

**AN APPROACH TOWARDS DESIGN AND IMPLEMENTATION
OF MICROPROCESSOR AND FPGA BASED SYMMETRIC
ENCODER FOR EMBEDDED SYSTEM**

**Thesis submitted for the Degree of Doctor of Philosophy (Engineering)
In the Department of Computer Science and Engineering
Faculty of Engineering, Technology and Management
University of Kalyani**

By

Rajdeep Chakraborty

Under the Supervision of

Dr. Jyotsna Kumar Mandal

**Professor, Department of Computer Science and Engineering
University of Kalyani**

**Department of Computer Science and Engineering
University of Kalyani
Kalyani, Nadia, West Bengal, India**

January 2016

University of Kalyani

FACULTY OF ENGINEERING TECHNOLOGY AND MANAGEMENT

Dr. J. K. Mandal

M. Tech (Comp. Sc.), Ph. D.(Engg.)
Professor, Department of Computer
Science and Engineering (CSE),
University of Kalyani



Kalyani 741235, Nadia, West
Bengal, India

Phone: +91-33-25809617 (O)

Mobile: +91-9434352214

E-mail:

jkmandal@rediffmail.com,

jkm.cse@gmail.com

CERTIFICATE FROM THE SUPERVISOR

This is to certify that the thesis entitled “**AN APPROACH TOWARDS DESIGN AND IMPLEMENTATION OF MICROPROCESSOR AND FPGA BASED SYMMETRIC ENCODER FOR EMBEDDED SYSTEM**” submitted by Shri Rajdeep Chakraborty, who got his name registered on 04/July/2007 (Ref. No. Ph.D./Regn./Comp.Sc. & Engg./RC/2007 dated August 04, 2008) and re-registered on 04/July/2012 (Ref. No. Ph.D./Re-Regn./Com. Sc. & Engg./RC/2012 dated June 18, 2012) for the award of Ph.D (Computer Science and Engineering) degree of the University of Kalyani is absolutely based upon his own work under my supervision and that neither his thesis nor any part of the thesis has been submitted for any degree or diploma or any other academic award anywhere before. I recommend that Shri Rajdeep Chakraborty has fulfilled all the requirements according to rules of this university regarding the work embodied in this thesis.

Date:

(Dr. JYOTSNA KUMAR MANDAL)

Place: Kalyani

Prof, Dept. of CSE, FETM

University of Kalyani

Dedicated to My Father, Mother, Younger Brother and My Wife, Mrs. Priya Chakraborty

Acknowledgements

This thesis would have been impossible without the support and mentoring of my advisor, Dr. Jyotsna Kumar Mandal. Even after several years of working with him, I am constantly surprised by his amazing intelligence, infinite energy, boundless optimism, and genuine friendliness. He exhibits every and all support for the research carried out by me. Day or night, Office or home, time and places are no boundary for his encouragement and guide towards me. It's also worthwhile to say that his family members also supported me in every respect.

There is no word to express my feeling for my family, especially my wife, Mrs. Priya Chakraborty. She encouraged me in every sphere with her constant cooperation; actually she is the driver of my success and this thesis. She kept me tensionless in keeping all the family matter to able me to concentrate on my work. I also like to express my gratitude towards my parent, my father, retired bank official, encouraged and supported me since my childhood. He supported me in every respect in my studies and also during my research work. My mother, a housewife, sacrificed her life in service to the family. Her moral support has also helped me to overcome the odd times and hurdles of my life and makes this research work successful. Finally, I would also like to mention my younger brother for his help for some of the figures and diagrams for me, he is a good artist.

I would like to thank my ex-colleague and friends at IMPS College of Engineering and Technology (IMPSCET, Malda, West Bengal, India), Sikkim – Manipal Institute of Technology (SMIT, Majitar, Sikkim, India), Dept. of Computer Science and Application of University of North Bengal (Siliguri, West Bengal, India) and Ajay Binay Institute of Technology (ABIT, Cuttack, Orissa, India). I would also like to acknowledge the cooperation extended to me by faculties and staff of Netaji Subhash Engineering College (NSEC, Kolkata, West Bengal, India).

Like all the students I know, I am very happy I've had the chance to study and do my research work in Computer Science at Department of Computer Science and Engineering (CSE) in University of Kalyani, Kalyani, Nodia, West Bengal, India. The department has a unique atmosphere of academic excellence combined with friendliness and openness that makes it a very special place to learn in. I greatly enjoyed the interaction with quite lot faculty members, research scholars and fellow students.

One of the many reasons I am grateful to Mr. Avijit Das was his encouragement and support of my first job. As a result, in a relatively short time, I have met, interacted and

worked with so many wonderful people, that I cannot mention all of them here and also able to carry out my research which required monetary support.

In my early hours of my research work, it is worthwhile to mention the name of two persons, Mr. S. Mal and Mr. S. Sinha, for helping me to learn the microprocessor based programming. I have had helpful discussions and received comments and suggestions from many other people, also received a lot of input and support from my relatives and friends, and gave me unconditional aid. Last, but not the least, I thank god for his kindness and grace, which drives me through this work, in my past and also in the future.

Date:

Place: Kalyani

(RAJDEEP CHAKRABORTY)

Dept. of CSE, University of
Kalyani, Kalyani, W.B., India

List of Publications

International Journals

1. **Rajdeep Chakraborty** and J. K. Mandal, “An RTL Based Design & Implementation of Block Cipher through Time-Stamp-Keyed-Oriented Encryption Technique (TSK-OET)”, published in International Journal of Advanced Research in Computer Science (**IJARCS**), **ISSN 0976 – 5697**, accepted & published in Volume 2 – No. 1 (**Jan – Feb 2011**) issue, pp-428 – 432, indexed by Index Copernicus, Directory of Open Access Journal (DAOJ), Open J Gate, Ulrichs Web, EBSCOhost, Electronic Journal Library, New Jour, ScienceCentral.com, Genamics, Mlibrary of University of Michigan, Kun Shan University Library and Dayang Journal System.
2. **Rajdeep Chakraborty** and J. K. Mandal, “FPGA Based Cipher Design & Implementation of Recursive Oriented Block Arithmetic and Substitution Technique (ROBAST)”, published in (**IJACSA**) International Journal of Advanced Computer Science and Applications, **ISSN 2156-5570 (online)**, **ISSN 2158-107X (print)**, accepted and published in Volume 2- Issue 4 (**April 2011**) issue pp-54 – 59, indexed by. docstoc, Scribd, getCITED, CiteSeer^x, EBSCO HOST, Directory of Open Access Journal (DAOJ), Google Scholar, Journal Seek, Index Copernicus, GEORGETOWN UNIVERSITY LIBRARY and Powered by Microsoft Research.
3. **Rajdeep Chakraborty**, Sananda Mitra and J. K. Mandal, “Shuffle-RAT: An FPGA-based Iterative Block Cipher”, published in International Journal of Advanced Research in Computer Science (**IJARCS**), **ISSN 0976 – 5697**, accepted & published in Volume 2 – No. 3 (**May – June 2011**) issue, pp-21 – 24, indexed by Index Copernicus, Directory of Open Access Journal (DAOJ), Open J Gate, Ulrichs Web, EBSCOhost, Electronic Journal Library, New Jour, ScienceCentral.com, Genamics, Mlibrary of University of Michigan, Kun Shan University Library and Dayang Journal System.

4. **Rajdeep Chakraborty**, Sonam Agarwal, Sridipta Misra, Vineet Khemka, Sunit Kr Agarwal and J. K. Mandal, “Triple SV: A Bit Level Symmetric Block-Cipher Having High Avalanche Effect”, published in (**IJACSA**) International Journal of Advanced Computer Science and Applications, **ISSN 2156-5570 (online), ISSN 2158-107X (print)**, accepted and published in Volume 2- Issue 7 (**July 2011**) issue pp-61 – 68, indexed by. docstoc, Scribd, getCITED, CiteSeer^x, EBSCO HOST, Directory of Open Access Journal (DAOJ), Google Scholar, Journal Seek, Index Copernicus, GEORGETOWN UNIVERSITY LIBRARY, Ulrichsweb, BASE, WorldCat and Powered by Microsoft Research.

5. **Rajdeep Chakraborty**, Debajyoti Guha and J. K. Mandal, “A Block Cipher Based Cryptosystem Through Forward Backward Overlapped Modulo Arithmetic Technique (FBOMAT)”, published in International Journal of Engineering & Science Journal (**IJESR**), **ISSN 2277 – 2685**, accepted & published in Volume 2 – Issue 5 (**May 2012**) issue, pp-349 – 360, indexed by Index Copernicus, Open J Gate, Ulrichs Web, Google Scholar.

6. Debajyoti Guha, **Rajdeep Chakraborty** and Abhirup Sinha “A Block Cipher Based Cryptosystem Through Modified Forward Backward Overlapped Modulo Arithmetic Technique (MFBOMAT)”, published in IOSR Journal of Computer Engineering (**IOSR–JCE**), **e-ISSN: 2278-0661, p-ISSN: 2278-8727**, accepted & published in Volume 13, Issue 1 (**Jul. - Aug. 2013**) issue, PP 138-146, indexed by NASA, Cross Ref, Arxiv.org, Cabell’s, Index Copernicus, EBSCO Host, Ulrichs Web, Google Scholar, ANED, Jour Informatics.

7. **Rajdeep Chakraborty**, Sibendu Biswas and JK Mandal “Modified Rabin Cryptosystem through Advanced Key Distribution System”, published in IOSR Journal of Computer Engineering (**IOSR–JCE**), **e-ISSN: 2278-0661, p-ISSN: 2278-8727**, accepted & published in Volume 16, Issue 2 Ver XII (**Mar. - Apr. 2014**) issue, PP 01-07, indexed by NASA, Cross Ref, Arxiv.org, Cabell’s, Index Copernicus, EBSCO Host, Ulrichs Web, Google Scholar, ANED, Jour Informatics.

8. **Rajdeep Chakraborty**, Santanu Basak and JK Mandal “An FPGA Based Crypto Processor through Triangular Modulo Arithmetic Technique (TMAT)”, published in **International Journal of Multidisciplinary in Cryptology and Information Security (IJMCIS) ISSN 2320 –2610**, accepted & published in **Volume 3, No.3 (May – June 2014)** issue, PP 14-20, indexed by Google scholar, Cite Seer, getCited, .docstoc, Scribd, Ulrich Web, Index Copernicus, Microsoft Academics, New Jour, DOAJ.

9. **Rajdeep Chakraborty**, Avishek Datta and J. K. Mandal, “Modified Recursive Modulo 2ⁿ and Key Rotation Technique (MRMKRT)”, published in International Journal of Engineering & Science Research (**IJESR**), **ISSN 2277 – 2685**, accepted & published in Volume 5 – Issue 2 (**February 2015**) issue, pp-76 – 81, indexed by Index Copernicus, Open J Gate, Ulrichs Web, Google Scholar.

10. **Rajdeep Chakraborty**, Avishek Datta and JK Mandal “Secure Encryption Technique (SET): A Private Key Crypto System”, published in **International Journal of Multidisciplinary in Cryptology and Information Security (IJMCIS) ISSN 2320 –2610**, accepted & published in **Volume 4, No.1 (January – February 2015)** issue, PP 10-13, indexed by Google scholar, Cite Seer, getCited, .docstoc, Scribd, Ulrich Web, Index Copernicus, Microsoft Academics, New Jour, DOAJ.

International Conferences

1. S. Sinha, J. K. Mandal and **R. Chakraborty**, “A Microprocessor-based Block Cipher through Overlapped Modulo Arithmetic Technique (OMAT)”, published in 12th International Conference on Advance Computing and Communication (**ADCOM 2004**), held on December 15-18, 2004 at Ahmedabad, INDIA, organized and sponsored by Advance Computing and Communication Society (ACS), IEEE Gujrat Section and Computer Society of India (CSI) Ahmedabad Chapter, and published by Allied Publishers Pvt. Limited, Mumbai, India, ISBN 81-7764-717-2, pp 276–280.

2. **Rajdeep Chakraborty** and J.K. Mandal, “A Microprocessor-Based Block Cipher through Rotational Addition Technique (RAT)”, **published & presented** in 9th International Conference on Information Technology (**ICIT 2006**), held on 18-21 December 2006, at Bhubaneswar, India, organized and sponsored by IEEE Computer Society, IEEE, Orissa Information Technology Society (OITS), Institute of Technical Education and Research (ITER), New Jersey Institute of Technology (NJIT), Satyam Computers Ltd. and IEEE New jersey Section, and published by IEEE Computer Society Conference Publishing Services, ISBN-10: 0-7695-2635-7/06, ISBN-13: 978-0-7695-2635-5, pp 155–159.

3. **Rajdeep Chakraborty** and J.K. Mandal, “An Approach Towards Digital Content Protection Through Two Pass Replacement Technique (TPRT)”, published in First International Conference on Information Technology (**INTL-INFOTECH 2007**), held on March 19-21, 2007 at Haldia, INDIA, organized and sponsored by Department of Computer Science and Informatics, Haldia Institute of Technology (HIT), Technical Education Quality Improvement Program (TEQIP), Computer Society of India (CSI) (Kolkata), TEQIP network Partners, University of Calcutta (Kolkata), Govt. College of Engineering and Ceramic Technology (Kolkata), and published by Vitasta Publishing Pvt. Ltd., New Delhi, India, ISBN 81-89766-74-0, ISSN 0973-6824,pp 62–65.

4. **Rajdeep Chakraborty** and J.K. Mandal, “A Microprocessor-Based Stream Cipher Through Stream Addition Technique (SAT)”, published in International Conference on Systemics, Cybernetics and Informatics (**ICSCI 2009**), held on January 07-10, 2009 at Hyderabad, India, organized, sponsored and, published by Pentagram Research Center Pvt. Ltd. Hyderabad, India, Volume 1 of 2, pp 41–45.

