686357210275358715493405392185760408272858)
\]

\[
S_k = k_a K_B \mod p = 354684134338427178248341788335664844735957824203528261950 \\
\times (414712324657986268199761933154009809426422151451232702766, \\
475414318105551686357210275358715493405392185760408272858) \\
= (472803943651158730618988180737355535665961855491617197373, \\
4253469622903940143028516631910075165803201780065008295480)
\]

\[
k_b K_A \mod p = 1799075742360715006626509900004488067778762135129614130234431 \\
\times (2933140742448625203731641248407683827818026042006007265108, \\
1342160346315844524113810504268757567486830356720401950124) \\
= (472803943651158730618988180737355535665961855491617197373, \\
4253469622903940143028516631910075165803201780065008295480)
\]

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.
2.4 Cipher Types

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

\[
LFSR_1 = x^{19} + x^5 + x^2 + x + 1
\]  
\[
LFSR_2 = x^{22} + x + 1
\]  
\[
LFSR_3 = x^{23} + x^{15} + x^2 + x + 1.
\]

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

![Diagram of A5/1 Cipher Scheme]

Table 2.7: LFSR Clocking Behaviour for A5/1

<table>
<thead>
<tr>
<th>Clocking Bit</th>
<th>Movement</th>
</tr>
</thead>
<tbody>
<tr>
<td>( LFSR_1 )</td>
<td>( LFSR_2 )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 2.3: A5/1 Cipher Scheme
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.
2.4 Cipher Types

![Diagram of Cipher Types]

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. BACKGROUND

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.
2.4 Cipher Types

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 2.5: The DES Algorithm
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
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

\[ c = DES \{ E_{k_3}, DES \{ D_{k_2}, DES \{ E_{k_1}, m \} \} \} \]  \hspace{1cm} (2.45)
Figure 2.8: An S-Box from DES

\[
m = \text{DES}\{D_{k_1}, \text{DES}\{E_{k_2}, \text{DES}\{D_{k_3}, c\}\}\}\}
\]

for encryption and 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 doesn't 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>
<thead>
<tr>
<th>Algorithm</th>
<th>Yes</th>
<th>No</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rijndael</td>
<td>86</td>
<td>10</td>
<td>76</td>
</tr>
<tr>
<td>Serpent</td>
<td>59</td>
<td>7</td>
<td>52</td>
</tr>
<tr>
<td>Twofish</td>
<td>31</td>
<td>21</td>
<td>10</td>
</tr>
<tr>
<td>RC6</td>
<td>23</td>
<td>37</td>
<td>-14</td>
</tr>
<tr>
<td>MARS</td>
<td>13</td>
<td>84</td>
<td>-71</td>
</tr>
</tbody>
</table>

Table 2.8: Votes for AES Selection
2. BACKGROUND

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

\[ C_j = E_K(P_j) \text{ for } j = 1, 2, \ldots, n \quad (2.47) \]

for encryption and

\[ P_j = D_K(C_j) \text{ for } j = 1, 2, \ldots, n \quad (2.48) \]

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

\[ C_0 = IV \]
\[ C_j = E_K(P_j \oplus C_{j-1}) \text{ for } j = 1, 2, \ldots, n \quad (2.49) \]

and for decryption

\[ C_0 = IV \]
\[ P_j = D_K(C_j) \oplus C_{j-1} \text{ for } j = 1, 2, \ldots, n. \quad (2.50) \]
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

\[
C_0 = IV
\]

\[
C_j = E_K (C_{j-1}) \oplus P_j \text{ for } j = 1, 2, \ldots, n.
\]
and decryption as

\[ C_0 = IV \]
\[ P_j = E_K (C_{j-1}) \oplus C_j \text{ for } j = 1, 2, \ldots, n. \] (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

\[ O_0 = IV \]
\[ O_j = E_K (O_{j-1}) \text{ for } j = 1, 2, \ldots, n. \] (2.53)

then

\[ C_j = P_j \oplus O_j \] (2.54)

for encryption and

\[ P_j = C_j \oplus O_j \] (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

\[ I = IV \]
\[ C_j = E_K (I + j - 1) \oplus P_j \text{ for } j = 1, 2, \ldots, n \] (2.56)
and decryption as

\[ I = IV \]
\[ P_j = E_K(I + j - 1) \oplus C_j \text{ for } j = 1, 2, \ldots, n. \]  \hspace{1cm} (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$^2$NMNT. 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

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

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

$$\beta(n) = (\beta_1(n) + \beta_2(n))_{M_p} \quad (3.2)$$

where

$$\beta_1(n) = \langle \text{Re}(\alpha_1 + j\alpha_2)^n \rangle_{M_p}, \quad \beta_2(n) = \langle \text{Im}(\alpha_1 + j\alpha_2)^n \rangle_{M_p} \quad (3.3)$$

for

$$\alpha_1 = \pm \langle 2^q \rangle_{M_p}, \quad \alpha_2 = \pm \langle -3^q \rangle_{M_p} \quad \text{and} \quad q = 2^p - 2 \quad (3.4)$$

and $\text{Re}(\cdot)$ and $\text{Im}(\cdot)$ denote the real and imaginary parts of the enclosed terms respectively. For transform lengths equal to $\frac{N}{2}$, $\beta_1$ and $\beta_2$ can be calculated as

$$\beta_1(n) = \langle \text{Re} \left[ (\alpha_1 + j\alpha_2)^d \right]^n \rangle_{M_p} \quad (3.5)$$
### 3.2 The NMNT

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>82</td>
<td>111</td>
<td>82</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>124</td>
<td>126</td>
<td>45</td>
<td>16</td>
<td>45</td>
<td>126</td>
<td>124</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>111</td>
<td>1</td>
<td>0</td>
<td>126</td>
<td>16</td>
<td>126</td>
<td>0</td>
<td>1</td>
<td>111</td>
<td>1</td>
<td>0</td>
<td>126</td>
<td>16</td>
<td>126</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>82</td>
<td>0</td>
<td>45</td>
<td>126</td>
<td>3</td>
<td>111</td>
<td>3</td>
<td>126</td>
<td>45</td>
<td>0</td>
<td>82</td>
<td>1</td>
<td>124</td>
<td>16</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>126</td>
<td>126</td>
<td>1</td>
<td>1</td>
<td>126</td>
<td>126</td>
<td>1</td>
<td>1</td>
<td>126</td>
<td>126</td>
<td>1</td>
<td>1</td>
<td>126</td>
<td>126</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>16</td>
<td>3</td>
<td>1</td>
<td>45</td>
<td>0</td>
<td>82</td>
<td>126</td>
<td>124</td>
<td>111</td>
<td>124</td>
<td>126</td>
<td>82</td>
<td>0</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>126</td>
<td>111</td>
<td>126</td>
<td>0</td>
<td>1</td>
<td>16</td>
<td>1</td>
<td>0</td>
<td>126</td>
<td>111</td>
<td>126</td>
<td>0</td>
<td>1</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>124</td>
<td>0</td>
<td>3</td>
<td>126</td>
<td>82</td>
<td>16</td>
<td>82</td>
<td>126</td>
<td>3</td>
<td>0</td>
<td>124</td>
<td>1</td>
<td>45</td>
<td>111</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>126</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>126</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>45</td>
<td>111</td>
<td>45</td>
<td>1</td>
<td>124</td>
<td>0</td>
<td>3</td>
<td>126</td>
<td>82</td>
<td>16</td>
<td>82</td>
<td>126</td>
<td>3</td>
<td>0</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>16</td>
<td>1</td>
<td>0</td>
<td>126</td>
<td>111</td>
<td>126</td>
<td>0</td>
<td>1</td>
<td>16</td>
<td>1</td>
<td>0</td>
<td>126</td>
<td>111</td>
<td>126</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>45</td>
<td>0</td>
<td>82</td>
<td>126</td>
<td>124</td>
<td>111</td>
<td>124</td>
<td>126</td>
<td>82</td>
<td>0</td>
<td>45</td>
<td>1</td>
<td>3</td>
<td>16</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>126</td>
<td>126</td>
<td>1</td>
<td>1</td>
<td>126</td>
<td>126</td>
<td>1</td>
<td>1</td>
<td>126</td>
<td>126</td>
<td>1</td>
<td>1</td>
<td>126</td>
<td>126</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>124</td>
<td>16</td>
<td>124</td>
<td>1</td>
<td>82</td>
<td>0</td>
<td>45</td>
<td>126</td>
<td>3</td>
<td>111</td>
<td>3</td>
<td>126</td>
<td>45</td>
<td>0</td>
<td>82</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>126</td>
<td>16</td>
<td>126</td>
<td>0</td>
<td>1</td>
<td>111</td>
<td>1</td>
<td>0</td>
<td>126</td>
<td>16</td>
<td>126</td>
<td>0</td>
<td>1</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>124</td>
<td>126</td>
<td>45</td>
<td>16</td>
<td>45</td>
<td>126</td>
<td>124</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>82</td>
<td>111</td>
<td>82</td>
<td></td>
</tr>
</tbody>
</table>

and

$$\beta_2(n) = \left\langle \text{Im} \left[ (\alpha_1 + j\alpha_2)^d \right]^n \right\rangle_{M_p}$$

(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

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

(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

$$\langle G_N G_N N^{-1} \rangle_{M_p} = I$$

(3.8)

where $G_N$ 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 $G_N = G_N^T$ 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 $M_p = 127$. The NMNT, as well as being
### Table 3.2: The ONMNT Kernel for $N = 16$ and $M_p = 127$

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>72</td>
<td>19</td>
<td>19</td>
<td>72</td>
<td>5</td>
<td>99</td>
<td>28</td>
<td>122</td>
<td>55</td>
<td>108</td>
<td>108</td>
<td>55</td>
<td>122</td>
<td>28</td>
<td>99</td>
<td>5</td>
</tr>
<tr>
<td>82</td>
<td>82</td>
<td>3</td>
<td>124</td>
<td>45</td>
<td>45</td>
<td>124</td>
<td>3</td>
<td>82</td>
<td>82</td>
<td>3</td>
<td>124</td>
<td>45</td>
<td>45</td>
<td>124</td>
<td>3</td>
</tr>
<tr>
<td>19</td>
<td>5</td>
<td>122</td>
<td>108</td>
<td>28</td>
<td>72</td>
<td>72</td>
<td>28</td>
<td>108</td>
<td>122</td>
<td>5</td>
<td>19</td>
<td>99</td>
<td>55</td>
<td>55</td>
<td>99</td>
</tr>
<tr>
<td>111</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>111</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>111</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>111</td>
<td>0</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>122</td>
<td>122</td>
<td>19</td>
<td>28</td>
<td>55</td>
<td>72</td>
<td>99</td>
<td>108</td>
<td>5</td>
<td>5</td>
<td>108</td>
<td>99</td>
<td>72</td>
<td>55</td>
<td>28</td>
</tr>
<tr>
<td>82</td>
<td>45</td>
<td>3</td>
<td>3</td>
<td>45</td>
<td>82</td>
<td>124</td>
<td>45</td>
<td>3</td>
<td>3</td>
<td>45</td>
<td>82</td>
<td>124</td>
<td>45</td>
<td>3</td>
<td>45</td>
</tr>
<tr>
<td>72</td>
<td>108</td>
<td>19</td>
<td>55</td>
<td>5</td>
<td>28</td>
<td>28</td>
<td>5</td>
<td>55</td>
<td>19</td>
<td>108</td>
<td>72</td>
<td>122</td>
<td>99</td>
<td>99</td>
<td>122</td>
</tr>
<tr>
<td>1</td>
<td>126</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>126</td>
<td>1</td>
<td>126</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>28</td>
<td>5</td>
<td>55</td>
<td>19</td>
<td>108</td>
<td>72</td>
<td>122</td>
<td>99</td>
<td>99</td>
<td>122</td>
<td>72</td>
<td>108</td>
<td>19</td>
<td>55</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>45</td>
<td>82</td>
<td>124</td>
<td>124</td>
<td>82</td>
<td>45</td>
<td>3</td>
<td>3</td>
<td>45</td>
<td>82</td>
<td>124</td>
<td>124</td>
<td>82</td>
<td>45</td>
</tr>
<tr>
<td>99</td>
<td>72</td>
<td>55</td>
<td>28</td>
<td>19</td>
<td>122</td>
<td>122</td>
<td>19</td>
<td>28</td>
<td>55</td>
<td>72</td>
<td>99</td>
<td>108</td>
<td>5</td>
<td>5</td>
<td>108</td>
</tr>
<tr>
<td>0</td>
<td>111</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>111</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>111</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>111</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>28</td>
<td>72</td>
<td>72</td>
<td>28</td>
<td>108</td>
<td>122</td>
<td>5</td>
<td>19</td>
<td>99</td>
<td>55</td>
<td>55</td>
<td>99</td>
<td>19</td>
<td>5</td>
<td>122</td>
<td>108</td>
</tr>
<tr>
<td>124</td>
<td>3</td>
<td>82</td>
<td>82</td>
<td>3</td>
<td>124</td>
<td>45</td>
<td>45</td>
<td>124</td>
<td>3</td>
<td>82</td>
<td>82</td>
<td>3</td>
<td>124</td>
<td>45</td>
<td>45</td>
</tr>
<tr>
<td>122</td>
<td>28</td>
<td>99</td>
<td>5</td>
<td>72</td>
<td>19</td>
<td>19</td>
<td>72</td>
<td>5</td>
<td>99</td>
<td>28</td>
<td>122</td>
<td>55</td>
<td>108</td>
<td>108</td>
<td>55</td>
</tr>
</tbody>
</table>

Orthogonal is also symmetric, producing a kernel that looks like

\[
M = \begin{bmatrix}
\beta(0) & \beta(0) & \beta(0) & \beta(0) & \ldots & \beta(0) \\
\beta(0) & \beta(1) & \beta(2) & \beta(3) & \ldots & \beta(N-1) \\
\beta(0) & \beta(2) & \beta(4) & \beta(6) & \ldots & \beta[2(N-1)] \\
\beta(0) & \beta(3) & \beta(6) & \beta(9) & \ldots & \beta[3(N-1)] \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
\beta(0) & \beta(N-1) & \beta[2(N-1)] & \beta[3(N-1)] & \ldots & \beta[(N-1)(N-1)] 
\end{bmatrix}
\] . \ (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

\[
X_O(k) = \left\langle \sum_{n=0}^{N-1} x(n) \beta \left( \frac{2k+1}{2} n \right) \right\rangle_{M_p} \quad \text{for } k = 0, 1, 2, \ldots, N - 1. \quad (3.10)
\]
The kernel that is produced looks like

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

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

\[
x(n) = \left\langle N^{-1} \sum_{n=0}^{N-1} X_O(k) \beta\left(\frac{2n+1}{2}k\right) \right\rangle_{M_p}
\]

for \(n = 0, 1, 2, \ldots, N - 1\).
3. THE GENERALISED NEW MERSENNE TRANSFORM

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

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

(3.13)

where \( G_O \) is an ONMNT transform matrix of length \( N \), \( G_O^T \) 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

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

and the inverse is defined as

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

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 \( p = 7 \).
3.4 Odd-Squared-NMNT (O^2NMNT)

Table 3.4: The O^2NMNT Kernel for N = 16 and Mp = 127

<table>
<thead>
<tr>
<th></th>
<th>15</th>
<th>56</th>
<th>106</th>
<th>105</th>
<th>105</th>
<th>106</th>
<th>56</th>
<th>15</th>
<th>83</th>
<th>26</th>
<th>14</th>
<th>36</th>
<th>91</th>
<th>113</th>
<th>101</th>
<th>44</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>105</td>
<td>15</td>
<td>14</td>
<td>113</td>
<td>112</td>
<td>22</td>
<td>71</td>
<td>101</td>
<td>36</td>
<td>83</td>
<td>106</td>
<td>106</td>
<td>83</td>
<td>36</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>15</td>
<td>91</td>
<td>71</td>
<td>71</td>
<td>91</td>
<td>15</td>
<td>106</td>
<td>14</td>
<td>44</td>
<td>22</td>
<td>101</td>
<td>26</td>
<td>105</td>
<td>83</td>
<td>113</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>14</td>
<td>71</td>
<td>44</td>
<td>83</td>
<td>56</td>
<td>113</td>
<td>22</td>
<td>91</td>
<td>106</td>
<td>26</td>
<td>112</td>
<td>112</td>
<td>26</td>
<td>106</td>
<td>91</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>113</td>
<td>71</td>
<td>83</td>
<td>83</td>
<td>71</td>
<td>113</td>
<td>105</td>
<td>91</td>
<td>21</td>
<td>26</td>
<td>15</td>
<td>112</td>
<td>101</td>
<td>106</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>112</td>
<td>91</td>
<td>56</td>
<td>71</td>
<td>36</td>
<td>15</td>
<td>21</td>
<td>14</td>
<td>83</td>
<td>22</td>
<td>26</td>
<td>26</td>
<td>22</td>
<td>83</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>22</td>
<td>15</td>
<td>113</td>
<td>113</td>
<td>15</td>
<td>22</td>
<td>56</td>
<td>101</td>
<td>91</td>
<td>83</td>
<td>21</td>
<td>106</td>
<td>44</td>
<td>36</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>71</td>
<td>106</td>
<td>22</td>
<td>105</td>
<td>21</td>
<td>56</td>
<td>112</td>
<td>83</td>
<td>101</td>
<td>14</td>
<td>91</td>
<td>91</td>
<td>14</td>
<td>101</td>
<td>83</td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>101</td>
<td>14</td>
<td>91</td>
<td>91</td>
<td>14</td>
<td>101</td>
<td>83</td>
<td>112</td>
<td>56</td>
<td>21</td>
<td>105</td>
<td>22</td>
<td>106</td>
<td>71</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>26</td>
<td>36</td>
<td>44</td>
<td>106</td>
<td>21</td>
<td>83</td>
<td>91</td>
<td>101</td>
<td>56</td>
<td>22</td>
<td>15</td>
<td>113</td>
<td>113</td>
<td>15</td>
<td>22</td>
<td>56</td>
</tr>
<tr>
<td>14</td>
<td>83</td>
<td>22</td>
<td>26</td>
<td>26</td>
<td>22</td>
<td>83</td>
<td>14</td>
<td>21</td>
<td>15</td>
<td>36</td>
<td>71</td>
<td>56</td>
<td>91</td>
<td>112</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>106</td>
<td>101</td>
<td>112</td>
<td>15</td>
<td>26</td>
<td>21</td>
<td>91</td>
<td>105</td>
<td>113</td>
<td>71</td>
<td>83</td>
<td>83</td>
<td>71</td>
<td>113</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>106</td>
<td>26</td>
<td>112</td>
<td>112</td>
<td>26</td>
<td>106</td>
<td>91</td>
<td>22</td>
<td>113</td>
<td>56</td>
<td>83</td>
<td>44</td>
<td>71</td>
<td>14</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>83</td>
<td>105</td>
<td>26</td>
<td>101</td>
<td>22</td>
<td>44</td>
<td>14</td>
<td>106</td>
<td>15</td>
<td>91</td>
<td>71</td>
<td>71</td>
<td>91</td>
<td>15</td>
<td>106</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>36</td>
<td>83</td>
<td>106</td>
<td>106</td>
<td>83</td>
<td>36</td>
<td>101</td>
<td>71</td>
<td>22</td>
<td>112</td>
<td>113</td>
<td>14</td>
<td>15</td>
<td>105</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>101</td>
<td>113</td>
<td>91</td>
<td>36</td>
<td>14</td>
<td>26</td>
<td>83</td>
<td>15</td>
<td>56</td>
<td>106</td>
<td>105</td>
<td>105</td>
<td>106</td>
<td>56</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.5: Values of α₁ and α₂ According to p for NMNT