5. **Rajdeep Chakraborty** and J.K. Mandal, “A Microprocessor-based Stream Cipher through Stream Parity Technique (SPT)”, **published & presented** in First International Conference on Computer, Communication, Control and Information Technology (**C³IT 2009**), held on 06-07 February, 2009 at Hoogly, West Bengal, INDIA, organized and sponsored by Academy of Technology (AOT), IEEE Leos Calcutta Chapter, IEEE EDS Calcutta Chapter, All India Council for Technical Education (AICTE), Indian Space Research Organization (ISRO), and CSIR, and published by MACMILLAN PUBLISHERS INDIA LTD Advanced Research Series, New Delhi, India, ISBN 10: 0230-63759-0, ISBN 13: 978-0230-63759-7, pp 417–423.

6. **Rajdeep Chakraborty** and J.K. Mandal, “Ensuring e-Security through Microprocessor-Based Recursive Transposition Technique (RTT)”, **published & presented** in 12th International Conference on Information Technology (**ICIT 2009**), held on December 21-24, 2009, at Bhubaneswar, India, organized and sponsored by IEEE, IEEE Computer Society, Orissa Information Technology Society (OITS), Temple City Institute of Technology and Engineering (TITE), Silicon Institute of Technology, IIIT, NIST, OCAC and Techno India Group (Kolkata), and published by Tata McGraw Hill Education Private Limited, New Delhi, India, ISBN-10: 0-07-068104-0, ISBN-13: 978-0-07-068104-2, pp 66–69.

7. **Rajdeep Chakraborty** and J. K. Mandal, “An RTL Based Design & Implementation of Block Cipher through Oriented Encryption Technique (OET)”, **published & presented** in International Conference on Computing and Systems (**ICCS 2010**), held on November 19-20, 2010, at Burdwan, West Bengal, India, organized and published by Department of Computer Science, The University of Burdwan, Burdwan – 713104, West Bengal, India, ISBN – 93-80813-01-5, pp 335 – 338.

8. **Rajdeep Chakraborty**, Sridipta Misra, Sunit Kumar Agarwal, Vineet Khemka, Sonam Agarwal and J. K. Mandal, “Efficient Hardware Realization of Triple Data Encryption Standard (TDES) Algorithm using Spartan-3E FPGA”, **published & presented** in International Conference on Issues and Challenges in Networking, Intelligence and Computing (**ICNICT 2011**), held on 2-3 September, 2011, at Ghaziabad, India, organized and sponsored by Department of Computer Science and Engineering, Krishna Institute of Engineering and Technology (KIET), India, Department of Science & Technology, Govt. of India, New Delhi, Computer Society of India (Ghaziabad Chapter), International Neural Network Society (India Regional Chapter), International Journal of Advanced Research in Computer Science and Ubiquitous Computing and Communication Journal, and published by Nandani Prakashan Pvt. Ltd., New Delhi, India, ISBN – 978-93-81126-27-1, pp 60 – 63.

9. Avishek Datta, **Rajdeep Chakraborty** and J.K. Mandal, “The CRYPTSTER: A Private Key Crypto System”, **published & presented** in 2015 IEEE International Conference on Computer Graphics, Vision and Information Security (**IEEE CGVIS 2015**) **IEEE Conference Record number: #36759**, held on November 02-03, 2015, at Bhubaneswar, India, organized and sponsored by IEEE Kolkata Section, KIIT University, Bhubaneswar, Odisha India and published in IEEE XPLORE CFP 15C89-ART ISBN: 978-1-4673-7437-8, CD-ROM CFP 15C89-CDR ISBN: 978-1-4673-7436-1, pp 35–37.

Papers Presented

1. **Rajdeep Chakraborty** and J.K. Mandal, “A Microprocessor-Based Block Cipher through Rotational Addition Technique (RAT)”, **published and presented** in 9th International Conference on Information Technology (**ICIT 2006**), held on 18-21 December 2006, at Bhubaneswar, India, organized and sponsored by IEEE Computer Society, IEEE, Orissa Information Technology Society (OITS), Institute of Technical Education and Research (ITER), New Jersey Institute of Technology (NJIT), Satyam Computers Ltd. and IEEE New Jersey Section, and published by IEEE Computer Society Conference Publishing Services, ISBN-10: 0-7695-2635-7/06, ISBN-13: 978-0-7695-2635-5, pp 155–159.
2. **Rajdeep Chakraborty** and J.K. Mandal, “A Microprocessor-based Stream Cipher through Stream Parity Technique (SPT)”, **published and presented** in First International Conference on Computer, Communication, Control and Information Technology (**C³IT 2009**), held on 06-07 February, 2009 at Hoogly, West Bengal, INDIA, organized and sponsored by Academy of Technology (AOT), IEEE Leos Calcutta Chapter, IEEE EDS Calcutta Chapter, All India Council for Technical Education (AICTE), Indian Space Research Organization (ISRO), and CSIR, and published by MACMILLAN PUBLISHERS INDIA LTD Advanced Research Series, New Delhi, India, ISBN 10: 0230-63759-0, ISBN 13: 978-0230-63759-7, pp 417–423.
3. **Rajdeep Chakraborty** and J.K. Mandal, “Ensuring e-Security through Microprocessor-Based Recursive Transposition Technique (RTT)”, **published and presented** in 12th International Conference on Information Technology (**ICIT 2009**), held on December 21-24, 2009, at Bhubaneswar, India, organized and sponsored by IEEE, IEEE Computer Society, Orissa Information Technology Society (OITS), Temple City Institute of Technology and Engineering (TITE), Silicon Institute of Technology, IIIT, NIST, OCAC and Techno India Group (Kolkata), and published by Tata McGraw Hill Education Private Limited, New Delhi, India, ISBN-10: 0-07-068104-0, ISBN-13: 978-0-07-068104-2, pp 66–69.

4. **Rajdeep Chakraborty** and J. K. Mandal, “An RTL Based Design and Implementation of Block Cipher through Oriented Encryption Technique (OET)”, **published and presented** in International Conference on Computing and Systems (**ICCS 2010**), held on November 19-20, 2010, at Burdwan, West Bengal, India, organized and published by Department of Computer Science, The University of Burdwan, Burdwan – 713104, West Bengal, India, ISBN – 93-80813-01-5, pp 335 – 338.

5. **Rajdeep Chakraborty**, Sridipta Misra, Sunit Kumar Agarwal, Vineet Khemka, Sonam Agarwal and J. K. Mandal, “Efficient Hardware Realization of Triple Data Encryption Standard (TDES) Algorithm using Spartan-3E FPGA”, **published and presented** in International Conference on Issues and Challenges in Networking, Intelligence and Computing (**ICNICT 2011**), held on 2-3 September, 2011, at Ghaziabad, India, organized and sponsored by Department of Computer Science and Engineering, Krishna Institute of Engineering and Technology (KIET), India, Department of Science and Technology, Govt. of India, New Delhi, Computer Society of India (Ghaziabad Chapter), International Neural Network Society (India Regional Chapter), International Journal of Advanced Research in Computer Science and Ubiquitous Computing and Communication Journal, and published by Nandani Prakashan Pvt. Ltd., New Delhi, India, ISBN – 978-93-81126-27-1, pp 60 – 63.

6. Avishek Datta, **Rajdeep Chakraborty** and J.K. Mandal, “The CRYPTSTER: A Private Key Crypto System”, **published & presented** in 2015 IEEE International Conference on Computer Graphics, Vision and Information Security (**IEEE CGVIS 2015**) **IEEE Conference Record number: #36759**, held on November 02-03, 2015, at Bhubaneswar, India, organized and sponsored by IEEE Kolkata Section, KIIT University, Bhubaneswar, Odisha India and published in IEEE XPLORE CFP 15C89-ART ISBN: 978-1-4673-7437-8, CD-ROM CFP 15C89-CDR ISBN: 978-1-4673-7436-1, pp 35–37.

Contents of Thesis

Topic	Page no.
Acknowledgements	vii
List of Publications	ix
Papers Presented	xv
List of Figures	xxiii
List of Tables	xxxix
List of Abbreviations	xxxvii
List of Symbols	xli
Abstract	xliii
Chapter 1: Introduction	1
1.1 Introduction	3
1.2 Literature Survey	33
1.3 Problem Domain	59
1.4 Proposed Methodology	61
1.5 Salient Features of the Thesis	64
1.6 Organization of the Thesis	65
Section I: Microprocessor Based Solutions	71
Chapter 2: Modified Recursive Modulo-2ⁿ and Key Rotation Technique (MRMKRT)	73
2.1 Introduction	75
2.2 The Algorithm of MRMKRT	76
2.3 Example	78
2.4 The Modulo Addition	81
2.5 Key Generation	81
2.5.1 Example of Key Generation	82
2.6 Analysis	83
2.7 Implementation	85
2.7.1 Algorithm of routine ‘addblocks’	85
2.7.2 Algorithm of routine ‘rot’	86
2.7.3 Algorithm of routine ‘store’	87
2.7.4 Algorithm of routine ‘subblocks’	88

2.7.5 Main Program of MRMKRT	89
2.8 Results and Comparisons	90
2.8.1 Implementation Based Results	90
2.8.2 Frequency Distribution Analysis	92
2.8.3 Non-Homogeneity Test	93
2.8.4 Time Complexity Analysis	95
2.8.5 The Avalanche Test	97
2.9 Discussions	98
Chapter 3: Recursive Transposition Technique (RTT)	99
3.1 Introduction	101
3.2 The Algorithm of RTT	102
3.2.1 The Encryption Process	103
3.2.2 The Decryption Process	104
3.2.3 Example	104
3.3 Key Generation Process	105
3.4 Analysis	107
3.5 Implementation	108
3.5.1 Routine ‘save’	109
3.5.2 Routine ‘b’	109
3.5.3 Routine ‘a’	110
3.5.4 Routine ‘c’	111
3.5.5 Routine ‘prg’	111
3.5.6 Routine ‘outp’	112
3.5.7 Routine ‘supply’	113
3.5.8 Algorithm of the Main Program for RTT Encoder ...	113
3.6 Results and Comparisons	114
3.6.1 Implementation Based Results	114
3.6.2 Frequency Distribution Graph	116
3.6.3 Non-Homogeneity Test	118
3.6.4 Time Complexity Analysis	119
3.6.5 The Avalanche Ratio Test	121
3.7 Discussions	122

Section II: FPGA-Based Solutions	123
Chapter 4: Two Pass Replacement Technique (TPRT)	125
4.1 Introduction	127
4.2 The Algorithm of TPRT	128
4.2.1 The Encryption Process	129
4.2.2 The Decryption Process	130
4.3 Example	130
4.4 Implementation and Key Generation	132
4.4.1 Key Generation	135
4.5 Analysis	138
4.6 Results and Simulations	140
4.6.1 RTL Simulation Based Results	140
4.6.2 Frequency Distribution Graph	142
4.6.3 The Non-Homogeneity Test	144
4.6.4 The Time Complexity Analysis	145
4.6.5 The Avalanche Ratio	147
4.7 Discussions	148
Chapter 5: Triangular Modulo Arithmetic Technique (TMAT)	149
5.1 Introduction	151
5.2 The Algorithm of TMAT	152
5.2.1 The Modulo Addition	155
5.3 Example	155
5.3.1 The Encryption Process	155
5.3.2 The Decryption Process	158
5.4 Implementation and Key Generation	160
5.4.1 Key Generation	163
5.4.1.1 Example of Key Generation	164
5.4.1.1.1 Key Generation for Phase 1 and Phase 3 ...	164
5.4.1.1.2 Key Generation for Phase 2	165
5.5 Analysis	166
5.6 Results and Simulations	167
5.6.1 RTL Simulation Based Result	167

5.6.2 The Frequency Distribution Graph	170
5.6.3 The Non-Homogeneity Test	172
5.6.4 The Time Complexity Analysis	174
5.6.5 The Avalanche Ratio Test	176
5.7 Discussions	177
Chapter 6: Recursively Oriented Block Addition and Substitution Technique (ROBAST)	179
6.1 Introduction	181
6.2 The Algorithm of ROBAST	182
6.3 Example	184
6.4 Implementation and Key Generation	185
6.4.1 The Key Generation Process of ROBAST	188
6.4.2 An Example of Key Generation	189
6.4.3 Modulo Addition Used in ROBAST	190
6.5 Analysis	190
6.6 Results and Simulations	191
6.6.1 RTL Simulation Based Result	191
6.6.2 The Frequency Distribution Graph	194
6.6.3 The Non-Homogeneity Test	196
6.6.4 The Time Complexity Analysis	198
6.6.5 The Avalanche Ratio Test	201
6.7 Discussions	202
Chapter 7: Shuffle – Rotational Addition Technique (SRAT)	203
7.1 Introduction	205
7.2 The Algorithm of SRAT	206
7.3 Example	209
7.4 Implementation and Key Generation	211
7.4.1 Key Generation	214
7.4.2 Example of Key Generation	216
7.5 Analysis	216
7.6 Results and Simulations	217
7.6.1 RTL Simulation Based Result	218
7.6.2 The Frequency Distribution Graph	221

7.6.3 The Non-Homogeneity Test	223
7.6.4 The Time Complexity Analysis	225
7.6.5 The Avalanche Ratio Test	228
7.7 Discussions	228
Chapter 8: Triple Sagacious Vanquish (TSV)	231
8.1 Introduction	233
8.2 The Algorithm of TSV	234
8.2.1 Modes of Operation	234
8.2.2 Encryption	236
8.2.3 Decryption	240
8.3 Example	241
8.4 Implementation and Key Generation	242
8.4.1 Cipher Block Chaining (CBC) Mode	242
8.4.2 Round Key Generation	244
8.5 Analysis	246
8.6 Results and Simulations	246
8.6.1 RTL Simulation Based Result	247
8.6.2 The Frequency Distribution Graph	250
8.6.3 The Non-Homogeneity Test	253
8.6.4 The Time Complexity Analysis	255
8.6.5 The Avalanche Ratio Test	257
8.7 Discussions	258
Chapter 9: Modified Forward Backward Overlapped	259
Modulo Arithmetic Technique (MFBOMAT)	259
9.1 Introduction	261
9.2 The Algorithm of MFBOMAT	262
9.2.1 The Modulo Addition	264
9.3 Example	264
9.3.1 The Encryption	264
9.3.2 The Decryption	266
9.4 Implementation and Key Generation	268
9.4.1 The Key Generation Process of MFBOMAT	270
9.4.2 An Example of Key Generation	272

9.5 Analysis	273
9.6 Results and Simulations	273
9.6.1 RTL Simulation Based Result	273
9.6.2 The Frequency Distribution Graph	278
9.6.3 The Non-Homogeneity Test	281
9.6.4 The Time Complexity Analysis	283
9.6.5 The Avalanche Ratio Test	286
9.7 Discussions	287
Chapter 10: Proposed Models	289
10.1 Proposed Models	291
10.2 The Proposed Model for Microprocessor-Based Solutions ..	291
10.2.1 Results and Comparisons	292
10.2.1.1 Implementation based result	293
10.2.1.2 Frequency Distribution Graph	294
10.2.1.3 Non-Homogeneity Test	296
10.2.1.4 Time Complexity Analysis	297
10.2.1.5 The Avalanche Ratio Test	298
10.3 The Proposed Model for FPGA-Based Solutions	299
10.3.1 Results and Simulations	302
10.3.1.1 RTL Simulation Based Result	302
10.3.1.2 The Frequency Distribution Graph	303
10.3.1.3 The Non-Homogeneity Test	305
10.3.1.4 The Time Complexity Analysis	306
10.3.1.5 The Avalanche Ratio Test	308
10.4 Discussions	309
Chapter 11: Conclusions	311
11.1 Conclusive Discussions	313
11.2 The Future Work	320
References	323

List of Figures

Figure No.	Figure Caption	Page No.
1.1	Taxonomy of cryptography	4
1.2	Taxonomy of symmetric key cryptography	7
1.3	Taxonomy of public key cryptography	9
1.4	Intel 8085 microprocessor	11
1.5	Intel 8085 microprocessor architecture	12
1.6	Internal structure of a generic FPGA	16
1.7	Simplified illustration of a logic cell	17
2.1	Modified recursive modulo-2 ⁿ and key rotation technique (MRMKRT)	76
2.2	First round of MRMKRT	78
2.3	Second round of MRMKRT	79
2.4	Third round of MRMKRT	80
2.5	Fourth round of MRMKRT	80
2.6	Round v/s iteration in MRMKRT	82
2.7	MRMKRT encryption and decryption algorithm	89
2.8	Frequency distribution of ASCII characters in the RSA encrypted file	92
2.9	Frequency distribution of source file and MRMKRT encrypted files	93
2.10	Chi-Square values for MRMKRT and RSA encrypted files	94
2.11	Encryption and decryption time of MRMKRT and RSA	96
3.1	RTT encoder	101
3.2	Block diagram of Recursive Transposition Technique (RTT)	102
3.3	Algorithmic flow in RTT	104
3.4	Graphical representation of comparisons of MRMKRT and RTT	115
3.5	The frequency distribution graph of RSA encrypted file	116
3.6	The frequency distribution graph of source file and MRMKRT encrypted file	117
3.7	Frequency distribution graph of RTT encrypted file	117

Figure No.	Figure Caption	Page No.
3.8	Graphical comparisons of Chi-Square values of RTT, MRMKRT and RSA	119
3.9	Pictorial representation of time graph of RTT, MRMKRT and RSA encryption	120
3.10	Pictorial representation of time graph of RTT, MRMKRT and RSA decryption	121
4.1	Block diagram of Two Pass Replacement Technique (TPRT)	129
4.2	Top-level design of TPRT	133
4.3	Top-level RTL design of TPRT	134
4.4	Top-level RTL design of TPRT fixed size key generation	136
4.5	Top-level RTL design of TPRT variable size key generation	138
4.6	RTL diagram of RSA	140
4.7	RTL diagram for Spartan 3E of the proposed TPRT	141
4.8	The frequency distribution graph of source, RSA encrypted and TPRT encrypted files	143
4.9	Graphical representation of Chi-Square values of RSA and TPRT	145
4.10	Graphical comparisons of encryption and decryption time of TPRT and RSA	147
5.1	Block diagram of TMAT	152
5.2	Triangle formation	153
5.3	Encryption example of phase 1 in TMAT	156
5.4	Encryption example of phase 2 in TMAT	157
5.5	Encryption example of phase 3 in TMAT	158
5.6	Decryption example of phase 1 in TMAT	158
5.7	Decryption example of phase 2 in TMAT	159
5.8	Decryption example of phase 3 in TMAT	160
5.9	Top level design of TMAT	161
5.10	Top level RTL design of TMAT	161
5.11	Top level RTL for round key generation for TMAT	166
5.12	RTL diagram of RSA	168

Figure No.	Figure Caption	Page No.
5.13	Spartan 3E RTL diagram of TPRT	168
5.14	Spartan 3E RTL diagram of TMAP	169
5.15	The frequency distribution graph of source, RSA encrypted and TPRT encrypted files	171
5.16	Frequency distribution graph of TMAP encrypted file	172
5.17	Graphical representation Chi-Square value of TMAP, RSA and TPRT	173
5.18	Pictorial representation of time complexity analysis of TMAP, RSA and TPRT	175
6.1	Block diagram of ROBAST	183
6.2	ROBAST entity and its function	186
6.3	Top level RTL design of ROBAST	187
6.4	Graphical representation of key generation of ROBAST	188
6.5	Session key generation of ROBAST	190
6.6	RTL diagram of RSA	191
6.7	Spartan 3E RTL diagram of TPRT	192
6.8	Spartan 3E RTL diagram of TMAP	192
6.9	Spartan 3E schematic of ROBAST	192
6.10	Frequency distribution graph of source, RSA encrypted and TPRT encrypted files	195
6.11	Frequency distribution graph of TMAP and ROBAST encrypted files	196
6.12	Pictorial representation of Chi-Square values of ROBAST, RSA, TPRT and TMAP	197
6.13	Pictorial representation of encryption time of ROBAST, RSA, TMAP and TPRT	200
6.14	Pictorial representation of decryption time of ROBAST, RSA, TMAP and TPRT	200
7.1	Block diagram of Shuffle-RAT	206
7.2	Graphical representation of number of iterations to obtain source stream using modulo-addition	209

Figure No.	Figure Caption	Page No.
7.3	Top-level hardware architecture for Shuffle-RAT	211
7.4	Full adder at stage i with P_i and G_i	213
7.5	4-bit modulo carry look-ahead adder implementation details	214
7.6	Graphical representation of round v/s iteration	215
7.7	Session key generation for SRAT	216
7.8	RTL diagram of RSA	218
7.9	Spartan 3E RTL diagram of TPRT	218
7.10	Spartan 3E RTL diagram of TMAT	219
7.11	Spartan 3E schematic of ROBAST	219
7.12	Spartan 3E RTL schematic of the main controller module of Shuffle-RAT	219
7.13	Frequency distribution graph of source, RSA encrypted and TPRT encrypted files	222
7.14	Frequency distribution graph of TMAT and ROBAST encrypted files	222
7.15	Frequency distribution graph of SRAT encrypted files	223
7.16	Comparison of Chi-Square values for ROBAST, RSA, TMAT, TPRT and SRAT	225
7.17	Pictorial representation of encryption time against file size	227
7.18	Pictorial representation of decryption time against file size	227
8.1	Overview of the TSV	233
8.2	The Cipher Block Chaining (CBC) mode for encryption in TSV	234
8.3	The Cipher Block Chaining (CBC) mode for decryption in TSV	234
8.4	TSV encryption overview	235
8.5	n-BIT level structure (encryption) for TSV	236
8.6	n-bit far swap function for TSV	237
8.7	n-bit near swap function for TSV	237
8.8	Expansion function for encryption of TSV	237
8.9	TSV decryption process	238
8.10	n-bit level structure (decryption) of TSV	238
8.11	Expansion function for decryption of TSV	239

Figure No.	Figure Caption	Page No.
8.12	Top level algorithm for CBC encryption of TSV	243
8.13	Top level algorithm for CBC decryption of TSV	243
8.14	Top level VHDL module for round key generation of TSV	244
8.15	Top level entity of round key generation of TSV	244
8.16	RTL diagram of RSA	247
8.17	Spartan 3E RTL diagram of TPRT	247
8.18	Spartan 3E RTL diagram of TMAT	248
8.19	Spartan 3E schematic of ROBAST	248
8.20	Spartan 3E RTL schematic of the main controller module of Shuffle-RAT	248
8.21	Spartan 3E RTL diagram of TSV	249
8.22	Frequency distribution graph of source, RSA encrypted and TPRT encrypted files	251
8.23	Frequency distribution graph of TMAT and ROBAST encrypted files	252
8.24	Frequency distribution graph of SRAT encrypted files	252
8.25	Frequency distribution graph of TSV encrypted files	253
8.26	Pictorial representation of Chi-Square values	253
8.27	Pictorial representation of encryption time against file size	256
8.28	Pictorial representation of decryption time against file size	257
9.1	Block diagram of MFBOMAT	263
9.2	MFBOMAT entity and its function	268
9.3	Top level RTL design of MFBOMAT	269
9.4	Graphical representation of key generation of MFBOMAT	271
9.5	Session key generation of MFBOMAT	272
9.6	RTL diagram of RSA	274
9.7	Spartan 3E RTL diagram of TPRT	274
9.8	Spartan 3E RTL diagram of TMAT	274
9.9	Spartan 3E schematic of ROBAST	275
9.10	Spartan 3E RTL schematic of the main controller module of Shuffle-RAT	275

Figure No.	Figure Caption	Page No.
9.11	Spartan 3E RTL diagram of TSV	275
9.12	Spartan 3E RTL diagram of MFBOMAT	276
9.13	Frequency distribution graph of source, RSA encrypted and TPRT encrypted files	279
9.14	Frequency distribution graph of TMAT and ROBAST encrypted files	280
9.15	Frequency distribution graph of SRAT encrypted files	280
9.16	Frequency distribution graph of TSV encrypted files	280
9.17	Frequency distribution graph of MFBOMAT encrypted files	281
9.18	Pictorial representation of Chi-Square values against file size	282
9.19	Pictorial representation of encryption time against file size	284
9.20	Pictorial representation of decryption time against file size	285
10.1	Proposed model for microprocessor-based solutions	291
10.2	Graphical representation of implementation based results of the model, MRMKRT and RTT	294
10.3	Frequency distribution of source file	295
10.4	The frequency distribution graph of RSA encrypted file	295
10.5	Graphical representation of frequency distribution of proposed model (encrypted file)	295
10.6	Graphical representation of Chi-Square for RSA and proposed model	296
10.7	Pictorial representation of encryption time	298
10.8	Graphical representation of decryption time	298
10.9	Proposed model for FPGA-based solutions	300
10.10	RTL diagram of RSA	302
10.11	Spartan 3E RTL diagram of proposed model	302
10.12	Frequency distribution of source file	304
10.13	The frequency distribution graph of RSA encrypted file	304
10.14	The frequency distribution graph of proposed model	305
10.15	Graphical representation of Chi-Square value of RSA and proposed model	306

Figure No.	Figure Caption	Page No.
10.16	Pictorial representation of encryption time of RSA and proposed model	307
10.17	Pictorial representation of decryption time of RSA and proposed model	308
11.1	Graphical representation of comparisons of MRMKRT, RTT and RSA	316
11.2	Pictorial representation of HDL synthesis report of net-list generation	318
11.3	Pictorial representation of HDL synthesis report of timing summary	320

List of Tables

Table No.	Table Caption	Page No.
2.1	Representation of number of iterations in each round by bits	81
2.2	Plaintext and ciphertext pair in hex for single iteration of MRMKRT	83
2.3	Implementation based results of MRMKRT	91
2.4	Chi-Square values of RSA and MRMKRT	94
2.5	The time complexity analysis of MRMKRT and RSA	95
2.6	Avalanche ratio values of MRMKRT and RSA	97
3.1	Number of iteration against block sizes	105
3.2	Key generation for variable block length technique, RTT	106
3.3	Comparisons of MRMKRT and RTT	115
3.4	Chi-Square values of RSA, MRMKRT and RTT	118
3.5	Comparisons of time complexity analysis of RTT, MRMKRT and RSA	120
3.6	Comparisons of avalanche ratio of RTT, MRMKRT and RSA	122
4.1	Encryption process of TPRT	131
4.2	Decryption process of TPRT	132
4.3	Representation of number of iterations in each round by bits for 2^n	135
4.4	Key generation for variable block length technique for TPRT	137
4.5	Plaintext and equivalent Hex code	139
4.6	Hex code and equivalent ciphertext	139
4.7	HDL synthesis report (netlist generation of RSA and TPRT)	141
4.8	HDL Synthesis Report (Timing Summary of RSA and TPRT) ...	141
4.9	Chi-Square values of RSA and TPRT	144
4.10	Comparisons of time complexity analysis of TPRT and RSA	146
4.11	Comparisons of avalanche ratio of TPRT and RSA	148
5.1	Target block selection using selection key	164
5.2	Key distribution for the Triangular Technique	165
5.3	Key generation for MAT	165