<table>
<thead>
<tr>
<th>p</th>
<th>α₁</th>
<th>α₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>88</td>
</tr>
<tr>
<td>13</td>
<td>128</td>
<td>181</td>
</tr>
<tr>
<td>17</td>
<td>512</td>
<td>87260</td>
</tr>
<tr>
<td>19</td>
<td>1024</td>
<td>385302</td>
</tr>
<tr>
<td>31</td>
<td>65536</td>
<td>1268011823</td>
</tr>
</tbody>
</table>

As the O^2NMNT is symmetrical, like the NMNT, the forward and inverse kernels are derived as

\[
M = \begin{bmatrix}
\beta \left( \frac{1}{4} \right) & \beta \left( \frac{3}{4} \right) & \beta \left( \frac{5}{4} \right) & \beta \left( \frac{7}{4} \right) & \ldots & \beta \left( \frac{2N-1}{4} \right) \\
\beta \left( \frac{3}{4} \right) & \beta \left( \frac{9}{4} \right) & \beta \left( \frac{15}{4} \right) & \beta \left( \frac{21}{4} \right) & \ldots & \beta \left( \frac{3(2N-1)}{4} \right) \\
\beta \left( \frac{5}{4} \right) & \beta \left( \frac{15}{4} \right) & \beta \left( \frac{25}{4} \right) & \beta \left( \frac{35}{4} \right) & \ldots & \beta \left( \frac{5(2N-1)}{4} \right) \\
\beta \left( \frac{7}{4} \right) & \beta \left( \frac{21}{4} \right) & \beta \left( \frac{35}{4} \right) & \beta \left( \frac{49}{4} \right) & \ldots & \beta \left( \frac{7(2N-1)}{4} \right) \\
\beta \left( \frac{9}{4} \right) & \beta \left( \frac{27}{4} \right) & \beta \left( \frac{45}{4} \right) & \beta \left( \frac{63}{4} \right) & \ldots & \beta \left( \frac{9(2N-1)}{4} \right) \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
\beta \left( \frac{2N-1}{4} \right) & \beta \left[ \frac{3(2N-1)}{4} \right] & \beta \left[ \frac{5(2N-1)}{4} \right] & \beta \left[ \frac{7(2N-1)}{4} \right] & \ldots & \beta \left[ \frac{(2N-1)(2N-1)}{4} \right]
\end{bmatrix}
\]
### 3. THE GENERALISED NEW MERSENNE TRANSFORM

<table>
<thead>
<tr>
<th>N</th>
<th>$\alpha_1$</th>
<th>$\alpha_2$</th>
<th>$\alpha_3$</th>
<th>$\alpha_4$</th>
<th>$\alpha_5$</th>
<th>$\alpha_6$</th>
<th>$\alpha_7$</th>
<th>$\alpha_8$</th>
<th>$\alpha_9$</th>
<th>$\alpha_{10}$</th>
<th>$\alpha_{11}$</th>
<th>$\alpha_{12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>61</td>
<td>-1</td>
<td>0</td>
<td>-7</td>
<td>30</td>
<td>-1</td>
<td>0</td>
<td>-31</td>
<td>126</td>
<td>-1</td>
<td>0</td>
<td>-127</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>-7</td>
<td>6</td>
<td>-1</td>
<td>0</td>
<td>-31</td>
<td>30</td>
<td>-1</td>
<td>0</td>
<td>-127</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>-2</td>
<td>2</td>
<td>-5</td>
<td>27</td>
<td>-4</td>
<td>2</td>
<td>-27</td>
<td>119</td>
<td>-8</td>
<td>11</td>
<td>-8</td>
</tr>
<tr>
<td>16</td>
<td>-1</td>
<td>18</td>
<td>-133</td>
<td>7</td>
<td>-24</td>
<td>106</td>
<td>-21</td>
<td>103</td>
<td>-24</td>
<td>6456</td>
<td>-1735</td>
<td>7379</td>
</tr>
<tr>
<td>32</td>
<td>5</td>
<td>-26</td>
<td>10</td>
<td>-21</td>
<td>102</td>
<td>-25</td>
<td>97</td>
<td>-30</td>
<td>5114</td>
<td>-3077</td>
<td>674</td>
<td>-5712</td>
</tr>
<tr>
<td>64</td>
<td>-1</td>
<td>49</td>
<td>-78</td>
<td>93</td>
<td>-34</td>
<td>2089</td>
<td>-5693</td>
<td>3389</td>
<td>-8202</td>
<td>69417</td>
<td>-61621</td>
<td>29296</td>
</tr>
<tr>
<td>128</td>
<td>5</td>
<td>-122</td>
<td>22</td>
<td>-105</td>
<td>4634</td>
<td>-3557</td>
<td>2338</td>
<td>-5853</td>
<td>77543</td>
<td>-53528</td>
<td>19726</td>
<td>-111315</td>
</tr>
<tr>
<td>256</td>
<td>-1</td>
<td>336</td>
<td>-7855</td>
<td>1198</td>
<td>-9993</td>
<td>94715</td>
<td>-30356</td>
<td>57903</td>
<td>-73168</td>
<td>419100</td>
<td>-85187</td>
<td>63508</td>
</tr>
<tr>
<td>512</td>
<td>-1</td>
<td>5152</td>
<td>-3039</td>
<td>4112</td>
<td>-3779</td>
<td>90069</td>
<td>-41002</td>
<td>83147</td>
<td>-47924</td>
<td>431498</td>
<td>-97289</td>
<td>331707</td>
</tr>
<tr>
<td>1024</td>
<td>-1</td>
<td>2533</td>
<td>-7088</td>
<td>2016</td>
<td>-6175</td>
<td>94892</td>
<td>-36179</td>
<td>42772</td>
<td>-88299</td>
<td>141999</td>
<td>-383278</td>
<td>49067</td>
</tr>
<tr>
<td>2048</td>
<td>-1</td>
<td>4801</td>
<td>-3390</td>
<td>6567</td>
<td>-1624</td>
<td>125429</td>
<td>-59642</td>
<td>41255</td>
<td>-89186</td>
<td>156575</td>
<td>-367712</td>
<td>75021</td>
</tr>
<tr>
<td>4096</td>
<td>-1</td>
<td>49</td>
<td>-8142</td>
<td>4664</td>
<td>-3527</td>
<td>72359</td>
<td>-58712</td>
<td>108667</td>
<td>-22404</td>
<td>206207</td>
<td>-318809</td>
<td>41687</td>
</tr>
<tr>
<td>8192</td>
<td>-1</td>
<td>5</td>
<td>-8186</td>
<td>5381</td>
<td>-2810</td>
<td>20129</td>
<td>-110942</td>
<td>45131</td>
<td>-85940</td>
<td>342993</td>
<td>-181704</td>
<td>291292</td>
</tr>
<tr>
<td>16384</td>
<td>-1</td>
<td>90280</td>
<td>-37701</td>
<td>36519</td>
<td>-94352</td>
<td>100120</td>
<td>-421487</td>
<td>341961</td>
<td>-182236</td>
<td>1737836</td>
<td>-1140821</td>
<td>976265</td>
</tr>
<tr>
<td>32768</td>
<td>-1</td>
<td>4801</td>
<td>-120270</td>
<td>63818</td>
<td>-69190</td>
<td>204661</td>
<td>-319062</td>
<td>284290</td>
<td>-239707</td>
<td>1590419</td>
<td>-59160765</td>
<td>100657803</td>
</tr>
<tr>
<td>65536</td>
<td>-1</td>
<td>49</td>
<td>-1302022</td>
<td>31393</td>
<td>-99678</td>
<td>186232</td>
<td>-380535</td>
<td>463103</td>
<td>-66114</td>
<td>20112886</td>
<td>-136319983</td>
<td>108607642</td>
</tr>
<tr>
<td>131072</td>
<td>-1</td>
<td>5</td>
<td>-131006</td>
<td>94869</td>
<td>-36182</td>
<td>4801</td>
<td>-519490</td>
<td>16811</td>
<td>-507476</td>
<td>453787664</td>
<td>-1091808983</td>
<td>555295815</td>
</tr>
<tr>
<td>262144</td>
<td>-1</td>
<td>49</td>
<td>-524238</td>
<td>46508</td>
<td>-56677</td>
<td>758473271</td>
<td>-1388089026</td>
<td>160114785</td>
<td>-1987368862</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>524288</td>
<td>-1</td>
<td>5</td>
<td>-524282</td>
<td>46503</td>
<td>-477726</td>
<td>767347098</td>
<td>-183736509</td>
<td>129754064</td>
<td>-894070947</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 3.6: Values of $\alpha_1$ and $\alpha_2$ According to $p$ and $N$ for NMNT**
<table>
<thead>
<tr>
<th>N</th>
<th>p = 3</th>
<th>p = 5</th>
<th>p = 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>-101</td>
<td>-214</td>
<td>-428</td>
</tr>
<tr>
<td>8</td>
<td>-514</td>
<td>-1028</td>
<td>-2056</td>
</tr>
<tr>
<td>16</td>
<td>-2056</td>
<td>-4112</td>
<td>-8224</td>
</tr>
<tr>
<td>32</td>
<td>-8224</td>
<td>-16456</td>
<td>-32912</td>
</tr>
<tr>
<td>64</td>
<td>-32912</td>
<td>-65824</td>
<td>-131648</td>
</tr>
<tr>
<td>128</td>
<td>-131648</td>
<td>-263296</td>
<td>-526592</td>
</tr>
<tr>
<td>256</td>
<td>-526592</td>
<td>-1053184</td>
<td>-2106368</td>
</tr>
<tr>
<td>512</td>
<td>-2106368</td>
<td>-4212736</td>
<td>-8425472</td>
</tr>
<tr>
<td>1024</td>
<td>-8425472</td>
<td>-16850944</td>
<td>-33701888</td>
</tr>
<tr>
<td>2048</td>
<td>-33701888</td>
<td>-67403776</td>
<td>-134807552</td>
</tr>
<tr>
<td>4096</td>
<td>-134807552</td>
<td>-269615104</td>
<td>-539230208</td>
</tr>
<tr>
<td>8192</td>
<td>-539230208</td>
<td>-1078460416</td>
<td>-2156920832</td>
</tr>
<tr>
<td>16384</td>
<td>-2156920832</td>
<td>-4313841664</td>
<td>-8627683328</td>
</tr>
<tr>
<td>32768</td>
<td>-8627683328</td>
<td>-17255366656</td>
<td>-34510733392</td>
</tr>
</tbody>
</table>

3.4 Odd-Squared-NMNT (O²NMNT)
<table>
<thead>
<tr>
<th>N</th>
<th>$p = 3$</th>
<th>$p = 5$</th>
<th>$p = 7$</th>
<th>$p = 13$</th>
<th>$p = 17$</th>
<th>$p = 19$</th>
<th>$p = 31$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>-5</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>-5</td>
<td>-1</td>
<td>-5</td>
</tr>
<tr>
<td>4</td>
<td>-4</td>
<td>1</td>
<td>-4</td>
<td>-1</td>
<td>-4</td>
<td>1</td>
<td>-4</td>
</tr>
<tr>
<td>8</td>
<td>-4</td>
<td>-3</td>
<td>1</td>
<td>-3</td>
<td>1</td>
<td>-3</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>-1</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>32</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>64</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>128</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>256</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>512</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>1024</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>2048</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>4096</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>8192</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>16384</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>32768</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>65536</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>131072</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>262144</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>524288</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>1048576</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>2097152</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>4194304</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>8388608</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>16777216</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>33554432</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>67108864</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>134217728</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>268435456</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>536870912</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
</tr>
</tbody>
</table>

Table 3.8: Values of $\alpha_1$ and $\alpha_2$ According to $p$ and $N$ for $O^2$NMNT
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\textsuperscript{2}NMNT,
including consideration of their respective maximal transform lengths. For all
transforms of the GNMNT, the initial values of \(\alpha_1\) and \(\alpha_2\) 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 \(\alpha_1\) and \(\alpha_2\), they both require
adjustment according to the transform length to produce the initial values of \(\beta_1\)
and \(\beta_2\). 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 \(N/d\) for \(d\) is a power of two, which is \(2^{p+1}\) for NMNT, \(2^p\) for ONMNT and
\(2^{p-1}\) for O\textsuperscript{2}NMNT. A comprehensive list of final values of \(\alpha_1\) and \(\alpha_2\) according to
the field derived by \(p\) and the transform length \(N\) for the NMNT, ONMNT and
O\textsuperscript{2}NMNTs 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

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

and

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

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\textsuperscript{2}NMNT are selected then the transforms are also
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

\[
y_{LC}(n) = x(n) \ast h(n) = \sum_{k=0}^{N-1} x(k) h(n-k)
\]

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

\[
y_{CC}(n) = x(n) \ast h(n) = \sum_{k=0}^{N-1} x(n-k \mod N) h(k)
\]

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

\[
y_{SCC}(n) = x(n) \ast h(n) = \sum_{k=0}^{n} x(k) h(n-k) - \sum_{k=n+1}^{N-1} x(k) h(n-k + N).
\]

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

\[ G_N[y_{cc}(n)] = G_N[x(n) \ast h(n)] \]

\[
= \left\langle \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} x(n - m)h(m) \beta(nk) \right\rangle_{M_p} \\
= \left\langle \sum_{m=0}^{N-1} h(m) \sum_{s=0}^{N-1} x(s)\beta(sk + mk) \right\rangle_{M_p}
\]

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

\[
\left\langle \sum_{m=0}^{N-1} h(m) \sum_{s=0}^{N-1} x(s)\beta(sk) \right\rangle_{M_p} \\
+ \sum_{m=0}^{N-1} h(m) \beta_2(mk) \sum_{s=0}^{N-1} x(s)\beta(-sk) \right\rangle_{M_p}
\]

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

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

and

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

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

\[
\left\langle \sum_{m=0}^{N-1} h(m) \beta_1(mk) \sum_{s=0}^{N-1} x(s)\beta(sk) + \sum_{m=0}^{N-1} h(m) \beta_2(mk) \sum_{s=0}^{N-1} x(s)\beta(-sk) \right\rangle_{M_p} \\
= \left\langle \sum_{m=0}^{N-1} h(m) \left[ \frac{\beta(mk) + \beta(-mk)}{2} \right] X(k) \right\rangle_{M_p} \\
+ \sum_{m=0}^{N-1} h(m) \left[ \frac{\beta(mk) - \beta(-mk)}{2} \right] X(N-k) \right\rangle_{M_p}
\]

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

\[
\left\langle \sum_{s=0}^{N-1} x(s)\beta(sk) \right\rangle_{M_p} = X(k),
\]
3. THE GENERALISED NEW MERSENNE TRANSFORM

\[
\langle \sum_{s=0}^{N-1} x(s) \beta (-sk) \rangle_{M_p} = X (-k) = X (N - k)
\]  

(3.29)

and

\[
\langle \sum_{m=0}^{N-1} h (m) \beta (mk) \rangle_{M_p} = H (k),
\]  

(3.30)

\[
\langle \sum_{m=0}^{N-1} h (m) \beta (-mk) \rangle_{M_p} = H (-k) = H (N - k).
\]  

(3.31)

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

\[
\langle H (k) + H (-k) \rangle_{M_p} X (k) + \langle H (k) - H (-k) \rangle_{M_p} X (N - k) \rangle_{M_p}
\]  

(3.32)

\[
= \left\{ [H (k) + H (N - k)] X (k) + [H (k) - H (N - k)] X (N - k) \right\} 2^{p-1}
\]  

= \langle \{
\]

(3.36)

where

\[
\frac{1}{2} = \langle 2^{p-1} \rangle_{M_p}.
\]  

(3.33)

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

\[
H^{ev}(k) = \langle \frac{H (k) + H (-k)}{2} \rangle_{M_p}
\]  

(3.34)

\[
= \langle \{ [H (k) + H (-k)] 2^{p-1} \} \rangle_{M_p}
\]

and

\[
H^{od}(k) = \langle \frac{H (k) - H (-k)}{2} \rangle_{M_p}
\]  