Table No.	Table Caption	Page No.
5.4	HDL synthesis report (Netlist generation of RSA, TPRT and TMAT)	169
5.5	HDL synthesis report (Timing summary of RSA, TPRT and TMAT)	170
5.6	Chi-Square values and degree of freedom of TMAT, RSA and TPRT	173
5.7	The time complexity analysis of TMAT, RSA and TPRT	175
5.8	The avalanche ratio of RSA, TPRT and TMAT	176
6.1	An encryption example of ROBAST	184
6.2	Example of decryption in ROBAST	185
6.3	Representation of number of iterations in each round by bits, the key generation for ROBAST	188
6.4	HDL synthesis report (Netlist generation of RSA, TPRT, TMAT and ROBAST)	193
6.5	HDL synthesis report (Timing summary of RSA, TPRT, TMAT and ROBAST)	194
6.6	Chi-Square values of ROBAST, RSA, TPRT and TMAT	197
6.7	Degree of freedom of ROBAST, RSA, TPRT and TMAT	298
6.8	Comparison of encryption time of ROBAST, RSA, TMAT and TPRT	199
6.9	Comparison of decryption time of ROBAST, RSA, TMAT and TPRT	199
6.10	Comparison of avalanche ratio of ROBAST, RSA, TPRT and TMAT encrypted files	201
7.1	Number of iteration to regenerate source stream using modulo-addition	208
7.2	Encryption process of SRAT	209
7.3	Decryption process of SRAT	210
7.4	Representation of number of iterations in each round in SRAT ...	215
7.5	HDL synthesis report (Netlist generation of RSA, TPRT, TMAT, ROBAST and SRAT)	220

Table No.	Table Caption	Page No.
7.6	HDL synthesis report (Timing summary of RSA, TPRT, TMAT, ROBAST and SRAT)	220
7.7	Comparison of Chi-Square values of ROBAST, RSA, TPRT, TMAT and SRAT	224
7.8	Comparison of degree of freedom of ROBAST, RSA, TPRT, TMAT and SRAT	224
7.9	Comparison encryption time of ROBAST, RSA, TMAT, TPRT and SRAT	226
7.10	Comparison of decryption time of ROBAST, RSA, TMAT, TPRT and SRAT	226
7.11	Comparison of avalanche ratio of ROBAST, RSA, TPRT, TMAT and SRAT encrypted files	228
8.1	TSV encryption using 2-bit level with 16-bit plaintext and 8-bit Key	241
8.2	HDL synthesis report (Netlist generation of RSA, TPRT, TMAT, ROBAST, SRAT and TSV)	249
8.3	HDL synthesis report (Timing summary of RSA, TPRT, TMAT, ROBAST, SRAT and TSV)	250
8.4	Comparison of Chi-Square values of ROBAST, RSA, TPRT, TMAT, SRAT and TSV	254
8.5	Comparison of degree of freedom of ROBAST, RSA, TPRT, TMAT, SRAT and TSV	254
8.6	Comparison of encryption time of ROBAST, RSA, TMAT, TPRT, SRAT and TSV	255
8.7	Comparison of decryption time of ROBAST, RSA, TMAT, TPRT, SRAT and TSV	256
8.8	Comparison of avalanche ratio of ROBAST, RSA, TPRT, TMAT, SRAT and TSV encrypted files	258
9.1	Representation of number of iterations in each round by bits, the key generation for MFBOMAT	270

Table No.	Table Caption	Page No.
9.2	HDL synthesis report (Netlist generation of RSA, TPRT, TMAT, ROBAST, SRAT, TSV and MFBOMAT)	276
9.3	HDL synthesis report (Timing summary of RSA, TPRT, TMAT, ROBAST, SRAT and TSV)	277
9.4	Comparison of Chi-Square values of ROBAST, RSA, TPRT, TMAT, SRAT, TSV and MFBOMAT	282
9.5	Comparison of degree of freedom of ROBAST, RSA, TPRT, TMAT, SRAT, TSV and MFBOMAT	283
9.6	Comparison of encryption time of ROBAST, RSA, TMAT, TPRT, SRAT, TSV and MFBOMAT	284
9.7	Comparison of decryption time of ROBAST, RSA, TMAT, TPRT, SRAT and TSV	285
9.8	Comparison of avalanche ratio of ROBAST, RSA, TPRT, TMAT, SRAT and TSV encrypted files	286
10.1	Implementation based results of MRMKRT, RTT and proposed model	293
10.2	Comparisons of Chi-Square values of RSA and proposed model .	296
10.3	Comparison of time complexity analysis of RSA and proposed model	297
10.4	Comparison of avalanche ratio of RSA and proposed model	299
10.5	HDL synthesis report (Netlist generation of RSA and proposed model)	303
10.6	HDL synthesis report (Timing summary of RSA and proposed model)	304
10.7	Chi-Square values of RSA and proposed model	305
10.8	Encryption and decryption time of RSA and proposed model	307
10.9	The avalanche ratio of RSA and proposed model	308
11.1	Characteristics of microprocessor-based solutions	314
11.2	Comparisons of MRMKRT, RTT and RSA	315
11.3	Characteristics of FPGA-based solutions	317

Table No.	Table Caption	Page No.
11.4	HDL synthesis report (Netlist generation of RSA, TPRT, TMAT, ROBAST, SRAT, TSV and MFBOMAT)	318
11.5	HDL synthesis report (Timing summary of RSA, TPRT, TMAT, ROBAST, SRAT and TSV)	319

List of Abbreviations


Abbreviation	Interpretation
ADC	Analog-to-Digital Converter
AES	Advance Encryption Standard, A Symmetric Cipher
ASCII	American Standard Code for Information Interchange
ASIC	Application Specific Integrated Circuit
BASIC	Beginners All-assigned Symbolic Instruction Code
BIC	Bit Independent Criteria
BISDN	Broadband ISDN
CAD	Computer Aided Design
CALL, RET, LHLD, SHLD, MOV, ADD, SUB, JMP	8085-Microprocessor Instructions
CBC	Cipher Block Chaining Mode
CFB	Cipher Feedback Mode
CL	Cryptography Language
CLB	Configurable Logic Blocks
CP	Crypto Processor
CPLD	Complex Programmable Logic Device
CRBOCAB	Cascaded Recursive Bitwise Operation and Carry Addition on Blocks
CRCAKR	Cascaded Recursive Carry Addition and Key Rotation
CRKRAB	Cascaded Recursive Key Rotation of a Session Key and Addition of Blocks
CRT	Chinese Remainder Theorem
DAC	Digital-to-Analog Converter
DES	Data Encryption Standard
Df	Degree of freedom
DLL	Delay Locked Loop
DMA	Direct Memory Access
ECB	Electronic Codebook Mode
ECC	Elliptic Curve Cryptography

Abbreviation	Interpretation
ECCP	Elliptic Curve Crypto Processor
EFT	Electronic Fund Transfer
FPAA	Field Programmable Analog Array
FPGA	Field Programmable Gate Array
HDL	Hardware Definition Language
HOLD, HLDA	8085-Microprocessor Signals
IC	Integrated Circuit
IDEA	International Data Encryption Algorithm, A Symmetric Cipher
IEEE	Institute of Electrical and Electronic Engineer
IP	Internet Protocol
IP, IP ⁻¹	Initial Permutation and Inverse Initial Permutation
IPsec	Secure IP Network
ISAKMP	Internet Security Association and Key Management Protocol
ISDN	Integrated Services Digital Network
IV	Initial Vector
KB	Kilo-Bytes
LAN	Local Area Network
LFSR	Linear Feedback Shift Register
LSB	Least Significant Bit
LUT	Look-Up Tables
MAN	Metropolitan Area Network
MB	Mega Bytes
MFBOMAT	Modified Forward Backward Overlapped Modulo Arithmetic Technique
MHz	Mega Hertz
MobileIP	Mobile IP Network
MOS	Metal Oxide Silicon Circuit
MP	Microprocessor
MRMKRT	Modified Recursive Modulo-2 ⁿ and Key Rotation Technique
MSB	Most Significant Bit
NIST	National Institute of Standard and Technology

Abbreviation	Interpretation
OET	Oriented Encryption Technique
OFB	Output Feedback Mode
OMAT	Overlapped Modulo Addition Technique
PIN	Personal Identification Number
PKI	Public Key Infrastructure
PLL	Phase Locked Loop
RAM	Random Access Memory
RAT	Rotational Addition Technique
RISC	Reduced Instruction Set Computer
RKEP	Remote Key Encryption Protocol
RKR	Recursive Key Rotation
ROBAST	Recursively Oriented Block Addition and Substitution Technique
ROM	Read Only Memory
RSA	Rivest-Shamir-Adleman (RSA), A Public Key Cipher
RTL	Register Transfer Logic
RTO	Recursive Transposition Operation
RTT	Recursive Transposition Technique
SAC	Strict Avalanche Criteria
SoC	System on Chip
SP	Stack Pointer
SRAT, S-RAT	Shuffle Rotational Arithmetic Technique
SSL	Secure Socket Layer Protocol
TDES, 3DES	Triple Data Encryption Standard, A Symmetric Cipher
TEA	Tiny Encryption Algorithm
TMAT	Triangular Modulo Arithmetic Technique
TPRT	Two Pass Replacement Technique
TRAP, RST, INTR	8085-Microprocessor Interrupt
TSK-OET	Time-Stamp-Keyed OET
TSV, 3SV	Triple Sagacious Vanquish

Abbreviation	Interpretation
VHDL, VHSICHDL	Very High Speed Integrated Circuit Hardware Definition Language
VLSI	Very Large Scale Integrated Circuit
VPN	Virtual Private Network
WAN	Wide Area Network
WEP	Wired Equivalent Privacy
WIFI	Wide-Fidelity Network, A Wireless Communication
XOR, EXOR	Exclusive-OR Operation

List of Symbols

Sr. No.	Symbol	Interpretation
01		Exclusive-OR Operation
02	=	Variable Assignment
03	<=	Signal Assignment
04	=>	Port Mapping
05	χ , X	Chi-Square value
06	\log_n	Natural Logarithm with Base n
07	$O(n)$	Big-Oh Notation for Time Complexity Analysis
08	Φ	Euler's Totient Function
09	\surd	Optimal Solution

Abstract

It is widely recognized that data security is playing a crucial role in the design of future IT systems. Cryptography is one of the methods to achieve this goal; basic cryptography is implemented in most aspects of the computer world, from emails to personal file. The outmost use of computer and communication systems by industry has increased the risk of theft of proprietary information. Thus cryptography is not only important for person in particular but it is also very much important for industry in general. Many of these IT applications in today's world are realized as embedded systems such as wireless phones, mobile phones, direct to home (DTH) pay-television, mobile internet connectivity to PC and Laptops, audio/video consumer products, digital cinemas, Automatic Teller Machines (ATMs), information kiosk and so on. All modern security protocols use symmetric key as well as public key cryptography algorithms. It is now established that symmetric key cryptography is as important in providing security to these embedded systems than that of public key cryptography. The symmetric key algorithms are very important in providing security for a large quantity of data with faster processing time. In this thesis eight symmetric key techniques have been proposed all are bit level implemented techniques. This thesis work is mainly focused in developing symmetric key cryptography having high degree of acceptance in terms of encryption and decryption time, degree of non-homogeneity, the avalanche effect and frequency distribution.

Now the question is where to implement or what is/are the target device. As the work is for embedded systems, the implementation is targeted on two most widely used devices, Microprocessor and Field Programmable Gate Array (FPGA). It is well known that till today most of the embedded systems are realized through microprocessors and the FPGA is the future of embedded systems. The implementation in hardware device is mainly focused on high throughput, low power consumptions and low computational complexity. Implementation of security protocols in FPGA leads to the achievement of high efficiency as well as cost effectiveness.

Therefore to achieve the above goal, this thesis is mainly divided into two parts, the first parts proposes two novel symmetric key techniques for microprocessor-based systems. The second part of the thesis proposes another novel set of six symmetric key techniques for FPGA-based systems. In this thesis new models are also being proposed.

The two microprocessor based techniques are Modified Recursive Modulo-2ⁿ and Key Rotation Technique (MRMKRT) and Recursive Transposition Technique (RTT). In

MRMKRT the plaintext is considered a block of bits then each block are modulo added replacing the second block after that the whole plaintext block is left circularly rotated. In RTT the whole plaintext is divided into blocks and matrix is formed for each block. Then bit wise XOR operation is performed between two adjacent matrixes, the result replacing the second matrix. After that these matrixes are transformed into blocks and combining all the blocks the resultant ciphertext is formed.

The six FPGA-based techniques have also been proposed and these are Two Pass Replacement Technique (TPRT), Triangular Modulo Arithmetic Technique (TMAT), Recursively Oriented Block Addition and Substitution Technique (ROBAST), Shuffle Rotational Arithmetic Technique (SRAT), Triple Sagacious Vanquish (TSV) and Modified Forward Backward Overlapped Modulo Arithmetic Technique (MFBOMAT).

In TPRT n-bit plaintext is divided into k-number of blocks and each block consists of n/k – bits. Each adjacent block are then XORED replacing the second block, and then again two adjacent blocks are XORED and now replacing the first block. After that combining the blocks n-bit ciphertext is generated. In TMAT n-bit plaintext is divided into blocks, the odd numbers of blocks are encrypted with triangular encryption techniques and even numbers of blocks are encrypted with modulo arithmetic technique. In second phase odd numbers of blocks are encrypted with modulo arithmetic technique and even number of blocks are encrypted with triangular encryption technique. Finally combining all the blocks the n-bit ciphertext is generated. In ROBAST n-bit plaintext is divided into blocks, first block is modulo added with second block, the result replacing the second block and so on. Finally permutation of bits is performed for all the blocks, thus combining the blocks the n-bit ciphertext is generated. In SRAT, during encryption, a butterfly shuffle is applied to the whole source stream, then the source stream is broken down into blocks of fixed size, then the consecutive blocks are modulo added, the result replaces the second block keeping the first block intact, in the next phase the whole block is left circular rotated. Now the blocks are concatenated and again another round of butterfly shuffle is applied. TSV consist of several rounds, first inverse function is applied then 2-bit level, 4-bit level, 8-bit level, 16-bit level, 32-bit level, 64-bit level and 128-bit levels are applied on n-bit plaintext. Finally again inverse function is applied. In MFBOMAT, The original message is considered as a stream of bits, which is then divided into a number of blocks, each containing n bits, where n is any one of 2, 4, 8, 16, 32, 64, 128, 256. The first and last blocks are then added where the modulus of addition is 2^n . The result replaces the last block (say Nth block), first block remaining unchanged (Forward mode). In the next attempt the second and the Nth block (the changed

block) are added and the result replaces the second block (Backward mode). Again the second (the changed block) and the (N-1)th block are added and the result replaces the (N-1)th block (Forward mode). Finally combining the blocks the n-bit ciphertext is generated.

Chapter 1
Introduction

1.1 Introduction

In the age of global connectivity and the presence of hacker's and electronic eavesdropping [121, 122] grows the need of security [69, 121, 122, 130] and there is an endless scope of research in this field. There are two main reasons for the essentiality of security of digital systems.

- First, explosive growth in the need of information through computers and networks.
- Second, the disciplines of cryptography [121, 122, 130] should be adaptive to enforce network security [38, 121, 122, 130].

Cryptography involves the study of mathematical techniques that allow the practitioner to achieve/provide the following objectives or services [121, 122]:

- **Confidentiality:** Service that keeps the data involved in an electronic transaction private. Meaning that the transmitted information is accessible only by authorized parties. This service includes both protections of all user data transmitted between two points over a period of time as well as protection of traffic flow from analysis.
- **Data Integrity:** Service that requires that computer system assets and transmitted information be capable of modification only by authorized users. Modification includes writing, changing, changing the status, deleting, creating, and the delaying or replaying of transmitted messages. It is important to point out that integrity relates to active attacks and therefore, it is concerned with detection rather than prevention. Moreover, integrity can be provided with or without recovery, the first option being the more attractive alternative.
- **Authentication:** Service that is concerned with assuring that the origin of a message is correctly identified. That is, information delivered over a channel should be authenticated as to the

origin, date of origin, data content, time sent, etc. For these reasons this service is subdivided into two major classes: entity authentication and data origin authentication. Notice that the second class of authentication implicitly provides data integrity.

- **Non-Repudiation:** This simply tells that the actions performed by the service user in an electronic transaction are non revocable so that they are legally binding. Therefore, neither the sender nor the receiver of a message should be able to deny the transaction.

There are two major classes of algorithms in cryptography: Private-key or Symmetric-key algorithms [10, 13, 22, 32, 39, 41, 42, 117, 121, 122, 130] and Public-key or Asymmetric-key algorithms [14, 60, 64, 67, 82, 121, 122, 130]. Symmetric-key cryptography can be divided into block ciphers [16, 17, 24, 25, 44, 45, 47, 89, 94, 121, 122, 130] and stream ciphers [7, 52, 54, 55, 56, 57, 58, 59, 121, 122, 130, 133, 134]. Figure 1.1 depicts the taxonomy of cryptography.

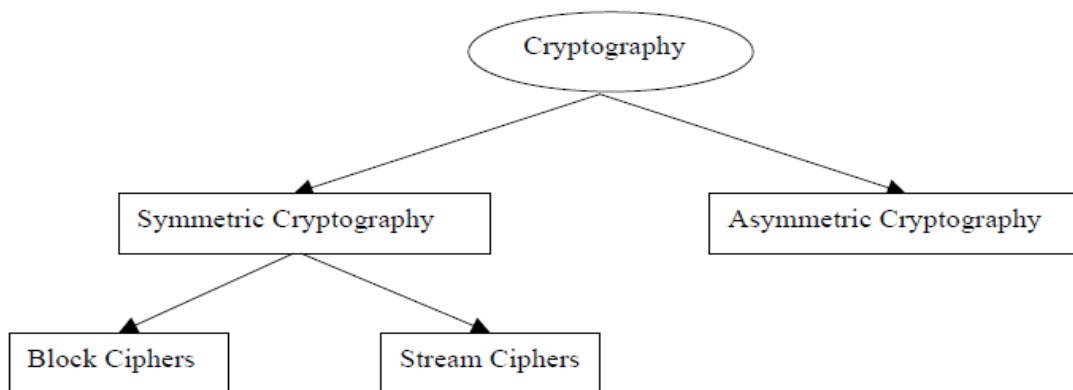


Figure 1.1: Taxonomy of cryptography

The current research work has been carried out using symmetric key cryptography implemented in 8085 microprocessor [17, 95, 101, 102, 104, 109, 111, 112, 113, 123, 124, 130] and FPGA-based system [1, 18, 19, 21, 29, 30, 63, 76, 77, 78, 79, 80, 81, 85, 86, 87, 88, 106, 107, 108, 125, 126] and has been simulated in Xilinx [127, 128] software for making of crypto-processor [1, 41, 61, 70, 96, 104] and or crypto-hardware [41, 88, 116, 117, 118].

Private-key or Symmetric-key algorithms are based on techniques where the encryption and decryption key [15, 27, 33, 35, 37, 46, 55, 71, 75, 121, 122, 130] is the identical, or the decryption key can easily be calculated from the encryption key and vice versa. The main function of these algorithms, which are also called secret-key algorithms, is encryption of data, often at high speeds. Private-key algorithms require the sender and the receiver to agree on the key prior to the communication. The security of private-key algorithms rests in the key; guessing the key means that anyone can perform encryption [8, 9, 15, 20, 27, 35, 36, 37, 62, 74, 77, 90, 121, 122, 130] and decryption [8, 9, 15, 20, 27, 35, 36, 37, 62, 74, 77, 90, 121, 122, 130] of messages [121, 122, 130]. Therefore, as long as the communication needs to remain secret, the key must remain secret. There are two types of symmetric-key algorithms, which are commonly distinguished: Block Ciphers and Stream Ciphers. The advantage of symmetric-key cryptography is it can encrypt bulk data very efficiently compared to asymmetric-key cryptography. Specialized algorithms for use in contexts where usual algorithms do not provide adequate performance, especially low-power embedded devices/systems [40, 63, 77, 119, 126] is another advantage of symmetric algorithms/techniques. The terminology of symmetric cryptography algorithms mainly includes the followings:

- Plaintext: Plaintext [121, 122, 130] is the ordinary information that the sender wishes to transmit to the receiver(s) at destination.
- Cipher Text: The encrypted text is called Ciphertext [121, 122, 130].
- Encryption and Decryption: Encryption is the process of converting plain text into cipher text. Decryption is the reverse process, converting from cipher text back to the original plain text.
- Cipher: A cipher is a pair of algorithms, which ensure the encryption and the reversing decryption. The detailed operation of a cipher is controlled both by the algorithm and by a specific key.
- Key: The key is a secret parameter for encrypting or decrypting a specific message-exchange context.

- **Block Ciphers:**

Keys are important, as ciphers without keys are trivially breakable and therefore less than useful for most purposes.

The block cipher is a type of symmetric-key encryption algorithm that transforms a fixed-length block of plain text data into a block of cipher text data of the same length. This transformation takes place under the action of a user-provided secret key. Decryption is performed by applying the reverse transformation to the cipher text block using the same secret key. The fixed length is called the block size and, for many block ciphers, the block size is 64 bits. In the coming years, the block size will increase to 128 bits as processors become more sophisticated. Since messages are almost always longer than a single block, some method of knitting together successive blocks is required. The different ways of knitting together blocks are known as the modes of operation and must be carefully considered when using block ciphers.

- **Rounds / Iteration**

Rounds [121, 122, 130] is the number of iterations in a cipher system. According to the crypto analysts, the bigger the number of rounds, the more secure the algorithms will be. The downside is that the execution time of the algorithms increases enormously.

- **Stream Ciphers:**

A stream cipher is a symmetric-key cipher where plaintext bits are combined with a pseudo-random cipher bit-stream (key stream), typically by an exclusive-or (XOR/EXOR) operation [128, 130]. In a stream cipher the plaintext digits are encrypted one at a time, and the transformation

- of successive digits varies during the encryption. Stream ciphers typically execute at a higher speed than block ciphers and have lower hardware complexity. Stream ciphers that only encrypt and decrypt data one bit at a time are not really suitable for software implementation. This explains why stream ciphers can be better implemented in hardware than block ciphers.

Figure 1.2 shows the block diagram of symmetric key cryptography.

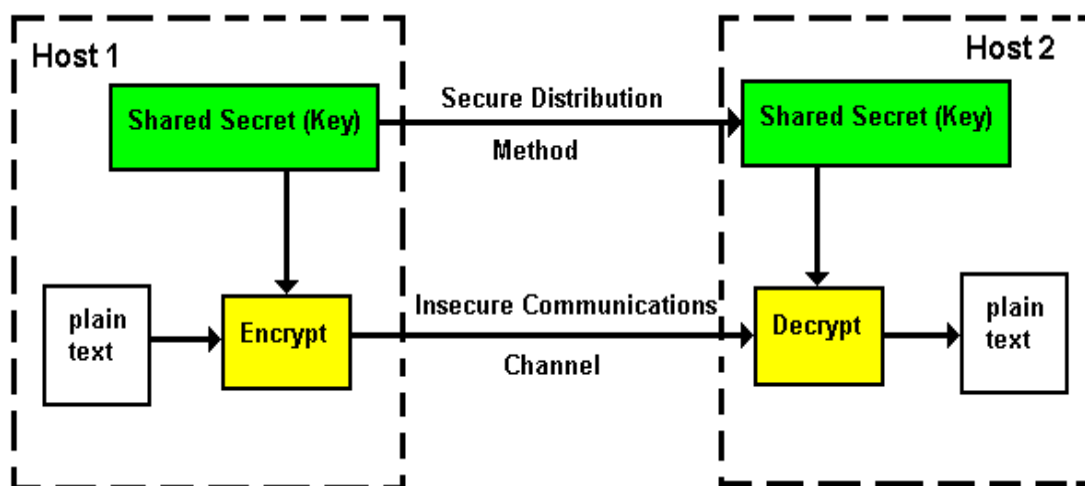


Figure 1.2: Taxonomy of symmetric key cryptography

Asymmetric cryptography, also called public key cryptography, invented by Diffie and Hellman [43, 121, 122] in 1976. The essential difference to symmetric cryptography is that this kind of algorithm uses two different keys for encryption and corresponding decryption.

Public-key encryption (also called asymmetric encryption) involves a pair of keys--a public and a private key--associated with an entity that needs to authenticate its identity electronically or to sign or encrypt data. Each public key is published, and the corresponding private key is kept secret.

- Private Key: This key must be known only by its owner.
- Public key: This key is known to everyone (it is public).

- Relation between both keys: What one key encrypts, the other one decrypts, and vice versa. That means that if you encrypt something with my public key (which you would know, because it's public -), I would need my private key to decrypt the message.

Each participant in a secure communication owns the pair of keys, public key 'p' and secret key 's'. The keys 'p' and 's' are mathematically dependent from each other, a requirement to the asymmetric algorithm being that, while 'p' can be computed easily from 's', obtaining 's' from 'p' is computationally unfeasible. This property allows making 'p' publicly known, while 's' must be kept secret by its owner. This asymmetry of the keys allows novel and very interesting uses of cryptography.

In a secure communication using public-key cryptography, the sender encrypts the message using the receiver's public key. Remember that this key is known to everyone. The encrypted message is sent to the receiving end, which will decrypt the message with his private key. Only the receiver can decrypt the message because no one else has the private key. Also, notice how the encryption algorithm is the same at both ends: what is encrypted with one key is decrypted with the other key using the same algorithm.

Public-key systems have a clear advantage over symmetric algorithms: there is no need to agree on a common key for both the sender and the receiver. If someone wants to receive an encrypted message, the sender only needs to know the receiver's public key (which the receiver will provide: publishing the public key in no way compromises the secure transmission). As long as the receiver keeps the private key secret, no one but the receiver will be able to decrypt the messages encrypted with the corresponding public key. This is due to the fact that, in public-key systems, it is relatively easy to compute the public key from the private key, but very hard to compute the private key from the public key (which is the one everyone knows). In fact, some algorithms need several months (and even years) of constant computation to obtain the private key from the public key.