(3.35)

\[
= \langle \{ [H (k) - H (-k)] 2^{p-1} \} \rangle_{M_p}
\]

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

\[
\mathcal{G}_N [y_{cc}(n)] = \mathcal{G}_N [x(n) \oplus h(n)]
\]  

(3.36)

\[
= \langle H^{ev}(k) \rangle_{M_p} X (k) + \langle H^{od}(k) \rangle_{M_p} X (N - k)
\]  

\[
= Y (k).
\]
This is shown in Figure 3.1, where the $\odot$ operator is defined as

$$Y(k) = X(k) \odot H(k)$$

$$= H^{ev}(k)X(k) + H^{od}(k)X(N-k).$$ (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

$$x(n) = [1, 1, 0, 1, 1, 0, 1, 1]$$ (3.38)

and

$$h(n) = [1, 1, 0, 1, 1, 0, 1, 0].$$ (3.39)

The resulting NMNT representations are

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

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

Applying the cyclic convolution algorithm produces

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

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

$$y_{CC} = [1, 2, 1, 2, 4, 2, 2, 4, 2, 1, 2, 1, 0, 1, 0].$$ (3.43)
Confirming the results with the built in function of MATLAB produces

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

\[
\mathcal{G}_O[y_{cc}(n)] = \mathcal{G}_O[x(n) \circledast h(n)] \\
= \left\langle \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} x(n-m) h(m) \beta \left( \frac{2k+1}{2} \right) \right\rangle_{M_p} \\
= \left\langle \sum_{m=0}^{N-1} h(m) \sum_{s=0}^{N-1} x(s) \beta \left( \frac{2k+1}{2} + \frac{2k+1}{2} m \right) \right\rangle_{M_p}
\] (3.45)

where \( s = m - n \), and \( \mathcal{G}_O \) denotes the ONMNT transform. From the \( \beta \) 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

\[
\left\langle \sum_{m=0}^{N-1} h(m) \sum_{s=0}^{N-1} x(s) \left[ \beta_1(mk) \beta \left( \frac{2k+1}{2} s \right) + \beta_2(mk) \beta \left( -\frac{2k+1}{2} s \right) \right] \right\rangle_{M_p}
\] (3.46)

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

\[
\left\langle \sum_{m=0}^{N-1} h(m) \sum_{s=0}^{N-1} x(s) \left[ \beta_1(mk) \beta \left( \frac{2k+1}{2} s \right) + \beta_2(mk) \beta \left( -\frac{2k+1}{2} s \right) \right] \right\rangle_{M_p}
\] (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

$$
\left< \sum_{s=0}^{N-1} x(s) \beta \left( \frac{2k+1}{2} s \right) \right>_{Mp} = X_O(k),
$$

(3.48)

$$
\left< \sum_{s=0}^{N-1} x(s) \beta \left( -\frac{2k+1}{2} s \right) \right>_{Mp} = X_O(-k-1)
$$

(3.49)

and

$$
\left< \sum_{m=0}^{N-1} h(m) \beta \left( \frac{2k+1}{2} m \right) \right>_{Mp} = H_O(k),
$$

(3.50)

$$
\left< \sum_{m=0}^{N-1} h(m) \beta \left( -\frac{2k+1}{2} m \right) \right>_{Mp} = H_O(-k-1)
$$

(3.51)

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

$$
\left< \sum_{m=0}^{N-1} h(m) \beta \left( \frac{2k+1}{2} m \right) \right>_{Mp} = H_O(k)
$$

(3.52)

$$
\left< \sum_{m=0}^{N-1} h(m) \beta \left( -\frac{2k+1}{2} m \right) \right>_{Mp} = H_O(N-k-1)
$$

(3.53)

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

$$
H_O^{ev}(k) = \left< \frac{H_O(k) + H_O(-k)}{2} \right>_{Mp}
$$

(3.53)
3. THE GENERALISED NEW MERSENNE TRANSFORM

Figure 3.2: Convolution Process Structure for the ONMNT

and

$$H_O^{od}(k) = \left\{ \frac{H_O (k) - H_O (-k)}{2} \right\}_{M_p}$$

(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

$$g_O [y_{cc}(n)] = g_O [x (n) \odot h (n)]$$

$$= H_O^{ev}(k) X (k) + H_O^{od}(k) X (N - k)$$

(3.55)

$$= Y_O (k).$$

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

$$Y_O (k) = X_O(k) \odot H_O(k)$$

(3.56)

$$= H_O^{ev}(k) X_O (k) + H_O^{od}(k) X_O (N - k).$$

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

$$x(n) = [1, 1, 1, 1, 0, 0, 0, 0],$$

(3.57)

$$h(n) = [0, 0, 0, 1, 1, 1, 1].$$

(3.58)

The resulting ONMNT representations are

$$X_O(k) = [2309, 5366, 5554, 2448, 5584, 3903, 2196, 3490, 790, 7921, 4485, 7591, 7703, 7387, 4151, 2857],$$

(3.59)
3.6 Cyclic Convolution of the GNMNT

and

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

Applying the circular convolution algorithm produces

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

and applying the inverse ONMNT results in the first \(2N - 1\) elements as

\[ ycc = [0, 0, 0, 1, 2, 3, 4, 3, 2, 1, 0, 0, 0]. \] (3.62)

Confirming the results with the built in function of MATLAB produces

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

\[
Y_{O^2}(k) = \left\langle \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} x(n-m)h(m) \beta \left[ \left( \frac{2k+1}{2} \right) \left( \frac{2n+1}{2} \right) \right] \right\rangle_{MP} \\
= \left\langle \sum_{m=0}^{N-1} h(m) \sum_{s=0}^{N-1} x(s) \beta \left[ \left( \frac{2k+1}{2} \right) \left( \frac{2n+1}{2} \right) \right] \right\rangle_{MP} 
\] (3.64)

where

\[ s = m - n. \] (3.65)

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

\[
\left\langle \sum_{m=0}^{N-1} h(m) \beta \left[ \left( \frac{2k+1}{2} \right) \left( \frac{2m+1}{2} \right) \right] \right\rangle_{MP} = H_{O^2}(k) 
\] (3.66)
3. THE GENERALISED NEW MERSENNE TRANSFORM

and

$$\left\langle \sum_{m=0}^{N-1} h(m) \beta \left[ \left( -\frac{2k+1}{2} \right) \left( \frac{2m+1}{2} \right) \right] \right\rangle_{M_p} = H_O (-k - 1)$$

$$= -H_O^2 (N - k - 1).$$

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

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

and (3.68) into (3.64) produces

$$Y_O^2 (k) = \left\langle \sum_{m=0}^{N-1} h(m) \sum_{s=0}^{N-1} x(s) \beta \left[ \left( \frac{2k+1}{2} \right) \left( \frac{2m+1}{2} + \frac{2k+1}{2} s \right) \right] \right\rangle_{M_p}$$

$$= \left\langle \sum_{m=0}^{N-1} h(m) \sum_{s=0}^{N-1} x(s) \left\{ \beta_1 \left[ \left( \frac{2k+1}{2} \right) \left( \frac{2m+1}{2} \right) \right] \beta \left( \frac{2k+1}{2} s \right) \right. \right.$$

$$+ \left. \beta_2 \left[ \left( \frac{2k+1}{2} \right) \left( \frac{2m+1}{2} \right) \right] \beta \left( -\frac{2k+1}{2} s \right) \right\} \right\rangle_{M_p}$$

$$= \left\langle \sum_{m=0}^{N-1} h(m) \beta_1 \left[ \left( \frac{2k+1}{2} \right) \left( \frac{2m+1}{2} \right) \right] \sum_{s=0}^{N-1} x(s) \beta \left( \frac{2k+1}{2} s \right) \right.$$

$$+ \left. \sum_{m=0}^{N-1} h(m) \beta_2 \left[ \left( \frac{2k+1}{2} \right) \left( \frac{2m+1}{2} \right) \right] \sum_{s=0}^{N-1} x(s) \beta \left( -\frac{2k+1}{2} s \right) \right\rangle_{M_p}.$$}

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

$$Y_O^2 (k) = \left\langle \frac{H_O^2 (k) - H_O^2 (N - K - 1)}{2} X_O (k) \right.$$

$$+ \left. \frac{H_O^2 (k) + H_O^2 (N - k - 1)}{2} X_O (N - k - 1) \right\rangle_{M_p}$$

$$= \left\langle \{ [H_O^2 (k) - H_O^2 (N - k - 1)] X_O (k) \right.$$

$$+ \left. [H_O^2 (k) + H_O^2 (N - k - 1)] X_O (N - k - 1) \} 2^{p-1} \right\rangle_{M_p}.$$
3.6 Cyclic Convolution of the GNMNT

![Diagram](image)

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

where

$$H_{O^2}^{ev}(k) = \langle [H_{O^2}(k) + H_{O^2}(-k)]2^{p-1} \rangle_{M_p}$$  \hspace{1cm} (3.72)

and

$$H_{O^2}^{od}(k) = \langle [H_{O^2}(k) - H_{O^2}(-k)]2^{p-1} \rangle_{M_p}.$$  \hspace{1cm} (3.73)

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

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

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

The resulting O$^2$NMNT representations are

$$X_O(k) = [3454, 6272, 3545, 18, 6333, 3215, 6685, 7957, 1642, 7015, 2802, 1830, 4957, 8075, 3354, 6581],$$  \hspace{1cm} (3.76)

and

$$H_{O^2}(k) = [1973, 5938, 6361, 2277, 1222, 606, 3763, 4807, 7005, 5958, 225, 7841, 2599, 4844, 464, 5500].$$  \hspace{1cm} (3.77)

Applying the circular convolution algorithm produces

$$Y_{O^2}(k) = [7595, 4072, 3529, 543, 963, 3337, 7814, 2753, 4813, 6359, 3429, 4057, 6479, 7859, 2744, 8088],$$  \hspace{1cm} (3.78)
3. THE GENERALISED NEW MERSENNE TRANSFORM

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

$$y_{cc} = [0, 1, 0, 0, 1, 0, 2, 0, 1, 1, 2, 1, 0].$$  \hfill (3.79)

Confirming the results with the built in function of MATLAB produces

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

$$T_{G} = \mathfrak{S}(t) \hfill (3.81)$$

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

$$K = \{0x3e41, 0x1a4d, 0x273f, 0xf670, 0x39f4, 0xea3e, 0xddc1, 0x99a0\} \hfill (3.82)$$

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

$$\hat{K} = \{0x9ef7, 0x2913, 0x7503, 0xf2cc, 0xea3e, 0xd934, 0xaab1, 0x2903\}. \hfill (3.83)$$
3.7 Encryption Example using the GNMNT

The cipher is then created by

\[ C_g = \langle T_g \otimes K \rangle_{M_p} \tag{3.84} \]

and

\[ c = \mathcal{G}(C_g)^{-1} \tag{3.85} \]

where \( \mathcal{G}(\cdot)^{-1} \) 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

\[ K^{-1} = \langle K^{M_p-2} \rangle_{M_p} \tag{3.86} \]
\[ \hat{K}^{-1} = \langle \hat{K}^{M_p-2} \rangle_{M_p} \tag{3.87} \]

The unadulterated key from (3.86) is the used to decrypt \( c \) to recover \( r \), which represents the original plaintext \( t \) by first deriving \( R_g \) as

\[ R_g = \langle C_g \otimes K^{-1} \rangle_{M_p} \tag{3.88} \]

and then

\[ r = \mathcal{G}(R_g)^{-1} \tag{3.89} \]

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

\[ \hat{R}_g = \langle C_g \otimes \hat{K}^{-1} \rangle_{M_p} \tag{3.91} \]

and then

\[ \hat{r} = \mathcal{G}(\hat{R}_g)^{-1}. \tag{3.92} \]

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. THE GENERALISED NEW MERSENNE TRANSFORM

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

\[
t = \{0xce, 0xd0, 0xd1, 0xcf, 0xd1, 0xcf, 0xd1, 0xd0, 0xcf, 0xd0, 0xd1, 0xd3, 0xd1, 0xd0, 0xd2\}
\]

and concatenating subsequent pairs of bytes so that

\[
t = \{0xced0, 0xd1cf, 0xd1d1, 0xcfd1, 0xd0cf, 0xd0d1, 0xd3d1, 0xd0d2\}.
\]

Applying the NMNT to the vector \( t \) we obtain

\[
T = \{0x08887, 0x00202, 0x1f7ff, 0x00000, 0x001fe, 0x1fdff, 0x1fbf9, 0x1f801\}
\]

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

\[
C = \{0x16Ce0, 0x0ceb4, 0x00763, 0x00000, 0x17451, 0x18315, 0x1c7bb, 0x030da\}.
\]

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

\[
c = \{0x1665e, 0x02da3, 0x0de24, 0x09f5d, 0x145b5, 0x1c096, 0x06eb0\}.
\]

Reverting \( c \) to the GNMNT domain \( C \) allows recovery of the plaintext \( r \), which can be obtained by using the inverse key (3.86) with \( C \) 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 \( C \) in (3.91) to respectively produce Figures 3.5(a)-3.5(d) for the cameraman image.

\[
\hat{R} = \{0x02c52, 0x0dcca, 0x06606, 0x00000, 0x149ec, 0x13a3a, 0x12ac2, 0x191e3\}
\]
3.7 Encryption Example using the GNMNT

Figure 3.4: Encrypted Cameraman using NMNT, \( N = 8 \) and \( M_p = 131071 \)

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.
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 O²NMNT

Like the ONMNT, the application of the O²NMNT within the field of encryption is very recent. However, there is a stronger interest in O²NMNT as characteristics of this transform appear to show that it is extremely resilient according to \[119\]. The
3.7 Encryption Example 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.
3. THE GENERALISED NEW MERSENNE TRANSFORM

(a) Original

(b) NMNT

(c) ONMNT

(d) O\textsuperscript{2}NMNT

Figure 3.7: Decrypted Cameraman Error using GNMNT, \( N = 8 \) and \( M_p = 131071 \)

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.
3.8 The GNMNT Kernel Components

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 equivalent of minus one in quantities that can be determined by

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

\[
\# \{(\mathcal{G}_N, m(0))\} = N \lfloor \log_2(N) - 2 \rfloor \tag{3.101}
\]
3. THE GENERALISED NEW MERSENNE TRANSFORM

Figure 3.9: Consistency of Trivial Elements in the ONMNT Kernel

and

$$\#(g_N, m(-1)) = N \lfloor \log_2(N) - 0.5 \rfloor,$$

(3.102)

where $g_N$ 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

$$K = \{0x3e41, 0x1a4d, 0x273f, 0xf670, 0x39f4, 0xea3e, 0xddc1, 0x99a1\}.$$

(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 shown 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

\[ \# \{ (G_O, m(1)) \} = \frac{3N}{2} \]  

(3.104)

\[ \# \{ (G_O, m(0)) \} = N \]  

(3.105)

and

\[ \# \{ (G_N, m(-1)) \} = \frac{N}{2} \].  

(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 shown 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$^2$NMNT Kernel 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
Figure 3.10: Decrypted $256 \times 256$ Cameraman Bit Error using NMNT with $N = 8, 16, 32, 64, 128$, $Mp = 131071$
Figure 3.11: Decrypted $256 \times 256$ Cameraman Bit Error using ONMNT with $N = 8, 16, 32, 64, 128$, $Mp = 131071$
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 O²NMNT 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

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

(3.107)

and the NMNT kernel as

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

(3.108)

where

\[
0 \leq i, j \leq N - 1
\]  

(3.109)

then by applying

\[
X = \langle xM \rangle_{M^p}
\]  

(3.110)

where the transformed vector \( X \) is similarly constructed as

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

(3.111)

the derivation of each element of \( X \) as a result from applying the NMNT to \( x \) can
be shown as

\[ X_0 = x_0 \beta (nk) + x_1 \beta [(n + 1)k] \]
\[ + x_i \beta [(n + i)k] + \ldots + x_{N-n-1} \beta [(N - n - 1)k] \]
\[ X_1 = x_0 \beta [n (k + 1)] + x_1 \beta [(n + 1)(k + 1)] \]
\[ + x_i \beta [(n + i)(k + 1)] + \ldots + x_{N-n-1} \beta [(N - n - 1)(k + 1)] \]
\[ X_j = x_0 \beta [n (j + 1)] + x_1 \beta [(n + 1)(j + 1)] \]
\[ + x_i \beta [(n + i)(j + 1)] + \ldots + x_{N-n-1} \beta [(N - n - 1)(j + 1)] \]
\[ \vdots \]
\[ X_{N-k-1} = x_0 \beta [n (N - k - 1)] + x_1 \beta [(n + 1)(N - k - 1)] \]
\[ + x_i \beta [(n + i)(N - k - 1)] + \ldots + x_{N-n-1} \beta [(N - n - 1)(N - k - 1)] \].