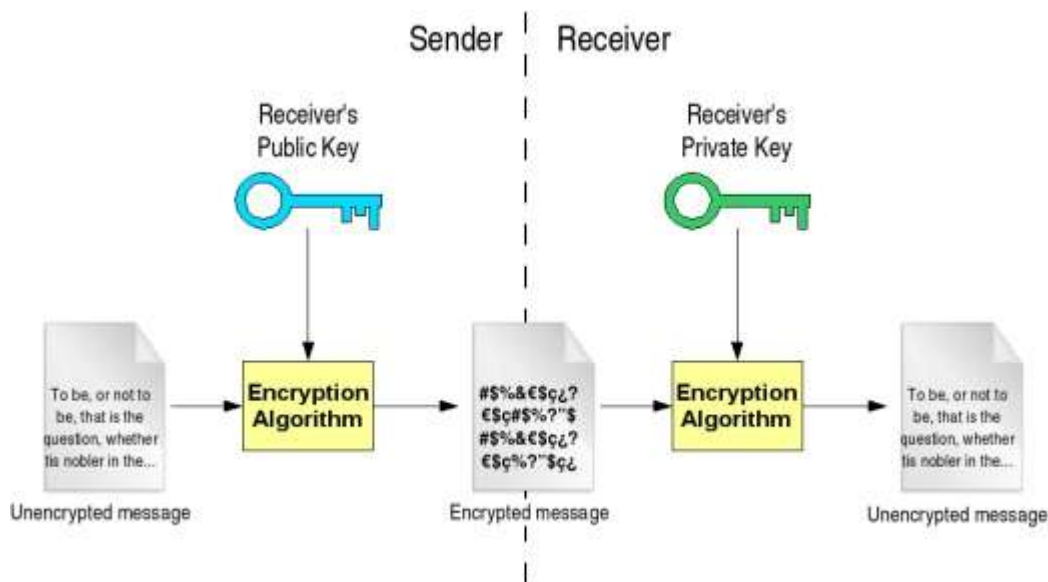


Figure 1.3: Taxonomy of public key cryptography

The 8085 is a conventional von Neumann design based on the Intel 8080. Unlike the 8080 it does not multiplex state signals onto the data bus, but the 8-bit data bus was instead multiplexed with the lower part of the 16-bit address bus to limit the number of pins to 40. Pin No. 40 is used for the power supply (+5v) and pin No. 20 for ground. Pin No. 39 is used as the hold pin. Pins No. 15 to No. 8 are generally used for address buses. The processor was designed using n MOS circuitry and the later "H" versions were implemented in Intel's enhanced nMOS process called HMOS, originally developed for fast static RAM products. Only a 5 Volt supply is needed, like competing processors and unlike the 8080. The 8085 uses approximately 6,500 transistors. The 8085 has extensions to support new interrupts, with three mask able interrupts (RST 7.5, RST 6.5 and RST 5.5), one non-mask able interrupt (TRAP), and one externally serviced interrupt (INTR). The RST n.5 interrupts refer to actual pins on the processor, a feature which permitted simple systems to avoid the cost of a separate interrupt controller.

Like 8080, the 8085 can accommodate slower memories through externally generated wait states (pin 35, READY), and has provisions for Direct Memory Access (DMA) using HOLD and HLDA signals (pins 39 and 38). An improvement over the 8080 was that the 8085 can itself drive a piezoelectric crystal directly connected to it, and a built in clock generator generates the internal high amplitude two-phase clock signals at half the crystal frequency (a 6.14 MHz crystal would yield a 3.07 MHz clock, for instance).

The processor has seven 8-bit registers named A, B, C, D, E, H, and L, where A is the 8-bit accumulator and the other six can be used as independent byte-registers or as three 16-

bit register pairs, BC, DE, and HL, depending on the particular instruction. Some instructions use HL as a (limited) 16-bit accumulator. As in the 8080, the contents of the memory address pointed to by HL could be accessed as pseudo register M. It also has a 16-bit stack pointer to memory (replacing the 8080's internal stack), and a 16-bit program counter. HL pair is called the primary data pointers.

As in many other 8-bit processors, all instructions are encoded in a single byte (including register numbers, but excluding immediate data), for simplicity. Some of them are followed by one or two bytes of data, which could be an immediate operand, a memory address, or a port number. Like larger processors, it has CALL and RET instructions for multi-level procedure calls and returns (which can be conditionally executed, like jumps) and instructions to save and restore any 16-bit register-pair on the machine stack. There are also eight one-byte call instructions (RST) for subroutines located at the fixed addresses 00h, 08h, 10h, ..., 38h. These were intended to be supplied by external hardware in order to invoke a corresponding interrupt-service routine, but are also often employed as fast system calls. The most sophisticated command was XTHL, which is used for exchanging the register pair HL with the value stored at the address indicated by the stack pointer.

Most 8-bit operations work on the 8-bit accumulator (the A register). For two operand 8-bit operations, the other operand can be an immediate value, another 8-bit register, or a memory cell addressed by the 16-bit register pair HL. Direct copying is supported between any two 8-bit registers and between any 8-bit register and a HL-addressed memory cell. Due to the regular encoding of the MOV-instruction (using a quarter of available op-code space) there are redundant codes to copy a register into itself (MOV B,B, for instance), which are of little use, except for delays. However, what would have been a copy from the HL-addressed cell into itself (i.e., MOV M,M) instead encodes the HLT instruction, halting execution until an external reset or interrupt occurred.

Although the 8085 is an 8-bit processor, it also has some 16-bit operations. Any of the three 16-bit register pairs (BC, DE, HL) or SP could be loaded with an immediate 16-bit value (using LXI), incremented or decremented (using INX and DCX), or added to HL (using DAD). LHLD loaded HL from directly-addressed memory and SHLD stored HL likewise. The XCHG operation exchanges the values of HL and DE. Adding HL to itself performs a 16-bit arithmetical left shift with one instruction. The only 16 bit instruction that affects any flag was DAD (adding HL to BC, DE, HL or SP), which updates the carry flag to facilitate 24-bit or larger additions and left shifts (for a floating point mantissa for instance). Adding the stack pointer to HL is useful for indexing variables in (recursive) stack frames. A stack

frame can be allocated using DAD SP and SPHL, and a branch to a computed pointer can be done with PCHL. These abilities make it feasible to compile languages such as PL/M, Pascal, or C with 16-bit variables and produce 8085 machine code. Subtraction and bitwise logical operations on 16 bits is done in 8-bit steps. Operations that have to be implemented by program code (subroutine libraries) included comparisons of signed integers as well as multiply and divide.

The 8085 supported up to 256 input/output (I/O) ports, accessed via dedicated Input / Output instructions—taking port addresses as operands. This Input / Output mapping scheme was regarded as an advantage, as it freed up the processor's limited address space.

For the extensive use of 8085 in various applications, the microprocessor is provided with an instruction set which consists of various instructions such as MOV, ADD, SUB, JMP etc. These instructions are written in the form of a program which is used to perform various operations such as branching, addition, subtraction, bitwise logical and bit shift operations. More complex operations and other arithmetic operations must be implemented in software. For example, multiplication is implemented using a multiplication algorithm.

The 8085 processor was used in a few early personal computers, for example, the TRS-80 Model 100 line used a OKI manufactured 80C85 (MSM80C85ARS). The CMOS version 80C85 of the NMOS/HMOS 8085 processor has several manufacturers. Some manufacturers provide variants with additional functions such as additional instructions. The red-hard version of the 8085 has been in on-board instrument data processors for several NASA and ESA space physics missions in the 1990s and early 2000s, including CRRES, Polar, FAST, Cluster, HESSI, the Sojourner Mars Rover, and THEMIS. The Swiss company SAIA used the 8085 and the 8085-2 as the CPUs of their PCA1 line of programmable logic controllers during the 1980s.

In many engineering schools the 8085 processor is used in introductory microprocessor courses. Trainer kits composed of a printed circuit board, 8085, and supporting hardware are offered by various companies. These kits usually include complete documentation allowing a student to go from solder to assembly language programming in a single course.



Figure 1.4: Intel 8085 microprocessor (courtesy Intel)

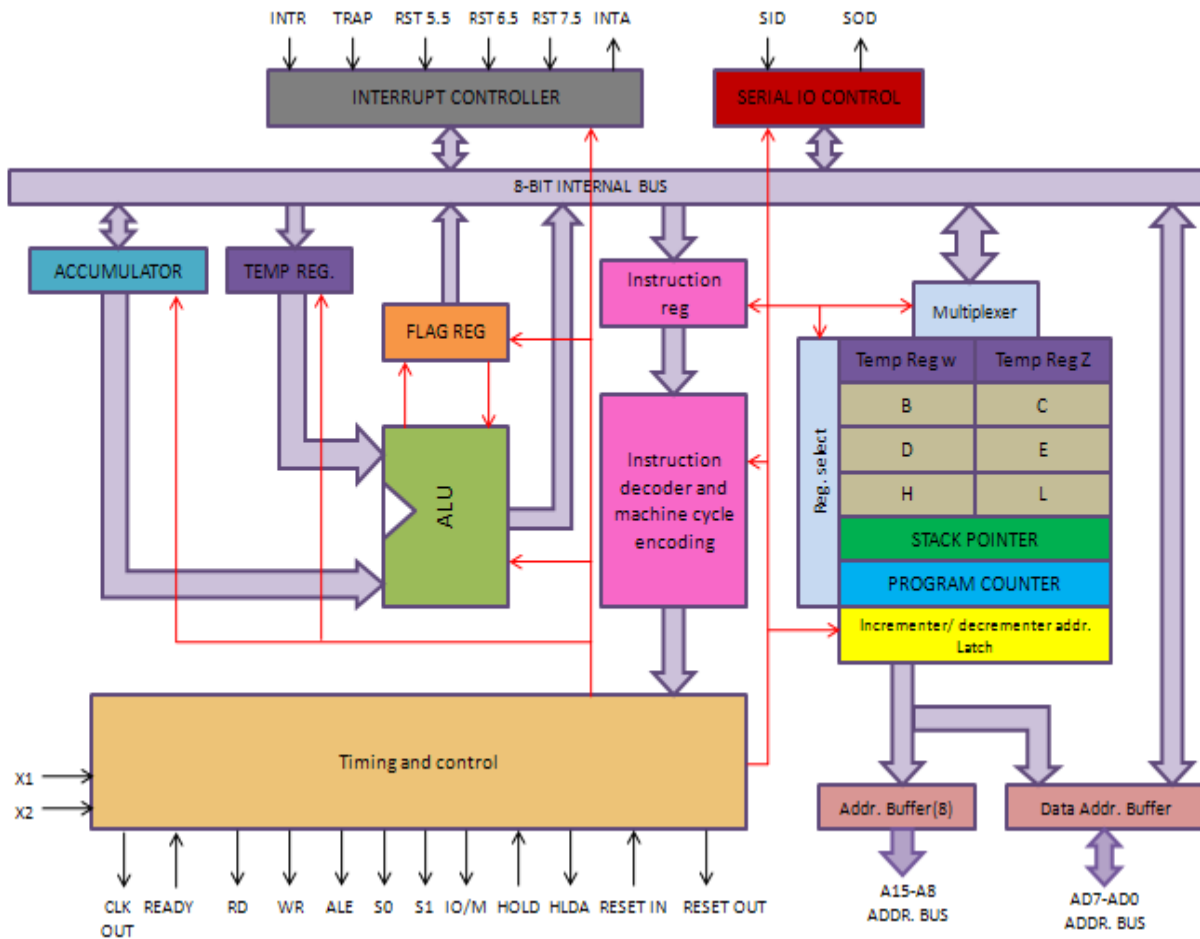


Figure 1.5: Intel 8085 microprocessor architecture (courtesy Intel)

Digital electronics is concerned with circuits which represent information using a finite set of output states. Most of the applications use in fact just two states, which are often labelled ‘0’ and ‘1’. Behind this choice is the fact that the whole Boolean formalism then becomes available for the solution of logic problems, and also that arithmetic using binary representations of numbers is a very mature field.

A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by the customer or designer after manufacturing—hence "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL) [127, 128, 130], similar to that used for an application-specific integrated circuit (ASIC) [63, 127, 128, 130] (circuit diagrams were previously used to specify the configuration, as they were for ASICs, but this is increasingly rare). FPGAs can be used to implement any logical function that an ASIC could perform. The ability to update the functionality after shipping, partial re-configuration of the portion of the design and the low non-recurring

engineering costs relative to an ASIC design (notwithstanding the generally higher unit cost), offer advantages for many applications.

FPGAs contain programmable logic components called "logic blocks", [125, 126] and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together"—somewhat like many (changeable) logic gates that can be inter-wired in (many) different configurations. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND [123, 124, 128, 130] and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

In addition to digital functions, some FPGAs have analog features. The most common analog feature is programmable slew rate and drive strength on each output pin, allowing the engineer to set slow rates on lightly loaded pins that would otherwise ring unacceptably, and to set stronger, faster rates on heavily loaded pins on high-speed channels that would otherwise run too slow. Another relatively common analog feature is differential comparators on input pins designed to be connected to differential signaling channels. A few "mixed-signal FPGAs" have integrated peripheral Analog-to-Digital Converters (ADCs) [123, 124, 130] and Digital-to-Analog Converters (DACs) [123, 124, 130] with analog signal conditioning blocks allowing them to operate as a system-on-a-chip. Such devices blur the line between an FPGA, which carries digital ones and zeros on its internal programmable interconnect fabric, and field-programmable analog array (FPAA) [125, 126], which carries analog values on its internal programmable interconnect fabric.

Applications of FPGAs include digital signal processing, software-defined radio, aerospace and defense systems, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation, radio astronomy, metal detection and a growing range of other areas.

FPGAs originally began as competitors to CPLDs [123, 124, 125, 126, 130] and competed in a similar space, that of glue logic for PCBs [123, 124, 125, 126, 130]. As their size, capabilities, and speed increased, they began to take over larger and larger functions to the state where some are now marketed as full systems on chips (SoC) [123, 124, 125, 126, 130]. Particularly with the introduction of dedicated multipliers into FPGA architectures in the late 1990s, applications which had traditionally been the sole reserve of DSPs [123, 124, 130] began to incorporate FPGAs instead.

Traditionally, FPGAs have been reserved for specific vertical applications where the volume of production is small. For these low-volume applications, the premium that

companies pay in hardware costs per unit for a programmable chip is more affordable than the development resources spent in creating an ASIC for a low-volume application. Today, new cost and performance dynamics have broadened the range of viable applications.