(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

\[ \hat{X} = \langle \hat{x}M \rangle_{M_p} \]  

(3.113)

and can show that

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

(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

\[ y = \left\langle \frac{1}{N} \left[ Y M' \right] \right\rangle_{M_p} \]  

(3.115)

noting that

\[ y = x \]  

(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

$$\hat{y} = \left\langle \frac{1}{N} [\hat{Y} M'] \right\rangle_{M_p}$$  \hspace{1cm} (3.117)

we can show that

$$\hat{y}_j = y_j \text{ when } \hat{Y}_i \neq Y_i \iff \beta [(n+i) (k+j)] = 0,$$  \hspace{1cm} (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.  \quad \square

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 $\alpha_1$ and $\alpha_2$ values from Table 3.6

$$M = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 512 & 1 & 0131070 & 130559 & 131070 & 0 \\
1 & 1 & 131070 & 131070 & 1 & 1 & 131070 & 131070 \\
1 & 0 & 131070 & 512 & 131070 & 0 & 1 & 130559 \\
1 & 131070 & 1 & 131070 & 1 & 131070 & 1 & 131070 \\
1 & 130559 & 1 & 0 & 131070 & 512 & 131070 & 0 \\
1 & 131070 & 131070 & 1 & 1 & 131070 & 131070 & 1 \\
1 & 0 & 131070 & 130559 & 131070 & 0 & 1 & 512
\end{bmatrix}$$  \hspace{1cm} (3.119)

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

$$M_{i,j} = 0 \text{ for } i = N - 1, j \in \left\{ \frac{N}{8}, \frac{5N}{8} \right\} \text{ and } 8 \leq N \leq 2^p.$$  \hspace{1cm} (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 $\alpha_1$ and $\alpha_2$ values from
3. THE GENERALISED NEW MERSENNE TRANSFORM

Table 3.7

\[
M = \begin{bmatrix}
1 & 82137 & 512 & 82137 & 1 & 29287 & 0 & 101784 \\
1 & 82137 & 0 & 48934 & 131070 & 29287 & 512 & 29287 \\
1 & 29287 & 130559 & 29287 & 1 & 48934 & 0 & 82137 \\
1 & 101784 & 0 & 29287 & 131070 & 82137 & 130559 & 82137 \\
1 & 48934 & 512 & 48934 & 1 & 101784 & 0 & 29287 \\
1 & 48934 & 0 & 82137 & 131070 & 101784 & 512 & 101784 \\
1 & 101784 & 130559 & 101784 & 1 & 82137 & 0 & 48934 \\
1 & 29287 & 0 & 101784 & 131070 & 48934 & 130559 & 48934
\end{bmatrix}
\]

(3.121)

and this time noting that

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

(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

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

(3.123)

and vector

\[
x = [52, 92, 99, 46]
\]

(3.124)
so that

\[ X = \langle xM \rangle_{127} \]

\[ = [118, 51, 53, 66]. \] (3.125)

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

\[ \hat{x} = [52, 92, 99, 3] \] (3.126)

results in

\[ \hat{X} = \langle \hat{x}M \rangle_{127} \]

\[ = [48, 51, 21, 64] \] (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

\[ \hat{x} = [52, 3, 99, 46] \] (3.128)

results in

\[ \hat{X} = \langle [52, 3, 99, 46] M \rangle_{127} \]

\[ = [35, 77, 53, 87] \] (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. FAST ALGORITHMS OF THE GNMNT

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

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

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

\[
X^{ev}(k) = \left\langle \sum_{n=0}^{\frac{N}{2}-1} x(2n) \beta \left( \frac{2k + 1}{2} \right) \right\rangle_{M_p} = X_{2n}(k) \tag{4.3}
\]

and

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

Simplifying \( \beta \) 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

\[
\beta \left( \frac{2k + 1}{2} 2n + \frac{2k + 1}{2} \right) = \beta_1 \left( \frac{2k + 1}{2} \right) \beta \left( \frac{2k + 1}{2} 2n \right) + \beta_2 \left( \frac{2k + 1}{2} \right) \beta \left( -\frac{2k + 1}{2} 2n \right) . \tag{4.7}
\]

To obtain a general odd form we get

\[
X_{2n+1}(k) = \left\langle \sum_{n=0}^{\frac{N}{2}-1} x(2n + 1) \beta \left( \frac{2k + 1}{2} 2n \right) \right\rangle_{M_p} \tag{4.8}
\]
4.2 Radix-2 ONMNT

\[ X(k) \quad X(N/2-k) \quad X(k+N/2) \quad X(N-k) \]

\[ X(k) \quad X(N/2-k) \quad X(k+N/2) \quad X(N-k) \]

\[ \beta(2^{\frac{k}{2}} + 1) \quad \beta(2^{\frac{k}{2}} + 1) \quad \beta(2^{\frac{k}{2}} + 1) \quad \beta(2^{\frac{k}{2}} + 1) \]

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

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

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

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

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

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 \( \frac{N}{2} - k; \therefore \frac{N}{2} - k : \leftrightarrow \frac{N}{2} - k - 1 \) by definition, then

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

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

and

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

According to the \( \beta \) in each point derived in (4.12)-(4.14), the following points can be derived by the respective processes as
4. FAST ALGORITHMS OF THE GNMNT

\[ X_{2n+1} \left( \frac{N}{2} - k \right) = \left\langle \sum_{n=0}^{N-1} x(2n+1)\beta \left[ \frac{2 \left( \frac{N}{2} - k - 1 \right) + 1 + 2n}{2} \right] \right\rangle_{M_p} \]

\[ = \left\langle \sum_{n=0}^{N-1} x(2n+1)\beta \left( -\frac{2k+1}{2} - 2n \right) \right\rangle_{M_p} \]  \hspace{1cm} (4.15)

\[ = X_{2n+1} (N - k) \]

\[ X_{2n+1} (k + \frac{N}{2}) = \left\langle \sum_{n=0}^{N-1} x(2n+1)\beta \left[ \frac{2 \left( k + \frac{N}{2} \right) + 1 + 2n}{2} \right] \right\rangle_{M_p} \]

\[ = \left\langle \sum_{n=0}^{N-1} x(2n+1)\beta \left( \frac{2k+1}{2} - 2n \right) \right\rangle_{M_p} \]  \hspace{1cm} (4.16)

\[ = X_{2n+1} (k) \]

and

\[ X_{2n+1} (N - k) = \left\langle \sum_{n=0}^{N-1} x(2n+1)\beta \left[ \frac{2(N - k - 1) + 1 + 2n}{2} \right] \right\rangle_{M_p} \]

\[ = \left\langle \sum_{n=0}^{N-1} x(2n+1)\beta \left( -\frac{2k+1}{2} - 2n \right) \right\rangle_{M_p} \]  \hspace{1cm} (4.17)

\[ = X_{2n+1} (N - k) \]

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

\[ X \left( \frac{N}{2} - k \right) = \left\langle X^{ev} \left( \frac{N}{2} - k \right) + X^{od} \left( \frac{N}{2} - k \right) \right\rangle_{M_p} \]

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

\[ + \sum_{n=0}^{N-1} x(2n+1)\beta \left[ \frac{2 \left( \frac{N}{2} - k - 1 \right) + 1 + (2n+1)}{2} \right] \]  \hspace{1cm} \( X_{2n+1} \left( \frac{N}{2} - k \right) \)

\[ + \beta_1 \left[ \frac{2 \left( \frac{N}{2} - k - 1 \right) + 1}{2} \right] X_{2n+1} \left[ N - \left( \frac{N}{2} - k \right) \right] \]

\[ + \beta_2 \left[ \frac{2 \left( \frac{N}{2} - k - 1 \right) + 1}{2} \right] X_{2n+1} \left( N - \left( \frac{N}{2} - k \right) \right) \]

\[ = \left\langle X_{2n} \left( N - k \right) - \beta_1 \left( \frac{2k+1}{2} \right) X_{2n+1} \left( N - k \right) \right. \]

\[ + \beta_2 \left( \frac{2k+1}{2} \right) X_{2n+1} \left( k \right) \right\rangle_{M_p} \]  \hspace{1cm} (4.18)
Similarly, in addition to equations (4.11) and (4.18) the rest of the points will be derived respectively as

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

and

\[ X(N - k) = \left\langle X_{2n}(N - k) + \beta_1 \left( \frac{2k + 1}{2} \right) X_{2n+1}(N - k) \right\rangle_{Mp} - \beta_2 \left( \frac{2k + 1}{2} \right) X_{2n+1}(k) \] (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)

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

and is split into left and right components where

\[ X(k) = X^{left}(2k) + X^{right}(2k + 1), \] (4.22)
4. FAST ALGORITHMS OF THE GNMNT

which expands to

\[ X(k) = \left\langle \sum_{n=0}^{N/2 - 1} x(n) \beta \left( \frac{2n + 1}{2} \right) + \sum_{n=N/2}^{N-1} x(n) \beta \left( \frac{2n + 1}{2} \right) \right\rangle_{M_p} \]  \hspace{1cm} (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

\[ X(k) = \left\langle \sum_{n=0}^{N/2 - 1} x(n) \beta \left( \frac{2n + 1}{2} \right) + \sum_{n=N/2}^{N-1} x(n) \beta \left( \frac{2n + 1}{2} \right) \right\rangle_{M_p} \]  \hspace{1cm} (4.24)

The next step is to solve the last term of \( \beta \) so

\[ \beta \left( \frac{2(n + N/2) + 1}{2} \right) = \beta \left( \frac{N + 2n + 1}{2} \right) \]  \hspace{1cm} (4.25)

and subsequently

\[ \beta \left( \frac{N + 2n + 1}{2} \right) = \beta \left( \frac{Nk}{2} + \frac{2n + 1}{2} \right) \]  \hspace{1cm} (4.26)

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

\[ \beta_1 \left( \frac{Nk}{2} \right) \beta \left( \frac{2n + 1}{2} \right) + \beta_2 \left( \frac{Nk}{2} \right) \beta \left( -\frac{2n + 1}{2} \right) \]  \hspace{1cm} (4.27)

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

\[ X(k) = \left\langle \sum_{n=0}^{N/2 - 1} \left[ x(n) + \beta_1 \left( \frac{Nk}{2} \right) x(n + \frac{N}{2}) + \beta_2 \left( \frac{Nk}{2} \right) x(N - n) \right] \beta \left( \frac{2n + 1}{2} \right) \right\rangle_{M_p} \]  \hspace{1cm} (4.28)

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

\[ X(k) = \left\langle \sum_{n=0}^{N/2 - 1} \left[ x(n) + \beta_1 \left( \frac{Nk}{2} \right) x(n + \frac{N}{2}) \right] \beta \left( \frac{2n + 1}{2} \right) \right\rangle_{M_p} \]  \hspace{1cm} (4.29)

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

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

84
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

\[
X^{ev}(k) = \left\langle \sum_{n=0}^{N-1} x(2n) \beta \left( \frac{2k+1}{2} \right) \right\rangle_{M_p} \tag{4.35}
\]

and

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

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

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

Decomposing the final \(\beta\) term in (4.37) produces

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

and further decomposition produces

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

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

\[
X_{2n}(k) = \left\langle \sum_{n=0}^{N-1} x(4n) \beta \left( \frac{2k+1}{2} \right) \right\rangle_{M_p} + \beta_1 \left( \frac{2k+1}{2} \right) \sum_{n=0}^{N-1} x(4n+2) \beta \left( \frac{2k+1}{2} \right) + \beta_2 \left( \frac{2k+1}{2} \right) \sum_{n=0}^{N-1} x(4n+2) \beta \left( -\frac{2k+1}{2} \right) . \tag{4.40}
\]

In order to simplify (4.40) we define

\[
X_{4n}(k) = \left\langle \sum_{n=0}^{N-1} x(4n) \beta \left( \frac{2k+1}{2} \right) \right\rangle_{M_p} . \tag{4.41}
\]
and
\[
X_{4n} (N - k) = \left\langle \sum_{n=0}^{N-1} x(4n)\beta \left( -\frac{2k+1}{2}4n \right) \right\rangle_{M_p} . \tag{4.42}
\]

Applying (4.41) and (4.42) to (4.40) now produces
\[
X_{2n} (k) = X_{4n} (k) + \beta_1 \left( \frac{2k+1}{2} \times 2 \right) X_{4n+2} (k) + \beta_2 \left( \frac{2k+1}{2} \times 2 \right) X_{4n+2} (N - k) . \tag{4.43}
\]
The odd part is denoted as
\[
X^{od} (k) = \left\langle \sum_{n=0}^{N-1} x(4n+1)\beta \left[ \frac{2k+1}{2} (4n+1) \right] \right\rangle_{M_p} . \tag{4.44}
\]
Simplifying both \( \beta \) terms in (4.44), produces
\[
\beta \left[ \frac{2k+1}{2} (4n+1) \right] = \beta \left( \frac{2k+1}{2}4n + \frac{2k+1}{2} \right) = \beta_1 \left( \frac{2k+1}{2} \right) \beta \left( \frac{2k+1}{2}4n \right) + \beta_2 \left( \frac{2k+1}{2} \beta \left( -\frac{2k+1}{2}4n \right) \right) \tag{4.45}
\]
and
\[
\beta \left[ \frac{2k+1}{2} (4n+3) \right] = \beta \left( \frac{2k+1}{2}4n + \frac{2k+1}{2} \times 3 \right) = \beta_1 \left( \frac{2k+1}{2} \times 3 \right) \beta \left( \frac{2k+1}{2}4n \right) + \beta_2 \left( \frac{2k+1}{2} \times 3 \beta \left( -\frac{2k+1}{2}4n \right) \right) . \tag{4.46}
\]
Applying equations (4.45) and (4.46) into (4.44) produces
\[
X^{od} (k) = \left\langle \beta_1 \left( \frac{2k+1}{2} \right) \sum_{n=0}^{N-1} x(4n+1)\beta \left( \frac{2k+1}{2}4n \right) \right. \\
+ \beta_2 \left( \frac{2k+1}{2} \right) \sum_{n=0}^{N-1} x(4n+1)\beta \left( -\frac{2k+1}{2}4n \right) \\
+ \beta_1 \left( \frac{2k+1}{2} \times 3 \right) \sum_{n=0}^{N-1} x(4n+3)\beta \left( \frac{2k+1}{2}4n \right) \\
+ \beta_2 \left( \frac{2k+1}{2} \times 3 \right) \sum_{n=0}^{N-1} x(4n+3)\beta \left( -\frac{2k+1}{2}4n \right) \right\rangle_{M_p} . \tag{4.47}
\]
4. FAST ALGORITHMS OF THE GNMNT

Similar to (4.41), define

\[ X_{4n+1}(k) = \left\langle \sum_{n=0}^{2^i-1} x(4n+1)\beta \left( \frac{2k+1}{2} \right) \right\rangle_{M_p} \quad (4.48) \]

and

\[ X_{4n+3}(k) = \left\langle \sum_{n=0}^{2^i-1} x(4n+3)\beta \left( \frac{2k+1}{2} \right) \right\rangle_{M_p} \quad (4.49) \]

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

\[ X^{\text{odd}}(k) = \left\langle \beta_1 \left( \frac{2k+1}{2} \right) X_{4n+1}(k) + \beta_2 \left( \frac{2k+1}{2} \right) X_{4n+1}(N-k) \right. \]
\[ + \beta_1 \left( \frac{2k+1}{2} \times 3 \right) X_{4n+3}(k) + \beta_2 \left( \frac{2k+1}{2} \times 3 \right) X_{4n+3}(N-k) \left. \right\rangle_{M_p} . \quad (4.50) \]

The equation of ONMNT radix-4 at point \( k \) is

\[ X(k) = X^{\text{ev}}(k) + X^{\text{odd}}(k) \quad (4.51) \]

and in the decomposed form becomes

\[ X(k) = \left\langle X_{4n}(k) + \beta_1 \left( \frac{2k+1}{2} \times 2 \right) X_{4n+2}(k) + \beta_2 \left( \frac{2k+1}{2} \times 2 \right) X_{4n+2}(N-k) \right. \]
\[ + \beta_1 \left( \frac{2k+1}{2} \right) X_{4n+1}(k) + \beta_2 \left( \frac{2k+1}{2} \right) X_{4n+1}(N-k) \]
\[ + \beta_1 \left( \frac{2k+1}{2} \times 3 \right) X_{4n+3}(k) + \beta_2 \left( \frac{2k+1}{2} \times 3 \right) X_{4n+3}(N-k) \left. \right\rangle_{M_p} . \quad (4.52) \]

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 \( \frac{N}{4} - k, \therefore \frac{N}{4} - k :\leftrightarrow \frac{N}{4} - k - 1 \) by definition, then

\[ X_{4n}(k) \xrightarrow{k = \frac{N}{4} - k} X_{4n}\left( \frac{N}{4} - k \right) = \left\langle \sum_{n=0}^{N-1} x(4n)\beta \left[ \frac{2 \left( \frac{N}{4} - k - 1 \right) + 1}{2} \right] \right\rangle_{M_p} . \quad (4.53) \]
Figure 4.3: Radix-4 1D-ONMNT In-Place DIT Butterfly

Decomposing the $\beta$ term in (4.53) produces

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

We can identify the following $\beta$ 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 $\theta$ for ease of
4. FAST ALGORITHMS OF THE GNMNT

reference

\[
\begin{bmatrix}
X(k) \\
X(k + \frac{N}{4}) \\
X(k + \frac{N}{4} - k) \\
X(k + \frac{3N}{4} - k) \\
X(k + \frac{3N}{4}) \\
X(N - k)
\end{bmatrix} = \begin{bmatrix}
A + B_2 + B_1 + B_3 \\
A + B_2 + B_1 - B_3 \\
A - B_2 - B_1 + B_3 \\
A - B_2 + B_1 - B_3 \\
A - B_2 + B_1 + B_3 \\
A - B_2 - B_1 - B_3
\end{bmatrix}
\]

(4.61)

where

\[
A = X_{4n}(k)
\]

(4.62)

\[
\overline{A} = X_{4n}(\frac{N}{4} - k)
\]

(4.63)

\[
B_m = \langle [\beta_1 (\theta m) X_{4n+m}(k) + \beta_2 (\theta m) X_{4n+m} (\frac{N}{4} - k)] \rangle_{M_p}
\]

(4.64)

and

\[
\overline{B}_m = \langle [\beta_2 (\theta m) X_{4n+m}(k) - \beta_1 (\theta m) X_{4n+m} (\frac{N}{4} - k)] \rangle_{M_p}
\]

(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

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

(4.66)

Adjusting the range in (4.66) we then obtain

\[
X(k) = \left\langle \sum_{n=0}^{N-1} \left\{ x(n) \beta \left( \frac{2n + 1}{2} k \right) + x \left( n + \frac{N}{4} \right) \beta \left( \frac{2(n + \frac{N}{4}) + 1}{2} k \right) \right\} \right\rangle_{M_p}
\]

(4.67)
Figure 4.4: Radix-4 In-Place DIT Flow Diagram, $N = 16$
and can be simplified as

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

The \( \beta \) terms in (4.68) can decomposed as

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

noting that the \( \beta_2 \) term in (4.69) will always equate to zero, we further obtain

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

and

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

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

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

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 \( k \):

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

\[
\beta_1 \left(\frac{N}{4}4k\right) = 1
\]

\[
\beta_2 \left(\frac{N}{4}4k\right) = 0
\]
and $4k + 2$ into (4.72) replacing $k$:

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

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

and

$$\beta_2 \left[ \frac{N}{4} (4k + 2) \right] = 0.$$  \hspace{1cm} (4.78)

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

$$X (4k) = \left< \sum_{n=0}^{N-1} \left[ x(n) + x \left( n + \frac{N}{2} \right) + x \left( n + \frac{N}{4} \right) + x \left( n + \frac{3N}{4} \right) \right] \beta \left( \frac{2n + 1}{2} 4k \right) \right>_{Mp}$$  \hspace{1cm} (4.79)

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

$$X (4k + 2) = \left< \sum_{n=0}^{N-1} \left[ x(n) + x \left( n + \frac{N}{2} \right) - x \left( n + \frac{N}{4} \right) - x \left( n + \frac{3N}{4} \right) \right] \beta \left( \frac{2n + 1}{2} 4k + 2 \right) \right>_{Mp}$$  \hspace{1cm} (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 $k$:

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

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

$$\beta_2 \left[ \frac{N}{4} (4k + 1) \right] = 1.$$  \hspace{1cm} (4.83)
4. FAST ALGORITHMS OF THE GNMNT

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

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

\[ \beta_1 \left[ \frac{N}{4} (4k + 3) \right] = 0 \]  \hspace{1cm} (4.85) \\

\[ \beta_2 \left[ \frac{3N}{4} (4k + 3) \right] = 1. \]  \hspace{1cm} (4.86)
Putting equations (4.81)-(4.83) back into (4.72) we get

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

(4.87)

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

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

(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 $\phi$ for ease of reference.