A typical modern FPGA provides the designer with programmable logic blocks that contain the pool of combinatorial blocks and flip-flops to be used in the design. In addition, vendors acknowledge the fact that logic is often used in conjunction with memory, and typically include variable amounts of static Random Access Memory (RAM) [123, 124, 130] inside their chips. Clock conditioning has also become commonplace, and support in the form of Delay Locked Loops (DLLs) [123, 124, 130] and Phase Locked Loops (PLLs) [123, 124, 130] is also provided inside the same silicon chip. Finally, an FPGA chip does not lead a solitary life isolated from the rest of the world. It needs to be easily interfaced to other chips or external signals. In order to make this interfacing easier, FPGA vendors have invested a great deal of effort in enhancing the flexibility of the input/output blocks behind the chip pads. Each pad can serve as an input, an output, or both. The list of electrical standards supported is extensive, and novel techniques for maximizing bandwidth, such as clocking data in using both edges of the clock, are widely supported. The designer facing a design problem must go through a series of steps between initial ideas and final hardware. This series of steps is commonly referred to as the 'design flow' [125, 126, 127, 128, 130]. First, all the requirements have been spelled out, a proper digital design phase must be carried out. It should be stressed that the tools supplied by the different FPGA vendors to target their chips do not help the designer in this phase. They only enter the scene once the designer is ready to translate a given design into working hardware.

The most common flow nowadays used in the design of FPGAs involves the following subsequent phases [125, 126, 127, 128, 130]:

- Design entry This step consists in transforming the design ideas into some form of computerized representation. This is most commonly accomplished using Hardware Description Languages (HDLs). The two most popular HDLs are Verilog [125, 126, 127, 128, 130] and the Very High Speed Integrated Circuit HDL (VHDL) [125, 126, 127, 128, 130]. It should be noted that an HDL, as its name implies, is only a tool to describe a

design that pre-existed in the mind, notes, and sketches of a designer. It is not a tool to design electronic circuits. Another point to note is that HDLs differ from conventional software programming languages in the sense that they don't support the concept of sequential execution of statements in the code. This is easy to understand if one considers the alternative schematic representation of an HDL file: what one sees in the upper part of the schematic cannot be said to happen before or after what one sees in the lower part.

- **Synthesis**

The synthesis tool receives HDL and a choice of FPGA vendor and model. From these two pieces of information, it generates a netlist which uses the primitives proposed by the vendor in order to satisfy the logic behaviour specified in the HDL files. Most synthesis tools go through additional steps such as logic optimization, register load balancing, and other techniques to enhance timing performance, so the resulting netlist can be regarded as a very efficient implementation of the HDL design.

- **Place and route**

The placer takes the synthesized netlist and chooses a place for each of the primitives inside the chip. The router's task is then to interconnect all these primitives together satisfying the timing constraints. The most obvious constraint for a design is the frequency of the system clock, but there are more involved constraints one can impose on a design using the software packages supported by the vendors.

- **Bit stream generation**

FPGAs are typically configured at power-up time from some sort of external permanent storage device, typically a flash memory. Once the place and route process is finished, the resulting choices for the configuration of each programmable element in the

FPGA chip, be it logic or interconnect, must be stored in a file to program the flash.

Of these four phases, only the first one is human-labour intensive. Somebody has to type in the HDL code, which can be tedious and error-prone for complicated designs involving, for example, lots of digital signal processing. This is the reason for the appearance, in recent years, of alternative flows which include a preliminary phase in which the user can draw blocks at a higher level of abstraction and rely on the software tool for the generation of the HDL. Some of these tools also include the capability of simulating blocks which will become HDLs with other blocks which provide stimuli and processing to make the simulation output easier to interpret. The concept of hardware co-simulation is also becoming widely used. In co-simulation, stimuli are sent to a running FPGA hosting the design to be tested and the outputs of the design are sent back to a computer for display (typically through a Joint Test Action Group (JTAG), or Ethernet connection). The advantage of co-simulation is that one is testing the real system, therefore suppressing all possible misinterpretations present in a pure simulator. In other cases, co-simulation may be the only way to simulate a complex design in a reasonable amount of time.

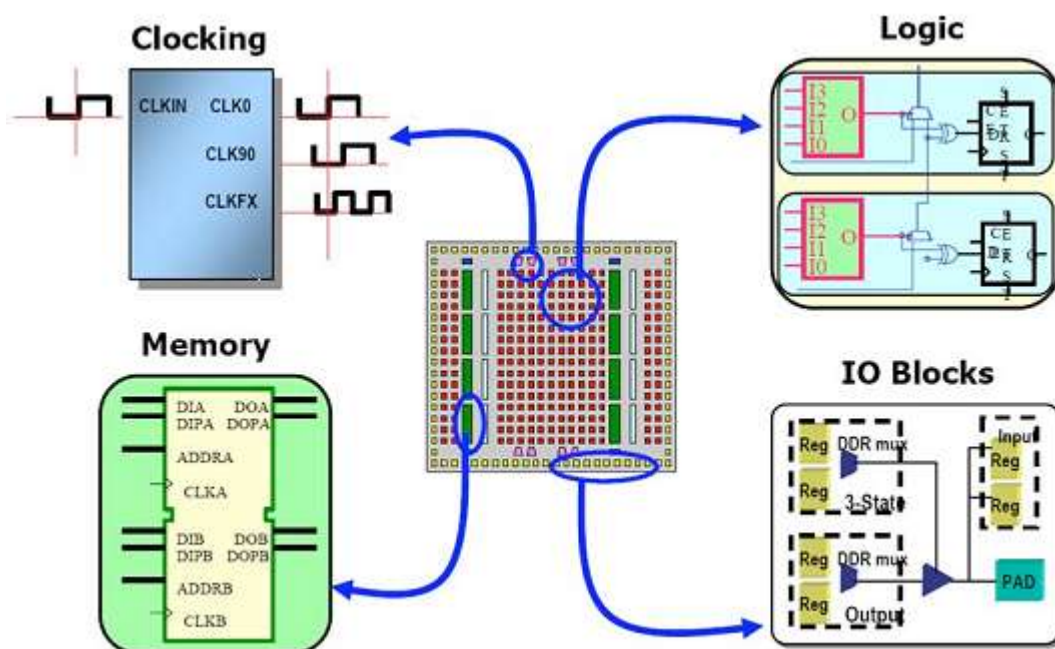


Figure 1.6: Internal structure of a generic FPGA (courtesy Xilinx)

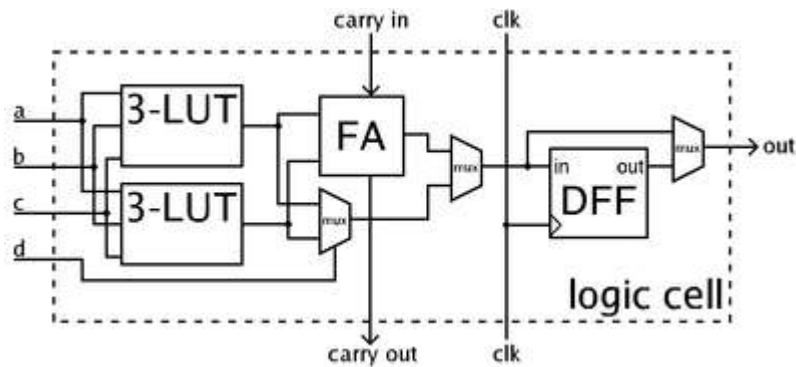


Figure 1.7: Simplified illustration of a logic cell (courtesy Xilinx)

VHDL stands for very high-speed integrated circuit hardware description language. This is one of the programming languages used to model a digital system by dataflow, behavioral and structural style of modeling. This language was first introduced in 1981 for the department of Defense (DoD) under the VHSIC [125, 126, 127, 128, 130] program. In 1983 IBM, Texas instruments and Intermetrics started to develop this language. In 1985 VHDL 7.2 version was released. In 1987 IEEE standardized the language. VHDL is commonly used to write text models that describe a logic circuit. Such a model is processed by a synthesis program, only if it is part of the logic design. A simulation program is used to test the logic design using simulation models to represent the logic circuits that interface to the design. This collection of simulation models is commonly called a test bench. VHDL has constructs to handle the parallelism inherent in hardware designs, but these constructs (processes) differ in syntax from the parallel constructs in Ada (tasks). Like Ada, VHDL is strongly typed and is not case sensitive. In order to directly represent operations which are common in hardware, there are many features of VHDL which are not found in Ada, such as an extended set of Boolean operators including nand and nor. VHDL also allows arrays to be indexed in either ascending or descending direction; both conventions are used in hardware, whereas in Ada and most programming languages only ascending indexing is available. VHDL has file input and output capabilities, and can be used as a general-purpose language for text processing, but files are more commonly used by a simulation test bench for stimulus or verification data. There are some VHDL compilers which build executable binaries. In this case, it might be possible to use VHDL to write a test bench to verify the functionality of the design using files on the host computer to define stimuli, to interact with the user, and to compare results with those expected. However, most designers leave this job to the simulator. It is relatively easy for an inexperienced developer to produce code that simulates successfully but that cannot be synthesized into a real device, or is too large to be practical.

One particular pitfall is the accidental production of transparent latches rather than D-type flip-flops as storage elements.

One can design hardware in a VHDL IDE [125, 126, 127, 128, 130] (for FPGA implementation such as Xilinx ISE, Altera Quartus, Synopsys Synplify or Mentor Graphics HDL Designer) to produce the RTL schematic of the desired circuit. After that, the generated schematic can be verified using simulation software which shows the waveforms of inputs and outputs of the circuit after generating the appropriate test-bench [125, 126, 127, 128, 130]. To generate an appropriate test-bench for a particular circuit or VHDL code, the inputs have to be defined correctly. For example, for clock input, a loop process or an iterative statement is required. A final point is that when a VHDL model is translated into the "gates and wires" that are mapped onto a programmable logic device such as a CPLD or FPGA, then it is the actual hardware being configured, rather than the VHDL code being "executed" as if on some form of a processor chip.

The key advantage of VHDL, when used for systems design, is that it allows the behavior of the required system to be described (modeled) and verified (simulated) before synthesis tools translate the design into real hardware (gates and wires). Another benefit is that VHDL allows the description of a concurrent system. VHDL is a dataflow language, unlike procedural computing languages such as BASIC, C, and assembly code, which all run sequentially, one instruction at a time. VHDL project is multipurpose. Being created once, a calculation block can be used in many other projects. However, many formational and functional block parameters can be tuned (capacity parameters, memory size, element base, block composition and interconnection structure). VHDL project is portable. Being created for one element base, a computing device project can be ported on another element base, for example VLSI with various technologies. In VHDL, a design consists at a minimum of an entity which describes the interface and an architecture which contains the actual implementation. In addition, most designs import library modules. Some designs also contain multiple architectures and configurations.

A simple AND gate in VHDL would look something like this:

```
-- (this is a VHDL comment)
```

```
-- import std_logic from the IEEE library
```

```
library IEEE;
```

```
use IEEE.std_logic_1164.all;
```



```

-- this is the entity
entity ANDGATE is
    port (
        I1 in std_logic,
        I2 in std_logic,
        O out std_logic);
end entity ANDGATE;

architecture RTL of ANDGATE is
begin
    O <= I1 and I2;
end architecture RTL;

```

While the example above may seem very verbose to HDL beginners, many parts are either optional or need to be written only once. Generally simple functions like this are part of a larger behavioural module, instead of having a separate module for something so simple. In addition, use of elements such as the `std_logic` type might at first seem to be overkill. One could easily use the built-in bit type and avoid the library import in the beginning. However, using this 9-valued logic (U,X,0,1,Z,W,H,L,-) instead of simple bits (0,1) offers a very powerful simulation and debugging tool to the designer which currently does not exist in any other HDL.

In the examples that follow, it is seen that VHDL code can be written in a very compact form. However, the experienced designers usually avoid these compact forms and use a more verbose coding style for the sake of readability and maintainability. Another advantage to the verbose coding style is the smaller amount of resources used when programming to a Programmable Logic Device such as a CPLD. Synthesizable constructs and VHDL templates.

VHDL is frequently used for two different goals: simulation of electronic designs and synthesis of such designs. Synthesis is a process where a VHDL is compiled and mapped into an implementation technology such as an FPGA or an ASIC. Many FPGA vendors have free (or inexpensive) tools to synthesize VHDL for use with their chips, where ASIC tools are often very expensive. Not all constructs in VHDL are suitable for synthesis. For example, most constructs that explicitly deal with timing such as `wait for 10 ns;` are not synthesizable.

despite being valid for simulation. While different synthesis tools have different capabilities, there exists a common synthesizable subset of VHDL that defines what language constructs and idioms map into common hardware for many synthesis tools. IEEE 1076.6 defines a subset of the language that is considered the official synthesis subset. It is generally considered a "best practice" to write very idiomatic code for synthesis as results can be incorrect or suboptimal for non-standard constructs.

In VHDL an entity is used to describe a hardware module.

An entity can be described using,

1. Entity declaration.
2. Architecture.
3. Configuration
4. Package declaration.
5. Package body.

Let's see what are these?

Entity declaration:

It defines the names, input output signals and modes of a hardware module.

Syntax:

```
entity entity_name is
  Port declaration;
end entity_name;
```

An entity declaration should starts with 'entity' and ends with 'end' keywords.

Ports are interfaces through which an entity can communicate with its environment. Each port must have a name, direction and a type. An entity may have no port declaration also. The direction will be input, output or inout.

In	Port can be read
Out	Port can be written
Inout	Port can be read and written
Buffer	Port can be read and written, it can have only one source

Architecture:

It describes the internal description of design or it tells what is there inside design. Each entity has at least one architecture and an entity can have many architecture. Architecture can be described using structural, data flow, behavioral or mixed style. Architecture can be used to describe a design at different levels of abstraction like gate level, register transfer level (RTL) or behavior level.

Syntax:

```
architecture architecture_name of entity_name
    architecture_declarative_part
begin
    Statements;
end architecture_name;
```

Here it should specify the entity name for which writing the architecture body was done. The architecture statements should be inside the begin and end keyword. Architecture declarative part may contain variables, constants, or component declaration.