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

(4.89)

where

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

(4.90)

(4.91)

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

(4.92)

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

(4.93)

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

(4.94)

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

(4.95)

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

(4.96)

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

(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.
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

\[ X(k) = X^{ev}(k) + X^{od}(k) \]  \hspace{1cm} (4.98)

where

\[ X^{ev}(k) = \left\langle \sum_{n=0}^{N/2-1} x(2n)\beta \left( \frac{2k + 1}{2} \right)^{2n} \right\rangle_{M_p} \]  \hspace{1cm} (4.99)

and

\[ X^{od}(k) = \left\langle \sum_{n=0}^{N/4-1} x(4n+1)\beta \left( \frac{4n + 1}{2} \right)^{2k+1} \right\rangle_{M_p} + \left\langle \sum_{n=0}^{N/4-1} x(4n+3)\beta \left( \frac{4n + 3}{2} \right)^{2k+1} \right\rangle_{M_p} \]  \hspace{1cm} (4.100)

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

\[ X_{2n}(k) = \left\langle \sum_{n=0}^{N/2-1} x(2n)\beta \left( \frac{2k + 1}{2} \right)^{2n} \right\rangle_{M_p} \]  \hspace{1cm} (4.101)
4. FAST ALGORITHMS OF THE GNMNT

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

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

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

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

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

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

$$X(k) = \left\langle X_{2n}(k) + \beta_1 \left( \frac{2k+1}{2} \right) X_{4n+1}(k) + \beta_2 \left( \frac{2k+1}{2} \right) X_{4n+1}(k) + \beta_1 \left( \frac{2k+1}{2} \times 3 \right) X_{4n+3}(k) + \beta_2 \left( \frac{2k+1}{2} \times 3 \right) X_{4n+3}(k) \right\rangle_{M_p} \hspace{1cm} (4.104)$$
Equations for all points of the split-radix ONMNT DIT butterfly can now be derived from

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

(4.105)

where

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

\[
\overline{A} = X_{2n}\left(\frac{N}{2} - k\right) \tag{4.107}
\]

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

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

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

\[
X(2k) = \left\langle \sum_{n=0}^{N-1} \left[ x(n) + x\left(n + \frac{N}{2}\right) \right] \beta\left(\frac{2n + 1}{2} - 2k\right) \right\rangle_{M_p} \tag{4.111}
\]
Similarly, the right parts of the radix-4 ONMNT DIF (4.87) and (4.88) as

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

and

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

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

Figure 4.9: Split-Radix In-Place DIF Flow Diagram, $N = 16$
4. FAST ALGORITHMS OF THE GNMNT

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

\[
M(N) = N (\log_2 N - 1) \lambda \tag{4.114}
\]

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>
<thead>
<tr>
<th>( N )</th>
<th>Radix-2</th>
<th>Direct</th>
<th>Improvement %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mults</td>
<td>Adds</td>
<td>Total</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>14</td>
<td>22</td>
</tr>
<tr>
<td>8</td>
<td>32</td>
<td>40</td>
<td>72</td>
</tr>
<tr>
<td>16</td>
<td>96</td>
<td>104</td>
<td>200</td>
</tr>
<tr>
<td>32</td>
<td>256</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td>64</td>
<td>640</td>
<td>608</td>
<td>1248</td>
</tr>
<tr>
<td>128</td>
<td>1536</td>
<td>1408</td>
<td>2944</td>
</tr>
<tr>
<td>256</td>
<td>3584</td>
<td>3200</td>
<td>6784</td>
</tr>
<tr>
<td>512</td>
<td>8192</td>
<td>7168</td>
<td>15360</td>
</tr>
<tr>
<td>1024</td>
<td>18432</td>
<td>15872</td>
<td>34304</td>
</tr>
<tr>
<td>2048</td>
<td>40960</td>
<td>34816</td>
<td>75776</td>
</tr>
<tr>
<td>4096</td>
<td>90112</td>
<td>75776</td>
<td>165888</td>
</tr>
<tr>
<td>8192</td>
<td>196608</td>
<td>163840</td>
<td>360448</td>
</tr>
<tr>
<td>16384</td>
<td>425984</td>
<td>352256</td>
<td>778240</td>
</tr>
<tr>
<td>32768</td>
<td>917504</td>
<td>753664</td>
<td>1671168</td>
</tr>
<tr>
<td>65536</td>
<td>1966080</td>
<td>1605632</td>
<td>3571712</td>
</tr>
<tr>
<td>131072</td>
<td>4194304</td>
<td>3407872</td>
<td>7602176</td>
</tr>
</tbody>
</table>

Table 4.1: ONMNT Radix-2 Complexity
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

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

and

\[
A(N) = \frac{N}{4} \left( 11 \log_4 N - 1 \right) , \quad (4.117)
\]

<table>
<thead>
<tr>
<th>( N )</th>
<th>Mults</th>
<th>Adds</th>
<th>Total</th>
<th>Mults</th>
<th>Adds</th>
<th>Total</th>
<th>Improvement %</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8</td>
<td>10</td>
<td>18</td>
<td>8</td>
<td>10</td>
<td>18</td>
<td>0.00 0.00 0.00</td>
</tr>
<tr>
<td>8</td>
<td>24</td>
<td>30</td>
<td>54</td>
<td>32</td>
<td>40</td>
<td>72</td>
<td>25.00 25.00 25.00</td>
</tr>
<tr>
<td>16</td>
<td>72</td>
<td>82</td>
<td>154</td>
<td>80</td>
<td>94</td>
<td>174</td>
<td>20.00 21.15 23.00</td>
</tr>
<tr>
<td>32</td>
<td>184</td>
<td>286</td>
<td>470</td>
<td>256</td>
<td>256</td>
<td>512</td>
<td>28.13 29.53 28.83</td>
</tr>
<tr>
<td>64</td>
<td>456</td>
<td>496</td>
<td>954</td>
<td>640</td>
<td>608</td>
<td>1248</td>
<td>27.85 28.78 28.56</td>
</tr>
<tr>
<td>128</td>
<td>1080</td>
<td>1166</td>
<td>2246</td>
<td>1536</td>
<td>1488</td>
<td>3024</td>
<td>23.56 22.71 23.71</td>
</tr>
<tr>
<td>256</td>
<td>2504</td>
<td>2674</td>
<td>5178</td>
<td>3816</td>
<td>3772</td>
<td>7588</td>
<td>23.71 23.71 23.71</td>
</tr>
<tr>
<td>512</td>
<td>5058</td>
<td>6030</td>
<td>11188</td>
<td>8192</td>
<td>7168</td>
<td>15360</td>
<td>23.71 23.71 23.71</td>
</tr>
<tr>
<td>1024</td>
<td>12744</td>
<td>13428</td>
<td>26172</td>
<td>18432</td>
<td>15872</td>
<td>34304</td>
<td>23.71 23.71 23.71</td>
</tr>
<tr>
<td>2048</td>
<td>28216</td>
<td>29582</td>
<td>57798</td>
<td>40960</td>
<td>34816</td>
<td>75776</td>
<td>23.71 23.71 23.71</td>
</tr>
<tr>
<td>4096</td>
<td>61806</td>
<td>64260</td>
<td>126066</td>
<td>90112</td>
<td>75776</td>
<td>165888</td>
<td>23.71 23.71 23.71</td>
</tr>
<tr>
<td>8192</td>
<td>134712</td>
<td>141712</td>
<td>276424</td>
<td>196088</td>
<td>163448</td>
<td>360448</td>
<td>23.71 23.71 23.71</td>
</tr>
<tr>
<td>16384</td>
<td>292152</td>
<td>301944</td>
<td>594096</td>
<td>425084</td>
<td>352592</td>
<td>777672</td>
<td>23.71 23.71 23.71</td>
</tr>
<tr>
<td>32768</td>
<td>624304</td>
<td>640875</td>
<td>1265179</td>
<td>937590</td>
<td>75828</td>
<td>1716378</td>
<td>23.71 23.71 23.71</td>
</tr>
<tr>
<td>65536</td>
<td>1308608</td>
<td>1385338</td>
<td>2694046</td>
<td>1960800</td>
<td>1605632</td>
<td>3571712</td>
<td>23.71 23.71 23.71</td>
</tr>
<tr>
<td>131072</td>
<td>2617216</td>
<td>2760672</td>
<td>5377888</td>
<td>3921600</td>
<td>3211296</td>
<td>7132904</td>
<td>23.71 23.71 23.71</td>
</tr>
<tr>
<td>262144</td>
<td>5234432</td>
<td>5521344</td>
<td>10755776</td>
<td>7843200</td>
<td>6422592</td>
<td>14265792</td>
<td>23.71 23.71 23.71</td>
</tr>
</tbody>
</table>

Table 4.3: ONMNT Split-Radix Complexity
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

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

(4.118)

and while the number of additions can be expressed as

\[
A(N) = \frac{4}{3} N \log_2 N - \frac{2}{9} N + \frac{2}{9} (-1)^{\log_2 N}.
\]

(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

<table>
<thead>
<tr>
<th>( N )</th>
<th>Mults</th>
<th>Adds</th>
<th>FMAs</th>
<th>Total</th>
<th>Mults</th>
<th>Adds</th>
<th>Total</th>
<th>Mults</th>
<th>Adds</th>
<th>FMAs</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
<td>8</td>
<td>4</td>
<td>16</td>
<td>8</td>
<td>10</td>
<td>18</td>
<td>50.00</td>
<td>20.00</td>
<td>22.22</td>
<td>11.11</td>
</tr>
<tr>
<td>8</td>
<td>12</td>
<td>24</td>
<td>12</td>
<td>48</td>
<td>24</td>
<td>30</td>
<td>54</td>
<td>50.00</td>
<td>20.00</td>
<td>22.22</td>
<td>11.11</td>
</tr>
<tr>
<td>16</td>
<td>36</td>
<td>64</td>
<td>36</td>
<td>136</td>
<td>72</td>
<td>82</td>
<td>154</td>
<td>50.00</td>
<td>21.95</td>
<td>23.38</td>
<td>11.69</td>
</tr>
<tr>
<td>32</td>
<td>92</td>
<td>160</td>
<td>92</td>
<td>344</td>
<td>184</td>
<td>206</td>
<td>390</td>
<td>50.00</td>
<td>22.33</td>
<td>24.04</td>
<td>11.79</td>
</tr>
<tr>
<td>64</td>
<td>228</td>
<td>384</td>
<td>228</td>
<td>896</td>
<td>456</td>
<td>498</td>
<td>954</td>
<td>50.00</td>
<td>22.89</td>
<td>24.18</td>
<td>12.02</td>
</tr>
<tr>
<td>128</td>
<td>540</td>
<td>896</td>
<td>540</td>
<td>2368</td>
<td>1080</td>
<td>1166</td>
<td>2246</td>
<td>50.00</td>
<td>23.16</td>
<td>24.41</td>
<td>12.09</td>
</tr>
<tr>
<td>256</td>
<td>1252</td>
<td>2048</td>
<td>1252</td>
<td>4552</td>
<td>2504</td>
<td>2674</td>
<td>5178</td>
<td>50.00</td>
<td>23.58</td>
<td>24.72</td>
<td>12.14</td>
</tr>
<tr>
<td>512</td>
<td>2844</td>
<td>4608</td>
<td>2844</td>
<td>10240</td>
<td>6568</td>
<td>6030</td>
<td>12598</td>
<td>50.00</td>
<td>23.73</td>
<td>24.95</td>
<td>12.17</td>
</tr>
<tr>
<td>1024</td>
<td>6372</td>
<td>10240</td>
<td>6372</td>
<td>22984</td>
<td>12744</td>
<td>13426</td>
<td>26170</td>
<td>50.00</td>
<td>23.85</td>
<td>25.21</td>
<td>12.20</td>
</tr>
<tr>
<td>2048</td>
<td>14108</td>
<td>22528</td>
<td>14108</td>
<td>50736</td>
<td>28216</td>
<td>29582</td>
<td>57798</td>
<td>50.00</td>
<td>24.03</td>
<td>25.45</td>
<td>12.23</td>
</tr>
<tr>
<td>4096</td>
<td>30948</td>
<td>49152</td>
<td>30948</td>
<td>106496</td>
<td>61896</td>
<td>64626</td>
<td>126522</td>
<td>50.00</td>
<td>24.21</td>
<td>25.68</td>
<td>12.26</td>
</tr>
<tr>
<td>8192</td>
<td>67356</td>
<td>104696</td>
<td>67356</td>
<td>221408</td>
<td>134712</td>
<td>140174</td>
<td>274886</td>
<td>50.00</td>
<td>24.39</td>
<td>25.90</td>
<td>12.29</td>
</tr>
<tr>
<td>16384</td>
<td>154568</td>
<td>239776</td>
<td>154568</td>
<td>526896</td>
<td>281272</td>
<td>302194</td>
<td>583466</td>
<td>50.00</td>
<td>24.56</td>
<td>26.12</td>
<td>12.32</td>
</tr>
<tr>
<td>32768</td>
<td>313116</td>
<td>491520</td>
<td>313116</td>
<td>1293728</td>
<td>562368</td>
<td>591208</td>
<td>1153576</td>
<td>50.00</td>
<td>24.73</td>
<td>26.34</td>
<td>12.35</td>
</tr>
<tr>
<td>65536</td>
<td>669924</td>
<td>1048576</td>
<td>669924</td>
<td>2708112</td>
<td>1293800</td>
<td>1351360</td>
<td>2645160</td>
<td>50.00</td>
<td>24.91</td>
<td>26.55</td>
<td>12.38</td>
</tr>
<tr>
<td>131072</td>
<td>1427228</td>
<td>2228224</td>
<td>1427228</td>
<td>5424336</td>
<td>2563856</td>
<td>2735776</td>
<td>5299632</td>
<td>50.00</td>
<td>25.09</td>
<td>26.77</td>
<td>12.41</td>
</tr>
<tr>
<td>262144</td>
<td>3029220</td>
<td>4718592</td>
<td>3029220</td>
<td>10860640</td>
<td>3029220</td>
<td>29788240</td>
<td>32817460</td>
<td>50.00</td>
<td>25.27</td>
<td>26.98</td>
<td>12.44</td>
</tr>
</tbody>
</table>

Table 4.4: ONMNT Fused-Split-Radix Complexity

4. FAST ALGORITHMS OF THE GNMNT
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

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

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

\[ A(N) = N \log_2 N, \quad (4.122) \]

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

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

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

Listing 4.1: Typical Implementation

```c
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];
```
4.5 Complexity Analysis

![Graph showing processing time against vector length](image)

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

```plaintext
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**

```plaintext
unsigned int t0, t1, in0, in1, out0, out1;
unsigned int B1[N], iB[N], B2[N];
t0 = in0 * B1[theta];
t1 = in1 * iB[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 $\beta_1$ (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>
<thead>
<tr>
<th>$r$</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>16</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
<td>256</td>
<td>-</td>
<td>256</td>
</tr>
<tr>
<td>9</td>
<td>512</td>
<td>-</td>
<td>512</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>1024</td>
<td>1024</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>2048</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>4096</td>
<td>4096</td>
<td>4096</td>
<td>4096</td>
</tr>
</tbody>
</table>
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 $2^r$ 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 processor 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**

```c
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 $\beta_1$ 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
4. FAST ALGORITHMS OF THE GNMNT

/ 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 $\beta_1$ or $\beta_2$, 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.
4. FAST ALGORITHMS OF THE GNMNT
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 $\beta$ 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

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

However, as the \(\beta\) 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

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

This is obviously a different algorithm because it is clear that

\[
\beta(n_1 k_1, n_2 k_2) \neq \beta(n_1 k_1) \beta(n_2 k_2). \tag{5.3}
\]

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 \(\beta(n_1 k_1, n_2 k_2)\) term to

\[
\beta(n_1 k_1, n_2 k_2) = \langle \beta_1(n_1 k_1, n_2 k_2) + \beta_2(n_1 k_1, n_2 k_2) \rangle_{M_p}. \tag{5.4}
\]

where we can further derive \(\beta_1\) and \(\beta_2\) by their definitions as

\[
\beta_1(n_1 k_1, n_2 k_2) = \left\langle \text{Re} \left( \alpha_1 + j \alpha_2 \right)^{n_1 k_1 + n_2 k_2} \right\rangle_{M_p}, \tag{5.5}
\]

and

\[
\beta_2(n_1 k_1, n_2 k_2) = \left\langle \text{Im} \left( \alpha_1 + j \alpha_2 \right)^{n_1 k_1 + n_2 k_2} \right\rangle_{M_p}. \tag{5.6}
\]

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

\[
\beta_1(n_1 k_1, n_2 k_2) = \left\langle \text{Re} \left[ \left( \alpha_1 + j \alpha_2 \right)^{n_1 k_1} \left( \alpha_1 + j \alpha_2 \right)^{n_2 k_2} \right] \right\rangle_{M_p}, \tag{5.7}
\]

and

\[
\beta_2(n_1 k_1, n_2 k_2) = \left\langle \text{Im} \left[ \left( \alpha_1 + j \alpha_2 \right)^{n_1 k_1} \left( \alpha_1 + j \alpha_2 \right)^{n_2 k_2} \right] \right\rangle_{M_p}. \tag{5.8}
\]
Simplifying the $\beta_1$ term in (5.7) produces

$$\beta_1(n_1k_1, n_2k_2) = \text{Re} \left( \left[ \text{Re} (\alpha_1 + j\alpha_2)^{n_1k_1} + j\text{Im} (\alpha_1 + j\alpha_2)^{n_1k_1} \right] \times \left[ \text{Re} (\alpha_1 + j\alpha_2)^{n_2k_2} + j\text{Im} (\alpha_1 + j\alpha_2)^{n_2k_2} \right] \right)_{M_p}$$

$$= \text{Re} \left( \beta_1(n_1k_1) \beta_1(n_2k_2) + j\beta_1(n_1k_1) \beta_2(n_2k_2) \right. + j\beta_1(n_2k_2) \beta_2(n_1k_1) - \beta_2(n_1k_1) \beta_2(n_2k_2) \right)_{M_p}$$

$$= \beta_1(n_1k_1 + n_2k_2)$$

and similarly

$$\beta_2(n_1k_1, n_2k_2) = \beta_2(n_1k_1 + n_2k_2).$$

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

$$\beta(n_1k_1, n_2k_2) = \left< \beta_1(n_1k_1 + n_2k_2) + \beta_2(n_1k_1 + n_2k_2) \right>_{M_p}$$

$$= \beta(n_1k_1 + n_2k_2)$$

$$= \left< \beta_1(n_1k_1) \beta(n_2k_2) + \beta_2(n_1k_1) \beta(-n_2k_2) \right>_{M_p}$$

because

$$\beta_1(n_1k_1) = \left< \beta(n_1k_1) - \beta_2(n_1k_1) \right>_{M_p}$$

and

$$\beta_1(-n_1k_1) = \beta_1(n_1k_1)$$

$$= \left< \beta(-n_1k_1) + \beta_2(n_1k_1) \right>_{M_p}.$$

Combining (5.12) and (5.13) produces

$$2\beta_1(n_1k_1) = \left< \beta(n_1k_1) + \beta(-n_2k_2) \right>_{M_p}$$

thus

$$\beta_1(n_1k_1) = \left< \frac{1}{2} \left[ \beta(n_1k_1) + \beta(-n_2k_2) \right] \right>_{M_p}$$

and

$$\beta_2(n_1k_1) = \left< \frac{1}{2} \left[ \beta(n_1k_1) - \beta(-n_2k_2) \right] \right>_{M_p}. $$
Substituting (5.15) and (5.16) into (5.11) produces

\[
\beta(n_1 k_1, n_2 k_2) = \left< \beta_1(n_1 k_1) \beta(n_2 k_2) + \beta_2(n_1 k_1) \beta(-n_2 k_2) \right>_{M_p}
\]

\[
= \left< \frac{1}{2} \left[ \beta(n_1 k_1) + \beta(-n_1 k_1) \right] \beta(n_2 k_2) \right. \\
+ \left. \frac{1}{2} \left[ \beta(n_1 k_1) - \beta(-n_1 k_1) \right] \beta(-n_2 k_2) \right>_{M_p}
\]

\[
= \left< \left[ \beta(n_1 k_1) \beta(n_2 k_2) + \beta(-n_1 k_1) \beta(n_2 k_2) \\
+ \beta(n_1 k_1) \beta(-n_2 k_2) - \beta(-n_1 k_1) \beta(-n_2 k_2) \right] 2^{p-1} \right>_{M_p}.
\]

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

\[
X(k_1, k_2) = \left< \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1, n_2) \left\{ \beta(n_1 k_1) \beta(n_2 k_2) + \beta(-n_1 k_1) \beta(n_2 k_2) \\
+ \beta(n_1 k_1) \beta(-n_2 k_2) - \beta(-n_1 k_1) \beta(-n_2 k_2) \right\} 2^{p-1} \right>_{M_p}
\]

and further simplified as

\[
X(k_1, k_2) = \left< \left[ X_{RC}(k_1, k_2) + X_{RC}(-k_1, k_2) \\
+ X_{RC}(k_1, -k_2) - X_{RC}(-k_1, -k_2) \right] 2^{p-1} \right>_{M_p}.
\]

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

#### Original Data

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

#### RC-NMNT

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7529</td>
<td>2694</td>
<td>481</td>
<td>4482</td>
<td>193</td>
<td>7035</td>
<td>485</td>
<td>2675</td>
</tr>
<tr>
<td>7002</td>
<td>1040</td>
<td>469</td>
<td>6771</td>
<td>4793</td>
<td>7249</td>
<td>6809</td>
<td>1345</td>
</tr>
<tr>
<td>115</td>
<td>7364</td>
<td>43</td>
<td>2416</td>
<td>8142</td>
<td>965</td>
<td>8146</td>
<td>5741</td>
</tr>
<tr>
<td>843</td>
<td>6700</td>
<td>7642</td>
<td>4925</td>
<td>7343</td>
<td>3853</td>
<td>2856</td>
<td>994</td>
</tr>
<tr>
<td>295</td>
<td>2738</td>
<td>83</td>
<td>2628</td>
<td>8078</td>
<td>5555</td>
<td>63</td>
<td>5445</td>
</tr>
<tr>
<td>1625</td>
<td>850</td>
<td>7890</td>
<td>7230</td>
<td>3514</td>
<td>5915</td>
<td>926</td>
<td>1396</td>
</tr>
<tr>
<td>657</td>
<td>4701</td>
<td>8012</td>
<td>175</td>
<td>45</td>
<td>2912</td>
<td>8076</td>
<td>8114</td>
</tr>
<tr>
<td>349</td>
<td>4362</td>
<td>8144</td>
<td>7137</td>
<td>688</td>
<td>1067</td>
<td>5419</td>
<td>3910</td>
</tr>
</tbody>
</table>

#### 2D-NMNT via Separable Algorithm

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7529</td>
<td>2694</td>
<td>481</td>
<td>4482</td>
<td>193</td>
<td>7035</td>
<td>485</td>
<td>2675</td>
</tr>
<tr>
<td>7002</td>
<td>5514</td>
<td>906</td>
<td>1854</td>
<td>4793</td>
<td>3975</td>
<td>6372</td>
<td>5062</td>
</tr>
<tr>
<td>115</td>
<td>4846</td>
<td>8158</td>
<td>322</td>
<td>8142</td>
<td>3059</td>
<td>31</td>
<td>68</td>
</tr>
<tr>
<td>843</td>
<td>3574</td>
<td>540</td>
<td>951</td>
<td>7343</td>
<td>7827</td>
<td>1767</td>
<td>4120</td>
</tr>
<tr>
<td>295</td>
<td>2738</td>
<td>83</td>
<td>2628</td>
<td>8078</td>
<td>5555</td>
<td>63</td>
<td>5445</td>
</tr>
<tr>
<td>1625</td>
<td>3976</td>
<td>6801</td>
<td>3013</td>
<td>3514</td>
<td>1941</td>
<td>2015</td>
<td>6461</td>
</tr>
<tr>
<td>657</td>
<td>7219</td>
<td>8088</td>
<td>2269</td>
<td>45</td>
<td>818</td>
<td>8000</td>
<td>5596</td>
</tr>
<tr>
<td>349</td>
<td>8079</td>
<td>7707</td>
<td>3863</td>
<td>688</td>
<td>4341</td>
<td>5856</td>
<td>193</td>
</tr>
</tbody>
</table>

#### Inverse RC-NMNT

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
<td>158</td>
<td>166</td>
<td>172</td>
<td>171</td>
<td>161</td>
<td>154</td>
<td>146</td>
</tr>
<tr>
<td>166</td>
<td>140</td>
<td>4242</td>
<td>135</td>
<td>152</td>
<td>132</td>
<td>4287</td>
<td>196</td>
</tr>
<tr>
<td>169</td>
<td>4190</td>
<td>60</td>
<td>108</td>
<td>106</td>
<td>115</td>
<td>141</td>
<td>4278</td>
</tr>
<tr>
<td>165</td>
<td>52</td>
<td>4130</td>
<td>94</td>
<td>91</td>
<td>93</td>
<td>4230</td>
<td>146</td>
</tr>
<tr>
<td>150</td>
<td>57</td>
<td>8</td>
<td>71</td>
<td>122</td>
<td>154</td>
<td>163</td>
<td>146</td>
</tr>
<tr>
<td>104</td>
<td>10</td>
<td>4088</td>
<td>26</td>
<td>122</td>
<td>171</td>
<td>4243</td>
<td>112</td>
</tr>
<tr>
<td>125</td>
<td>4158</td>
<td>61</td>
<td>134</td>
<td>164</td>
<td>141</td>
<td>4179</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>100</td>
<td>4195</td>
<td>82</td>
<td>125</td>
<td>171</td>
<td>4189</td>
<td>72</td>
</tr>
</tbody>
</table>

#### Original Data via Separable Algorithm

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

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

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

for \( k_1, k_2 = 0, 1, 2, ..., N - 1 \). (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

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

\[
= \beta \left( \frac{2Nn}{2} + \frac{-2k + 1}{2} n \right)
\]

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

\[
= \beta \left( \frac{-2k - 1}{2} n \right).
\]

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

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

\[
= \beta \left( \frac{2Nn}{2} + \frac{-2k - 1}{2} n \right)
\]

\[
= \beta \left( \frac{-2k - 1}{2} n \right)
\]

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

The RC definition for the 2D-ONMNT is

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

for \( k_1, k_2 = 0, 1, 2, ..., N - 1 \). (5.23)
and the full separable equation that was shown in (5.18) now becomes
\[
X(k_1, k_2) = \left\langle \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1, n_2) \left[ \beta \left( \frac{2k_1 + 1}{2} n_1 \right) \beta \left( \frac{2k_2 + 1}{2} n_2 \right) + \beta \left( -\frac{2k_1 + 1}{2} n_1 \right) \beta \left( -\frac{2k_2 + 1}{2} n_2 \right) \right] \right\rangle_{M_p}.
\]

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
\[
\beta \left( \frac{2k_1 + 1}{2} n_1 \right) \bigg|_{k_1=N-k_1-1} = \beta \left( \frac{2N - 2k_1 - 1}{2} n_1 \right) = \beta \left( -\frac{2k_1 + 1}{2} n_1 \right)
\]

and can be represented by using \( X_{RC} \) so that
\[
X(k_1, k_2) = \left\langle \left[ X_{RC}(k_1, k_2) + X_{RC}(-k_1 - 1, k_2) + X_{RC}(k_1, -k_2 - 1) - X_{RC}(-k_1 - 1, -k_2 - 1) \right] 2^{p-1} \right\rangle_{M_p}.
\]

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 \( \vec{X} \) to indentify that the IONMNT transform is currently selected. The standard equation for the IONMNT then becomes
\[
\vec{X}(k_1, k_2) = \left\langle \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1, n_2) \beta \left( \frac{2n_1 + 1}{2} k_1, \frac{2n_2 + 1}{2} k_2 \right) \right\rangle_{M_p}
\]
for \( k_1, k_2 = 0, 1, 2, ..., N - 1 \)

where the RC method is defined as
\[
\vec{X}_{RC}(k_1, k_2) = \left\langle \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1, n_2) \beta \left( \frac{2n_1 + 1}{2} k_1 \right) \beta \left( \frac{2n_2 + 1}{2} k_2 \right) \right\rangle_{M_p}
\]
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

$$
\beta \left( \frac{2n_1 + 1}{2} k_1, \frac{2n_2 + 1}{2} k_2 \right) = \beta \left( \frac{2n_1 + 1}{2} k_1 + \frac{2n_2 + 1}{2} k_2 \right)
$$

$$
= \left\langle \frac{1}{2} \left[ \beta \left( \frac{2n_1 + 1}{2} k_1 \right) + \beta \left( -\frac{2n_1 + 1}{2} k_1 \right) \right] \times \beta \left( \frac{2n_2 + 1}{2} k_2 \right) 
+ \frac{1}{2} \left[ \beta \left( \frac{2n_1 + 1}{2} k_1 \right) - \beta \left( -\frac{2n_1 + 1}{2} k_1 \right) \right] \times \beta \left( -\frac{2n_2 + 1}{2} k_2 \right) \right\rangle_{Mp}
$$

Substituting (5.29) into (5.27) produces

$$
\mathbf{X}(k_1, k_2) = \left\langle \left[ \mathbf{X}_{RC}(k_1, k_2) + \mathbf{X}_{RC}(-k_1, k_2) + \mathbf{X}_{RC}(k_1, -k_2) - \mathbf{X}_{RC}(-k_1, -k_2) \right] 2^{p-1} \right\rangle_{Mp}.
$$

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

$$
\mathbf{X}(k_1, k_2) = \left\langle \left[ \mathbf{X}_{RC}(k_1, k_2) + \mathbf{X}_{RC}(-k_1, k_2) + \mathbf{X}_{RC}(k_1, -k_2) - \mathbf{X}_{RC}(-k_1, -k_2) \right] 2^{p-1} \right\rangle_{Mp}.
$$

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

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

\[
X(k_1, k_2) = \left\langle \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1, n_2) \beta \left( \frac{2k_1 + 1}{2} n_1, \frac{2k_2 + 1}{2} n_2 \right) \right\rangle_{M_p} \tag{5.32}
\]

for \( k_1, k_2 = 0, 1, 2, ..., N - 1 \)

\((k_1, k_2)\) 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

\[
X(0, k_2) = \left\langle \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1, n_2) \beta \left( \frac{2k_2 + 1}{2} n_2 \right) \right\rangle_{M_p} \tag{5.33}
\]

for \( k_2 = 0, 1, 2, ..., N - 1 \).

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

\[
X(k_1, 0) = \left\langle \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1, n_2) \beta \left( \frac{2k_1 + 1}{2} n_1 \right) \right\rangle_{M_p} \tag{5.34}
\]

for \( k_1 = 0, 1, 2, ..., N - 1 \).

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

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

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

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

Comparing (5.35) and (5.36), the value in \( \beta \) 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 $8 \times 8$ 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

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

for $k_1, k_2 = 0, 1, 2, ..., N - 1$  \hspace{1cm} (5.37)

and

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

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

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

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

for $k_1, k_2 = 0, 1, 2, ..., N - 1$  \hspace{1cm} (5.39)
5. THE ROW-COLUMN GNMNT

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

<table>
<thead>
<tr>
<th>6369</th>
<th>2526</th>
<th>6957</th>
<th>6406</th>
<th>6828</th>
<th>1123</th>
<th>7131</th>
<th>7442</th>
</tr>
</thead>
<tbody>
<tr>
<td>7046</td>
<td>4108</td>
<td>6223</td>
<td>7810</td>
<td>1877</td>
<td>2225</td>
<td>3747</td>
<td>3642</td>
</tr>
<tr>
<td>4915</td>
<td>7986</td>
<td>3956</td>
<td>279</td>
<td>4174</td>
<td>1463</td>
<td>5526</td>
<td>1254</td>
</tr>
<tr>
<td>3810</td>
<td>5103</td>
<td>43</td>
<td>4742</td>
<td>5036</td>
<td>6544</td>
<td>4623</td>
<td>3575</td>
</tr>
<tr>
<td>4341</td>
<td>1657</td>
<td>6535</td>
<td>5159</td>
<td>4162</td>
<td>2913</td>
<td>1531</td>
<td>6198</td>
</tr>
<tr>
<td>2250</td>
<td>4247</td>
<td>6319</td>
<td>2299</td>
<td>2699</td>
<td>260</td>
<td>3158</td>
<td>2601</td>
</tr>
<tr>
<td>6009</td>
<td>4214</td>
<td>7901</td>
<td>8104</td>
<td>5408</td>
<td>8034</td>
<td>4751</td>
<td>499</td>
</tr>
<tr>
<td>5832</td>
<td>6204</td>
<td>1261</td>
<td>1761</td>
<td>7147</td>
<td>3917</td>
<td>7787</td>
<td>2977</td>
</tr>
</tbody>
</table>

Figure 5.3: Example of 2D-O²NMNT 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

$$
\beta\left[\frac{(2k_1+1)(2n_1+1)}{4}, \frac{(2k_2+1)(2n_2+1)}{4}\right] = \beta\left[\frac{(2k_1+1)(2n_1+1)}{4} + \frac{(2k_2+1)(2n_2+1)}{4}\right] + \beta\left[\frac{(2k_1+1)(2n_1+1)}{4}\right] - \beta\left[\frac{(2k_1+1)(2n_1+1)}{4}\right].
$$

(5.40)

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

$$
X(k_1,k_2) = \left[X_{RC}(k_1,k_2) + X_{RC}(-k_1-1,k_2) + X_{RC}(k_1,-k_2-1) - X_{RC}(-k_1-1,-k_2-1)\right]e^{2^{-1}}_{M_p}.
$$

(5.41)

A demonstration of how this is presented is shown in Figure 5.3, which started by reducing the cameraman image to an $8 \times 8$ array and using the RC technique to transform the data to a 2D-O$^2$NMNT, using $N = 8$, $p = 13$ and $M_p = 8191$. 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

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

(5.42)

and

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

(5.43)
for the radix-2 implementation;

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

and

\[ A_{RC}(N) = \frac{N^2}{2} (11 \log_4 N - 1) \]  \hspace{1cm} (5.45)

for the radix-4 implementation;

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

and

\[ A_{RC}(N) = \frac{8}{3} N^2 \log_2 N - \frac{4}{9} N^2 + \frac{4}{9} N(-1)^{\log_2 N} \]  \hspace{1cm} (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

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

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

and

\[ A_{RC}(N) = 2N^2 \log_2 N, \]  \hspace{1cm} (5.50)

to produce a total operational count of

\[ T_{RC}(N) = \frac{4\lambda + 6}{3} N^2 \log_2 N - \frac{8\lambda}{9} N^2 + \frac{8\lambda}{9} N(-1)^{\log_2 N}. \]  \hspace{1cm} (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

\[ M_S(N) = 2N^4 (\log_2 N - 1) \lambda \]  \hspace{1cm} (5.52)
5.5 Complexity Analysis

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

and

\[ A_S(N) = 9N^4 \log_2 N + 3N^4 \]  

(5.53)

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

\[ M_S(N) = \left[ N^4 \left( 3 \log_4 N - 1 \right) \right] \lambda \]  

(5.54)

and

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

(5.55)

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

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

(5.56)

and

\[ A_S(N) = 8N^4 \log_2 N - \frac{4}{3} N^4 + \frac{4}{3} N^3 (-1)^{\log_2 N} \]  

(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
5. THE ROW-COLUMN GNMNT

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

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

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

$$A_S(N) = 6N^4 \log_2 N, \quad (5.59)$$

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

$$T_S(N) = \frac{2\lambda + 3}{3} N^4 \log_2 N - \frac{4\lambda}{3} N^4 + \frac{4\lambda}{3} N^3 (-1)^{\log_2 N} + \frac{2\lambda}{3} N^2 \log_2 N - \frac{4\lambda}{3} N^2 + \frac{4\lambda}{3} N (-1)^{\log_2 N}. \quad (5.60)$$
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

$$
Y_{cc}(k_1, k_2) = \langle 2^{p-1} [H(k_1, k_2) + H(N - k_1, N - k_2)] X(k_1, k_2)
+ 2^{p-1} [H(k_1, k_2) - H(N - k_1, N - k_2)] X(N - k_1, N - k_2) \rangle_{M_p}
$$

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

$$
Y_{cc}(k_1, k_2) = X(k_1, k_2) \odot H(k_1, k_2)
= \langle H^{ev}(k_1, k_2)X(k_1, k_2)
+ H^{od}(k_1, k_2)X(N - k_1, N - k_2) \rangle_{M_p}
$$

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

$$
H^{ev}(k_1, k_2) = \langle [H(k_1, k_2) + H(N - k_1, N - k_2)] 2^{p-1} \rangle_{M_p}
$$

Figure 5.6: Convolution Process Structure for the 2D-NMNT
and

\[ H^{\text{od}}(k_1, k_2) = \left\langle [H(k_1, k_2) - H(N - k_1, N - k_2)] 2^{p-1} \right\rangle_{M_p}. \]  

(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 \( X_O(k_1, k_2) \), \( H_O(k_1, k_2) \) and \( Y_{\text{occ}}(k_1, k_2) \) are the signals \( x(n_1, n_2) \), \( h(n_1, n_2) \) and \( y_{\text{oc}}(n_1, n_2) \) that are within the ONMNT domain. The operation to derive the ONMNT result \( Y_{\text{occ}}(k_1, k_2) \) is processed using \( X_O(k_1, k_2) \), \( H_O(k_1, k_2) \) by

\[ Y_{\text{ooc}}(k_1, k_2) = X_O(k_1, k_2) \odot H_O(k_1, k_2) \]

\[ = \left\langle H^{\text{ev}}_O(k_1, k_2)X_O(k_1, k_2) + H^{\text{od}}_O(k_1, k_2)X_O(N - k_1 - 1, N - k_2 - 1) \right\rangle_{M_p} \]  

(5.65)

where \( H^{\text{ev}}_O(k_1, k_2) \) and \( H^{\text{od}}_O(k_1, k_2) \) stand for even and odd parts of \( H_O(k_1, k_2) \) respectively as

\[ H^{\text{ev}}_O(k_1, k_2) = \left\langle [H_O(k_1, k_2) + H_O(N - k_1 - 1, N - k_2 - 1)] 2^{p-1} \right\rangle_{M_p} \]  

(5.66)

and

\[ H^{\text{od}}_O(k_1, k_2) = \left\langle [H_O(k_1, k_2) - H_O(N - k_1 - 1, N - k_2 - 1)] 2^{p-1} \right\rangle_{M_p}. \]  

(5.67)

The process of computing the 2D cyclic convolution for the ONMNT is shown in Figure 5.7.
5.6.3 Cyclic Convolution for the 2D-O\textsuperscript{2}NMNT

Transforming into the O\textsuperscript{2}NMNT domain, the input and output signals \(x(n_1,n_2), h(n_1,n_2)\) and \(y_{cc}(n_1,n_2)\) are represented by \(X_{O^2}(k_1,k_2), H_{O^2}(k_1,k_2)\) and \(Y_{O^2cc}(k_1,k_2)\) respectively and processed as

\[
Y_{O^2cc}(k_1,k_2) = X_O(k_1,k_2) \odot H_{O^2}(k_1,k_2)
\]

\[
= \left\langle H_{O^2}^{ev}(k_1,k_2)X_O(k_1,k_2) + H_{O^2}^{od}(k_1,k_2)X_O(N-k_1-1,N-k_2-1) \right\rangle_{M_p}
\]

(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 \(H_{O^2}(k_1,k_2)\) respectively as

\[
H_{O^2}^{ev}(k_1,k_2) = \left\langle [H_{O^2}(k_1,k_2) + H_{O^2}(N-k_1-1,k_2-1)] 2^{p-1} \right\rangle_{M_p}
\]

(5.69)

and

\[
H_{O^2}^{od}(k_1,k_2) = \left\langle [H_{O^2}(k_1,k_2) - H_{O^2}(N-k_1-1,k_2-1)] 2^{p-1} \right\rangle_{M_p}.
\]

(5.70)

Figure 5.8 shows the process of computing 2D cyclic convolution for O\textsuperscript{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

\[
H = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}.
\]

(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...
Figure 5.9: Convolution using Sobel Filter for Edge Detection using Cameraman Image with NMNT
Figure 5.10: Convolution using Sobel Filter for Edge Detection using Lena Image with ONMNT
Figure 5.11: Convolution using Sobel Filter for Edge Detection using Baboon Image with O²NMNT
across the x- and y-axes produces

\[ G_x = H @ Y \tag{5.72} \]

and

\[ G_y = H' @ Y. \tag{5.73} \]

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**

<table>
<thead>
<tr>
<th>( N )</th>
<th>1D Block and Key Size</th>
<th>2D Block and Key Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )</td>
<td>5 7 13 17 19</td>
<td>5 7 13 17 19</td>
</tr>
<tr>
<td>4</td>
<td>20 28 52 68 76</td>
<td>80 112 208 272 304</td>
</tr>
<tr>
<td>8</td>
<td>40 56 104 136 152</td>
<td>320 448 832 1088 1216</td>
</tr>
<tr>
<td>16</td>
<td>80* 112 208 272 304</td>
<td>1280* 1792 3328 4352 4864</td>
</tr>
<tr>
<td>32</td>
<td>160* 224 416 544 608</td>
<td>5120* 7168 13312 17408 19456</td>
</tr>
<tr>
<td>64</td>
<td>- 448* 832 1088 1216</td>
<td>- 28672* 53248 69632 77824</td>
</tr>
<tr>
<td>128</td>
<td>- 896* 1664 2176 2432</td>
<td>- 114688* 212992 278528 311296</td>
</tr>
<tr>
<td>256</td>
<td>- - 3328 4352 4864</td>
<td>- - 851968 1114112 1245184</td>
</tr>
<tr>
<td>512</td>
<td>- - 6656 8704 9728</td>
<td>- - 3407872 4456448 4980736</td>
</tr>
<tr>
<td>1024</td>
<td>- - 13312 17408 19456</td>
<td>- - 13631488 17825792 19922944</td>
</tr>
</tbody>
</table>

* NMNT and ONMNT, \# NMNT Only
5. THE ROW-COLUMN GNMNT

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](image-url)
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 $16 \times 8$ 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

<table>
<thead>
<tr>
<th>Table 5.2: Pixel Values of 2D Key without Concatenation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xb1 0x7e 0x9e 0x87 0x80 0x95 0x63 0x60 0x76 0x7b 0x68 0x6a 0x7a 0x97 0xa0</td>
</tr>
<tr>
<td>0x90 0x80 0x9d 0x9f 0x8e 0x79 0x8a 0xbe 0xad 0x92 0x90 0x4b 0x9a 0x8d 0x78 0x5f</td>
</tr>
<tr>
<td>0x42 0x5f 0x75 0x6d 0xa5 0x88 0xaf 0x95 0x58 0x4e 0x85 0x82 0x86 0x73 0x9c</td>
</tr>
<tr>
<td>0x94 0x51 0x39 0x4e 0x7f 0x82 0x86 0x44 0x5f 0x88 0x64 0xa9 0x87 0x5e 0xa1</td>
</tr>
<tr>
<td>0x78 0x69 0x94 0x9b 0x36 0x62 0x54 0x57 0x72 0x45 0x40 0x8b 0xa6 0x6d 0x87 0x77</td>
</tr>
<tr>
<td>0x44 0x6a 0x69 0x32 0x5a 0x47 0x4f 0x87 0x68 0x5b 0x3c 0x49 0x55 0x69 0x6b</td>
</tr>
<tr>
<td>0x72 0x90 0x55 0x1c 0x6b 0x80 0x87 0x8e 0x86 0xb1 0x5b 0x3f 0x84 0xa4</td>
</tr>
<tr>
<td>0x67 0x76 0x59 0x1e 0x38 0x74 0x9c 0xa0 0x46 0x72 0x80 0x87 0x3a 0x64 0x99 0x61</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 5.3: Pixel Values of 2D Key with Concatenation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xb1 0x9e87 0x7180 0x9563 0xb076 0x7b68 0x6a7a 0x97a0</td>
</tr>
<tr>
<td>0x9080 0x9d9f 0x8e79 0x8abe 0xad92 0x904b 0x9a8d 0x785f</td>
</tr>
<tr>
<td>0x426f 0x5f75 0x6da5 0xa8af 0x9558 0x4e85 0x8286 0x739c</td>
</tr>
<tr>
<td>0x9451 0x3739 0x4e7f 0x8286 0x445f 0x8864 0xa987 0x5ea1</td>
</tr>
<tr>
<td>0x7869 0x949b 0x3662 0x5457 0x7245 0x408b 0xa66d 0x8777</td>
</tr>
<tr>
<td>0x4464 0xa66b 0x325a 0x474f 0x8768 0x5b3c 0x4955 0x6b6b</td>
</tr>
<tr>
<td>0x7291 0x551e 0x6b4e 0x78ac 0x4839 0x86b1 0x5b3f 0x8da4</td>
</tr>
<tr>
<td>0x6776 0x591e 0x3874 0x9ca0 0x4672 0x8087 0x3a64 0x9961</td>
</tr>
</tbody>
</table>
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
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.
5.7.1.3 RC-O\textsuperscript{2}NMNT

The second of the two new transforms applied for encryption using the RC method is the O\textsuperscript{2}NMNT. Like its counterparts, the O\textsuperscript{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\textsuperscript{2}NMNT would be applicable for use in the field of cryptography.

Figure 5.15: Encrypted Camerman using RC-O\textsuperscript{2}NMNT, $N = 8 \times 8$ and $Mp = 131071$
Figure 5.16: Decrypted Cameraman Error using RC-GNMNT, $N = 8 \times 8$ and $M_p = 131071$

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.
5. THE ROW-COLUMN GNMNT

Figure 5.17: Encrypted Cameraman using 2D-NMNT, $N = 8 \times 8$ and $M_p = 131071$

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.
5.7 Encryption Applications

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.

Figure 5.18: Encrypted Camerman using 2D-ONMNT, \( N = 8 \times 8 \) and \( M_p = 131071 \)
5. THE ROW-COLUMN GNMNT

Figure 5.19: Encrypted Camerman using 2D-O\(^2\)NMNT, \(N = 8 \times 8\) and \(M_p = 131071\)

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
decryption 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
Figure 5.21: Decrypted Bit Error using RC-NMNT with $N = 8, 16, 32, 64, 128$ and $Mp = 131071$
5.7 Encryption Applications

Figure 5.22: Decrypted Bit Error using RC-ONMNT with $N = 8, 16, 32, 64, 128$ and $Mp = 131071$
5. THE ROW-COLUMN GNMNT

Figure 5.23: Decrypted Bit Error using RC-O\(^2\)NMNT with \(N = 8, 16, 32, 64, 128\) and \(Mp = 131071\)
Figure 5.24: Decrypted Bit Error using 2D-NMNT with $N = 8, 16, 32, 64, 128$ and $Mp = 131071$
Figure 5.25: Decrypted Bit Error using 2D-ONMNT with $N = 8, 16, 32, 64, 128$ and $Mp = 131071$
Figure 5.26: Decrypted Bit Error using 2D-O^2NMNT 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 $O^2$NMNT 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-$O^2$NMNT, 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 $O^2$NMNT 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 $O^2$NMNT 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
6. THE AVALANCHE EFFECT OF THE GNMNT

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

$$c = \mathcal{E}(t).$$

(6.1)

We can then change a single bit in $c$ by selecting a bit position $n$ and performing

$$\hat{c}_m = c \oplus 2^n \text{ for } 0 \leq n < pN$$

(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
6.3 Methodology

\[ G(t) = G(t) \]

\[ t^\hat{G}[e] = < t_{G} \oplus 2^b > \]

\[ M_{p} \leq e < N, 0 \leq b \leq p \]

\[ \hat{t} = G^{-1}(t^\hat{G}) \]

\[ \delta_{m}[n^\hat{e}] = ((t[e] \oplus t^\hat{e}[e]) >> b) \wedge 1 \]

\[ 0 \leq e < N, 0 \leq b < p, m = p \cdot e + b, n = p \cdot e + b \]

\[ \delta_{n}[m^\hat{e}] = ((t[e] \oplus t^\hat{e}[e]) >> b) \wedge 1 \]

\[ 0 \leq e < N, 0 \leq b \leq p, m = p \cdot e + b, n = p \cdot e + b \]

\[ \Delta_{m}=\Sigma \delta_{m}(\hat{e}) \]

\[ \Delta_{n}=\Sigma \delta_{n}(\hat{e}) \]

\[ \Delta_{m} = \left[ \frac{1}{pN} \sum_{n=0}^{pN-1} \delta_{m}(n) \right] \times 100. \]  

(6.3)

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

\[ \delta_{m}(n) = \begin{cases} 
1, & \text{if } (t \oplus \hat{t}) \wedge 2^n \neq 0 \\
0, & \text{otherwise}
\end{cases} \]  

(6.7)

and

\[ \Delta_{n} = \left[ \frac{1}{pN} \sum_{n=0}^{pN-1} \delta_{n}(m) \right] \times 100. \]  

(6.8)

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

\[ n = pe + b, \text{ for } 0 \leq e < N \text{ and } 0 \leq b < p, \]  

(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

\[ \hat{t}_m = D(\hat{c}_m) \]  

(6.4)

where \( D(\cdot) \) refers to the corresponding decryption algorithm from (6.1). The bits that have changed can be easily detected by developing a small function

\[ \delta_{m}(n) = \begin{cases} 
1, & \text{if } (t \oplus \hat{t}_m) \wedge 2^n \neq 0 \\
0, & \text{otherwise}
\end{cases} \]  

(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

\[ \Delta_{m} = \left[ \frac{1}{pN} \sum_{n=0}^{pN-1} \delta_{m}(n) \right] \times 100. \]  

(6.6)
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

$$t = \{3, 2, 1, 2\}. \quad (6.9)$$

Table 6.1: Modification of $\hat{t}_9$ for $p = 3, N = 4$

<table>
<thead>
<tr>
<th>e</th>
<th>b</th>
<th>$\hat{t}_{9}[3]$</th>
<th>$\hat{t}_{9}[2]$</th>
<th>$\hat{t}_{9}[1]$</th>
<th>$\hat{t}_{9}[0]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>001</td>
<td>010</td>
<td>000</td>
<td>011</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>001</td>
<td>010</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>001</td>
<td>010</td>
<td>000</td>
<td>110</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>001</td>
<td>010</td>
<td>001</td>
<td>010</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>001</td>
<td>010</td>
<td>010</td>
<td>010</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>001</td>
<td>010</td>
<td>100</td>
<td>010</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>001</td>
<td>011</td>
<td>000</td>
<td>010</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>001</td>
<td>000</td>
<td>000</td>
<td>010</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>001</td>
<td>110</td>
<td>000</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>000</td>
<td>010</td>
<td>000</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>011</td>
<td>010</td>
<td>000</td>
<td>010</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>101</td>
<td>010</td>
<td>000</td>
<td>010</td>
</tr>
</tbody>
</table>

Table 6.2: Resultant Values of $\hat{t}$ for $p = 3, N = 4$

<table>
<thead>
<tr>
<th>$pe + b$</th>
<th>$\hat{t}[3]$</th>
<th>$\hat{t}[2]$</th>
<th>$\hat{t}[1]$</th>
<th>$\hat{t}[0]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>101</td>
<td>100</td>
<td>110</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>110</td>
<td>101</td>
<td>101</td>
<td>110</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>011</td>
<td>000</td>
<td>001</td>
</tr>
<tr>
<td>3</td>
<td>101</td>
<td>000</td>
<td>011</td>
<td>000</td>
</tr>
<tr>
<td>4</td>
<td>000</td>
<td>101</td>
<td>101</td>
<td>101</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>001</td>
<td>010</td>
<td>001</td>
</tr>
<tr>
<td>6</td>
<td>101</td>
<td>000</td>
<td>110</td>
<td>100</td>
</tr>
<tr>
<td>7</td>
<td>110</td>
<td>110</td>
<td>101</td>
<td>101</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
<td>001</td>
<td>000</td>
<td>011</td>
</tr>
<tr>
<td>9</td>
<td>001</td>
<td>000</td>
<td>110</td>
<td>000</td>
</tr>
<tr>
<td>10</td>
<td>000</td>
<td>110</td>
<td>101</td>
<td>110</td>
</tr>
<tr>
<td>11</td>
<td>100</td>
<td>011</td>
<td>010</td>
<td>011</td>
</tr>
</tbody>
</table>
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

\[ T_g = \{001_2, 010_2, 000_2, 010_2\}. \tag{6.11} \]

Now we can produce a loop using \( e \) and \( b \), which were defined in (6.3) to change each bit of \( T_g \) in sequence resulting in \( pN \) new transformed vectors denoted by \( \hat{T}_g \). These altered values of \( T_g \) are shown in Table 6.1. Applying the inverse NMNT transform to each of the \( \hat{T}_g \) 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 \( \delta_m \) and \( \Delta_m \) 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 \( \delta_n \) and \( \Delta_n \) respectively. The results

<table>
<thead>
<tr>
<th>( \delta_m )</th>
<th>( \Delta_m )</th>
<th>( m )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 0 1 1 0 1 1 1 0 1 0</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>1 0 1 1 1 1 1 1 0 0 1 0 0</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>1 1 1 0 0 1 0 0 1 0 1 1 1</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>1 1 1 0 0 1 0 0 1 0 0 1 0</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>0 1 1 1 1 1 0 0 1 1 1 1 1</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>1 1 1 0 1 1 0 1 1 0 1 1</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>1 1 0 0 1 0 1 1 1 1 1 1 0</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>1 0 1 1 1 0 0 1 0 0 1 1 1</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>1 1 1 0 1 1 0 0 1 0 0 1 0</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>0 1 0 0 1 0 1 1 1 0 1 0</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>0 1 1 1 0 0 1 0 0 1 0 0</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>1 1 1 0 0 1 0 1 1 0 0 1</td>
<td>7</td>
<td>11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \Delta_n )</th>
<th>( \sum \Delta_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 10 8 5 8 6 7 6 7 5 8 6</td>
<td>( \delta_g )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( n )</th>
<th>( \sum \Delta_m )</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 10 9 8 7 6 5 4 3 2 1 0</td>
<td>( \sum \Delta_m = \sum \Delta_n )</td>
</tr>
</tbody>
</table>
of the XOR operation between $t$ and each $\hat{t}$ that produces the respective $\delta$ and $\Delta$ values that can be seen in Table 6.3. This table also shows the sums of the two sequences of $\Delta$, which naturally should be equal. Calculating the mean $\bar{x}$ can be achieved by using either $\Delta_m$ or $\Delta_n$ 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 straightforward process as

$$\sigma = \sqrt{\frac{1}{m} \sum_{m=0}^{N-1} (\Delta_m - \bar{x})^2}. \quad (6.12)$$

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 $\sigma_\tau$ and calculate it thus:

$$\sigma_\tau = (pN\sigma) \times \frac{1}{100}. \quad (6.13)$$

### 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 \( \sigma \) that was observed to serve as a tolerance, also as a percentage and the standard deviation \( \sigma_r \) 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.
6. THE AVALANCHE EFFECT OF THE GNMNT

![Graphs showing changes in diffusion and amount of change over rounds.](image)

(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

<table>
<thead>
<tr>
<th>Rounds</th>
<th>$\Delta_n$%</th>
<th>$\Delta_n$ bits</th>
<th>$\Delta_{max}$%</th>
<th>$\Delta_{max}$ bits</th>
<th>$\Delta_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>Max</td>
<td>Range</td>
<td>$\sigma$</td>
<td>$\sigma_x$</td>
</tr>
<tr>
<td>1</td>
<td>0.7813</td>
<td>6.2500</td>
<td>5.4688</td>
<td>0.1252</td>
<td>0.1600</td>
</tr>
<tr>
<td>2</td>
<td>3.1250</td>
<td>23.4375</td>
<td>20.3125</td>
<td>0.3406</td>
<td>0.4360</td>
</tr>
<tr>
<td>3</td>
<td>25.7813</td>
<td>74.2188</td>
<td>48.4375</td>
<td>0.9190</td>
<td>1.1760</td>
</tr>
<tr>
<td>4</td>
<td>25.0000</td>
<td>73.4375</td>
<td>48.4375</td>
<td>0.9085</td>
<td>1.1630</td>
</tr>
<tr>
<td>5</td>
<td>25.0000</td>
<td>73.4375</td>
<td>48.4375</td>
<td>0.9085</td>
<td>1.1630</td>
</tr>
<tr>
<td>6</td>
<td>25.0000</td>
<td>75.0000</td>
<td>50.0000</td>
<td>0.9085</td>
<td>1.1630</td>
</tr>
<tr>
<td>7</td>
<td>25.0000</td>
<td>75.7813</td>
<td>50.7813</td>
<td>0.9085</td>
<td>1.1630</td>
</tr>
<tr>
<td>8</td>
<td>25.0000</td>
<td>73.4375</td>
<td>48.4375</td>
<td>0.9086</td>
<td>1.1630</td>
</tr>
<tr>
<td>9</td>
<td>25.7813</td>
<td>75.0000</td>
<td>49.2188</td>
<td>0.9086</td>
<td>1.1630</td>
</tr>
<tr>
<td>10</td>
<td>26.5625</td>
<td>74.2188</td>
<td>47.6563</td>
<td>0.9086</td>
<td>1.1630</td>
</tr>
<tr>
<td>11</td>
<td>25.7813</td>
<td>73.4375</td>
<td>47.6563</td>
<td>0.9085</td>
<td>1.1630</td>
</tr>
<tr>
<td>12</td>
<td>25.0000</td>
<td>76.5625</td>
<td>51.5625</td>
<td>0.9085</td>
<td>1.1630</td>
</tr>
<tr>
<td>13</td>
<td>25.0000</td>
<td>75.0000</td>
<td>50.0000</td>
<td>0.9086</td>
<td>1.1630</td>
</tr>
<tr>
<td>14</td>
<td>25.7813</td>
<td>75.0000</td>
<td>49.2188</td>
<td>0.9085</td>
<td>1.1630</td>
</tr>
<tr>
<td>15</td>
<td>26.5625</td>
<td>74.2188</td>
<td>47.6563</td>
<td>0.9085</td>
<td>1.1630</td>
</tr>
<tr>
<td>16</td>
<td>24.2188</td>
<td>75.0000</td>
<td>50.7813</td>
<td>0.9085</td>
<td>1.1630</td>
</tr>
</tbody>
</table>
6.4 Assessing the Default Configuration

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

<table>
<thead>
<tr>
<th>Rounds</th>
<th>Δm%</th>
<th>Δm Bits</th>
<th>Δm%</th>
<th>Δm Bits</th>
<th>Δm%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.412</td>
<td>38.971</td>
<td>34.559</td>
<td>0.212</td>
<td>0.288</td>
</tr>
<tr>
<td>2</td>
<td>0.735</td>
<td>11.764</td>
<td>11.029</td>
<td>0.063</td>
<td>0.085</td>
</tr>
<tr>
<td>3</td>
<td>4.412</td>
<td>38.971</td>
<td>34.559</td>
<td>0.212</td>
<td>0.288</td>
</tr>
<tr>
<td>4</td>
<td>0.735</td>
<td>11.764</td>
<td>11.029</td>
<td>0.063</td>
<td>0.085</td>
</tr>
<tr>
<td>5</td>
<td>4.412</td>
<td>38.971</td>
<td>34.559</td>
<td>0.212</td>
<td>0.288</td>
</tr>
<tr>
<td>6</td>
<td>0.735</td>
<td>11.764</td>
<td>11.029</td>
<td>0.063</td>
<td>0.085</td>
</tr>
<tr>
<td>7</td>
<td>4.412</td>
<td>38.971</td>
<td>34.559</td>
<td>0.212</td>
<td>0.288</td>
</tr>
<tr>
<td>8</td>
<td>0.735</td>
<td>11.764</td>
<td>11.029</td>
<td>0.063</td>
<td>0.085</td>
</tr>
</tbody>
</table>

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
6. THE AVALANCHE EFFECT OF THE GNMNT

(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

<table>
<thead>
<tr>
<th>Rounds</th>
<th>( \Delta m )%</th>
<th>( \Delta m ) bits</th>
<th>( \Delta mn )%</th>
<th>( \Delta mn ) bits</th>
<th>( \Delta n )%</th>
<th>( \Delta n ) bits</th>
<th>( x )</th>
<th>( \sigma )</th>
<th>( \sigma )</th>
<th>( \sigma )</th>
<th>( x )</th>
<th>( \sigma )</th>
<th>( \sigma )</th>
<th>( \sigma )</th>
<th>( \sigma )</th>
<th>( \sigma )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21.3235</td>
<td>54.4118</td>
<td>33.0882</td>
<td>0.6781</td>
<td>0.9220</td>
<td>30.9032</td>
<td>0.2230</td>
<td>0.1642</td>
<td>47.1186</td>
<td>52.9952</td>
<td>5.8766</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>31.6176</td>
<td>72.7941</td>
<td>41.1765</td>
<td>0.8713</td>
<td>1.1850</td>
<td>46.9111</td>
<td>0.1190</td>
<td>0.0877</td>
<td>35.7798</td>
<td>53.3721</td>
<td>17.5924</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>27.9412</td>
<td>72.7941</td>
<td>44.8529</td>
<td>0.7971</td>
<td>1.0840</td>
<td>47.2102</td>
<td>0.0480</td>
<td>0.0354</td>
<td>8.8330</td>
<td>50.6216</td>
<td>41.7886</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>32.3529</td>
<td>72.0588</td>
<td>39.7059</td>
<td>0.9238</td>
<td>1.2560</td>
<td>49.3714</td>
<td>0.0440</td>
<td>0.0327</td>
<td>4.9214</td>
<td>52.2194</td>
<td>47.2980</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>29.4118</td>
<td>75.0000</td>
<td>45.5882</td>
<td>0.8292</td>
<td>1.1280</td>
<td>49.0878</td>
<td>0.0510</td>
<td>0.0376</td>
<td>8.7430</td>
<td>53.7353</td>
<td>44.9923</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>29.4118</td>
<td>73.5294</td>
<td>44.1176</td>
<td>0.8787</td>
<td>1.1660</td>
<td>49.9016</td>
<td>0.0540</td>
<td>0.0395</td>
<td>9.2530</td>
<td>54.7078</td>
<td>45.4548</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>31.6176</td>
<td>75.0000</td>
<td>43.8324</td>
<td>0.8556</td>
<td>1.1640</td>
<td>49.5020</td>
<td>0.0480</td>
<td>0.0349</td>
<td>6.3323</td>
<td>53.2010</td>
<td>46.8666</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>32.3529</td>
<td>76.4706</td>
<td>44.1176</td>
<td>0.8882</td>
<td>1.2080</td>
<td>50.2657</td>
<td>0.0490</td>
<td>0.0360</td>
<td>7.8730</td>
<td>54.0375</td>
<td>46.1644</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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
6.4 Assessing the Default Configuration

(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: O²NMNT Analysis

Table 6.7: O²NMNT Metrics - Vector / Bit Position

<table>
<thead>
<tr>
<th>Rounds</th>
<th>$\Delta_m$ %</th>
<th>$\Delta_m$ bits</th>
<th>$\Delta_{mn}$</th>
<th>$\Delta_n$ bits</th>
<th>$\Delta_n$ %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>Max</td>
<td>Range</td>
<td>$\sigma$</td>
<td>$\sigma_n$</td>
</tr>
<tr>
<td>1</td>
<td>36.0294</td>
<td>76.4706</td>
<td>40.4412</td>
<td>1.0047</td>
<td>1.3660</td>
</tr>
<tr>
<td>2</td>
<td>0.7353</td>
<td>11.7647</td>
<td>11.0294</td>
<td>0.0628</td>
<td>0.0850</td>
</tr>
<tr>
<td>3</td>
<td>36.0294</td>
<td>76.4706</td>
<td>40.4412</td>
<td>1.0047</td>
<td>1.3660</td>
</tr>
<tr>
<td>4</td>
<td>0.7353</td>
<td>11.7647</td>
<td>11.0294</td>
<td>0.0628</td>
<td>0.0850</td>
</tr>
<tr>
<td>5</td>
<td>36.0294</td>
<td>76.4706</td>
<td>40.4412</td>
<td>1.0047</td>
<td>1.3660</td>
</tr>
<tr>
<td>6</td>
<td>0.7353</td>
<td>11.7647</td>
<td>11.0294</td>
<td>0.0628</td>
<td>0.0850</td>
</tr>
<tr>
<td>7</td>
<td>36.0294</td>
<td>76.4706</td>
<td>40.4412</td>
<td>1.0047</td>
<td>1.3660</td>
</tr>
<tr>
<td>8</td>
<td>0.7353</td>
<td>11.7647</td>
<td>11.0294</td>
<td>0.0628</td>
<td>0.0850</td>
</tr>
</tbody>
</table>

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 O²NMNT

On first impressions, the results that are provided by the O²NMNT 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 O²NMNT shows very desirable characteristics.
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 unflinching 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 O²NMNT 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
6. THE AVALANCHE EFFECT OF THE GNMNT

6.5.2 Assessing the ONMNT (Shuffled)

Modifying the ONMNT does not appear to have brought any noticeable benefits. It doesn’t 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 O²NMNT (Shuffled)

The O²NMNT 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.

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 Assessing the Modified Configuration

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

\[
(2^p - 1)^N = (2^5 - 1)^8
= 852, 891, 037, 441 \text{ permutations.}
\]  

In order to undertake an exhaustive analysis, we must also take \( pN = 40 \) bits into consideration as well as the three forward \( \#G \) and three inverse \( \#G^{-1} \) operations,
resulting in
\[
\# \mathcal{S} \left\{ pN \left[ (2^p - 1)^N \right] \right\} + \# \mathcal{S}^{-1} \left\{ pN \left[ (2^p - 1)^N \right] \right\} = 204,693,848,985,840 \text{ operations.}
\] (6.15)

However, we can mitigate almost half of these operations by reusing the forward part of the algorithm so that we then have
\[
\# \mathcal{S} \left\{ (2^p - 1)^N \right\} + \# \mathcal{S}^{-1} \left\{ pN \left[ (2^p - 1)^N \right] \right\} = 104,905,597,605,243 \text{ operations.} \quad (6.16)
\]

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
consumer 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

<table>
<thead>
<tr>
<th>N</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1024</th>
<th>2048</th>
<th>4096</th>
<th>8192</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1, 2, 3</td>
<td>1, 2</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>17</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>19</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>31</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
<td>1, 2, 3</td>
</tr>
</tbody>
</table>

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
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 O²NMNT 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 doesn’t 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\(^2\)NMNT. 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\(^2\)NMNT, which was able do so in only one round. However, a number of potential weaknesses were exposed within the GNMNT, including the O\(^2\)NMNT 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 O²NMNT 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 O²NMNT 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 $O^2$NMNT 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);
A. RADIX-2 GNMNT ALGORITHMS

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++)
{
    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;
        }
    }
}
A. RADIX-2 GNMNT ALGORITHMS

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) **

```c
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++)
                ...
        }
    }
}
```

---

188
\begin{verbatim}
{ 
    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);
}
\end{verbatim}
Listing A.6: Radix-2 $O^2$NMNT (DIT)

// Forward and Inverse Radix–2 $O^2$NMNT (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--;    }
A. RADIX-2 GNMNT ALGORITHMS

\[
\text{for } (j = 0; j < N; j++) \\
\quad \text{vec}[j] = \text{mods(vec}[j] \times \text{inverse}, \ p);
\]

}
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
B. PARALLEL GNMNT ALGORITHMS

#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;
B. PARALLEL GNMNT ALGORITHMS

```c
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, MP − b1O8x52, 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;
\begin{verbatim}
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);
}
\end{verbatim}
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 gmnntAvN8p5(_constant uint *inVec,
        _global uint *outBuffer,
        _local uint *dataStore,
        _local uint *eVal,
```c
_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);
    }
```


B. PARALLEL GNMNT ALGORITHMS

// 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);
barrier (CLK_LOCAL_MEM_FENCE);

if (lid == 0)
{
    gid *= STORE;
    for (y = 0; y < STORE; y++)
        outBuffer[gid + y] = dataStore[y];
}
}
References


international/data/worldpop/table_population.php, December 2013, [20th May 2014].


REFERENCES


REFERENCES


REFERENCES


REFERENCES


International Symposium on Recent Advances in Intrusion Detection (RAID), 2008.


REFERENCES


REFERENCES


REFERENCES