Configuration:

If an entity contains many architectures and any one of the possible architecture binding with its entity is done using configuration. It is used to bind the architecture body to its entity and a component with an entity.

Syntax:

```
configuration configuration_name of entity_name is
    block_configuration;
end configuration_name;
```

Block_configuration defines the binding of components in a block. This can be written as

```
for block_name
    component_binding;
end for;
```

block_name is the name of the architecture body. Component binding binds the components of the block to entities. This can be written as,

```
for component_labels:component_name
    block_configuration;
end for;
```

Package declaration:

Package declaration is used to declare components, types, constants, functions and so on.

Syntax:

```
package package_name is
    Declarations;
end package_name;
```

Package body:

A package body is used to declare the definitions and procedures that are declared in corresponding package. Values can be assigned to constants declared in package in package body.

Syntax:

```
package body package_name is
    Function_procedure
    definitions;
end package_name;
```

The internal working of an entity can be defined using different modeling styles inside architecture body. They are

1. Dataflow modeling.
2. Behavioral modeling.
3. Structural modeling.

Structure of an entity:

Let's try to understand with the help of one example.

Data flow modeling:

In this style of modeling, the internal working of an entity can be implemented using concurrent signal assignment.

Let's take half adder example which is having one XOR gate and a AND gate.

```
Library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity ha_en is
    port (A,B:in bit;S,C:out bit);
end ha_en;

architecture ha_ar of ha_en is
    begin
        S<=A xor B;
        C<=A and B;
    end ha_ar;
```

Here STD_LOGIC_1164 is an IEEE standard which defines a nine-value logic type, called STD_ULOGIC. use is a keyword, which imports all the declarations from this package. The architecture body consists of concurrent signal assignments, which describes

the functionality of the design. Whenever there is a change in RHS, the expression is evaluated and the value is assigned to LHS.

Behavioral modeling:

In this style of modeling, the internal working of an entity can be implemented using set of statements.

It contains:

- Process statements
- Sequential statements
- Signal assignment statements
- Wait statements

Process statement is the primary mechanism used to model the behavior of an entity. It contains sequential statements, variable assignment ($:=$) statements or signal assignment ($<=$) statements etc. It may or may not contain sensitivity list. If there is an event occurs on any of the signals in the sensitivity list, the statements within the process are executed. Inside the process the execution of statements will be sequential and if one entity is having two processes the execution of these processes will be concurrent. At the end it waits for another event to occur.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity ha_beha_en is
    port(
        A : in BIT;
        B : in BIT;
        S : out BIT;
        C : out BIT
    );
end ha_beha_en;

architecture ha_beha_ar of ha_beha_en is
```

```

begin
    process_beh:process(A,B)
    begin
        S<= A xor B;
        C<=A and B;
    end process process_beh;
end ha_beha_ar;

```

Here whenever there is a change in the value of a or b the process statements are executed.

Structural modeling:

The implementation of an entity is done through set of interconnected components.

It contains:

- Signal declaration.
- Component instances
- Port maps.
- Wait statements.
- Component declaration:

Syntax:

```

component component_name [is]
    List_of_interface_ports;
end component component_name;

```

Declaration is done before instantiation of the component. Component declaration declares the name of the entity and interface of a component. Let's try to understand this by taking the example of full adder using two half adder and one OR gate.

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity fa_en is
    port(A,B,Cin:in bit; SUM, CARRY:out bit);
end fa_en;

architecture fa_ar of fa_en is
    component ha_en
        port(A,B:in bit;S,C:out bit);
    end component;
    signal C1,C2,S1:bit;
begin

    HA1:ha_en port map(A,B,S1,C1);
    HA2:ha_en port map(S1,Cin,SUM,C2);
    CARRY <= C1 or C2;
end fa_ar;

```

The program that have written for half adder in dataflow modeling is instantiated as shown above. ha_en is the name of the entity in dataflow modeling. C1, C2, S1 are the signals used for internal connections of the component which are declared using the keyword signal. Port map is used to connect different components as well as connect components to ports of the entity.

Component instantiation is done as follows.

```
Component_label: component_name port map (signal_list);
```

Signal_list is the architecture signals which are connecting to component ports. This can be done in different ways. What is declared here is positional binding. There is another type of binding termed as ‘named’ binding. The situation described can be written in terms of named binding as,

```

HA1:ha_en port map(A => A,B => B, S => S1 ,C => C1 );
HA2:ha_en port map(A => S1,B => Cin, S=> SUM, C => C2);

```


Test bench:

The correctness of the above program can be checked by writing the test bench.

The test bench is used for generating stimulus for the entity under test. Let's write a simple test bench for full adder.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity tb_en is

end tb_en;

architecture tb_ar of tb_en is
    signal a_i,b_i,c_i,sum_i,carry_i:bit;
begin

    eut: entity work.fa_en(fa_ar)
        port map(A=>a_i,B=>b_i,Cin=>c_i,SUM=>sum_i,CARRY=>carry_i);
    stimulus: process
        begin
            a_i<='1';b_i<='1';c_i<='1';
            wait for 10ns;
            a_i<='0';b_i<='1';c_i<='1';
            wait for 10ns;
            a_i<='1';b_i<='0';c_i<='0';
            wait for 10ns;
            if now=30ns then
                wait;
            end if;
        end process stimulus;
end tb_ar;
```

In the above example 'now' is a predefined function that returns the current simulation time what it is to saw up to this is component instantiation by positional and by name. In this test bench example the entity is directly instantiated. The direct entity instantiation syntax is:

```
Component_label: entity entity_name
```

```
(architecture_name)
```

```
port map(signal_list);
```

To evaluate the techniques/schemes some statistical test are performed such as Chi-Square test

A Chi-Square test [12, 15, 17, 23, 27, 33, 34, 35, 36, 37, 39, 93, 94, 121, 122, 129, 130, 142], also referred to as χ^2 test, is any statistical hypothesis test in which the sampling distribution of the test statistic is a Chi-Square distribution when the null hypothesis is true, or any in which this is asymptotically true, meaning that the sampling distribution (if the null hypothesis is true) can be made to approximate a Chi-Square distribution as closely as desired by making the sample size large enough.

Some examples of Chi-Square tests where the Chi-Square distribution is only approximately valid are:

- Pearson's Chi-Square test, also known as the Chi-Square goodness-of-fit test or Chi-Square test for independence. When mentioned without any modifiers or without other precluding context, this test is usually understood (for an exact test used in place of χ^2 , see Fisher's exact test).
- Yates's correction for continuity, also known as Yates' Chi-Square test.
- Cochran-Mantel-Haenszel Chi-Square test.
- Linear-by-linear association Chi-Square test.
- The portmanteau test in time-series analysis, testing for the presence of autocorrelation.
- Likelihood-ratio tests in general statistical modeling, for testing whether there is evidence of the need to move from a simple model to a more complicated one (where the simple model is nested within the complicated one).

One case where the distribution of the test statistic is an exact Chi-Square distribution is the test that the variance of a normally-distributed population has a given value based on a sample variance. Such a test is uncommon in practice because values of variances to test against are seldom known exactly.

If a sample of size n is taken from a population having a normal distribution, then there is a well-known result (see distribution of the sample variance) which allows a test to be made of whether the variance of the population has a pre-determined value. For example, a manufacturing process might have been in stable condition for a long period, allowing a value for the variance to be determined essentially without error. Suppose that a variant of the process is being tested, giving rise to a small sample of product items whose variation is to be tested. The test statistic T in this instance could be set to be the sum of squares about the sample mean, divided by the nominal value for the variance (i.e. the value to be tested as holding). Then T has a Chi-Square distribution with $n - 1$ degrees of freedom. For example if the sample size is 21, the acceptance region for T for a significance level of 5% is the interval 9.59 to 34.17.

The Chi-Square (χ^2) test is used to determine whether there is a significant difference between the expected frequencies and the observed frequencies in one or more categories.

The followings are the requirements of Chi-Square test

- Quantitative data.
- One or more categories.
- Independent observations.
- Adequate sample size (at least 10).
- Random sample.
- Data in frequency form.
- All observations must be used.

To find the value for Chi-Square, it is determined whether the observed frequencies differ significantly from the expected frequencies.

The formula of Chi-Square value is:

$$\chi^2 = \sum \frac{(O - E)^2}{E}$$

Where O is the Observed Frequency in each category

E is the Expected Frequency in the corresponding category

df is the "degree of freedom" ($n-1$)

χ^2 is Chi-Square

The steps in using the Chi-Square test may be summarized as follows

- Write the observed frequencies in column O
- Figure the expected frequencies and write them in column E .
- Use the formula to find the Chi-Square value.
- Find the df . ($N-1$)
- Find the table value (consult the Chi-Square Table.)
- If the Chi-Square value is equal to or greater than the table value, reject the null hypothesis. differences in your data are not due to chance alone.

In Statistics, the number of degrees of freedom [12, 15, 17, 23, 27, 33, 34, 35, 36, 37, 39, 93, 94, 121, 122, 129, 130, 142] is the number of values in the final calculation of a statistic that are free to vary. Estimates of statistical parameters can be based upon different amounts of information or data. The number of independent pieces of information that go into the estimate of a parameter is called the degrees of freedom (df). In general, the degrees of freedom of an estimate of a parameter is equal to the number of independent scores that go into the estimate minus the number of parameters used as intermediate steps in the estimation of the parameter itself (which, in sample variance, is one, since the sample mean is the only intermediate step).

Mathematically, degree of freedom is the dimension of the domain of a random vector, or essentially the number of 'free' components: how many components need to be known before the vector is fully determined. The term is most often used in the context of linear models (linear regression, analysis of variance), where certain random vectors are constrained to lie in linear subspaces, and the number of degrees of freedom is the dimension of the subspace. The degrees-of-freedom are also commonly associated with the squared lengths (or "Sum of Squares") of such vectors, and the parameters of Chi-Square and other distributions that arise in associated statistical testing problems. In this study degree of freedom is the number of different ASCII characters present in a file.

A frequency distribution [12, 15, 17, 23, 27, 33, 34, 35, 36, 37, 39, 93, 94, 121, 122, 129, 130, 142] is a representation, either in a graphical or tabular format, which displays the

number of observations within a given interval. The intervals must be mutually exclusive and exhaustive. Frequency distributions are usually used within a statistical context. The size of the intervals used in a frequency distribution will depend on the data being analyzed and the goals of the analyst. However, the most important factor is that the intervals used must be non-overlapping and contain all of the possible observations.

A frequency distribution is one of the most common graphical tools used to describe a single population. It is a tabulation of the frequencies of each value (or range of values). There are a wide variety of ways to illustrate frequency distributions, including histograms, relative frequency histograms, density histograms, and cumulative frequency distributions. Histograms show the frequency of elements that occur within a certain range of values, while cumulative distributions show the frequency of elements that occur below a certain value.

Frequency Histogram is defined as follows -

- A graphical representation of a single dataset, tallied into classes.
- Frequency defined as the number of values that fall into each class.
- Histogram consists of a series of rectangles whose widths are defined by the limits of the classes, and whose heights are determined by the frequency in each interval.
- Histogram depicts many attributes of the data, including location, spread, and symmetry.
 - No rigid set of rules that determine the number of classes or class interval.
 - Between 5 and 20 classes suitable for most datasets.
 - Equal sized class widths are found by dividing the range by the number of classes.
 - Formal guide by which class intervals can be derived is the formula $K = 1 + 3.3 \cdot \log n$ where K is the number of classes and n is the number of variables.

A frequency distribution shows the number of observations falling into each of several ranges of values. Frequency distributions are portrayed as frequency tables, histograms, or polygons. Frequency distributions can show either the actual number of

observations falling in each range or the percentage of observations. In the latter instance, the distribution is called a relative frequency distribution.

In cryptography, the avalanche effect [121, 122, 129, 130, 136, 137, 138, 139, 140, 141, 142] is a desirable property of cryptographic algorithms, typically block ciphers and cryptographic hash functions. The avalanche effect is evident if, when an input is changed slightly (for example, flipping a single bit) the output changes significantly (e.g., half the output bits flip). In the case of quality block ciphers, such a small change in either the key or the plaintext should cause a drastic change in the ciphertext. The actual term was first used by Horst Feistel although the concept dates back to at least Shannon's diffusion. If a block cipher or cryptographic hash function does not exhibit the avalanche effect to a significant degree, then it has poor randomization, and thus a cryptanalyst can make predictions about the input, being given only the output. This may be sufficient to partially or completely break the algorithm. Thus, the avalanche effect is a desirable condition from the point of view of the designer of the cryptographic algorithm or device. Constructing a cipher or hash to exhibit a substantial avalanche effect is one of the primary design objectives. This is why most block ciphers are product ciphers. It is also why hash functions have large data blocks. Both of these features allow small changes to propagate rapidly through iterations of the algorithm, such that every bit of the output should depend on every bit of the input before the algorithm terminates. Even a slight change in an input string should cause the hash value to change drastically. A Hash Function is a transformation that takes a variable length bit sequence (Message) and produces a fixed length bit sequence (Message Digest). Even if 1 bit is flipped in the input string, at least half of the bits in the hash value will flip as a result. This is called an avalanche effect. A function has a good avalanche effect when a change in one bit of the input results in a change of half of the output bits.

The strict avalanche criterion (SAC) [121, 122, 129, 130, 136, 137, 138, 139, 140, 141, 142] is a generalization of the avalanche effect. It is satisfied if, whenever a single input bit is complemented, each of the output bits changes with a 50% probability. The SAC builds on the concepts of completeness and avalanche and was introduced by Webster and Tavares in 1985. The bit independence criterion (BIC) states that output bits j and k should change independently when any single input bit l is inverted, for all l, j and k .

In computer science, the analysis of algorithms is the determination of the amount of resources (such as time and storage) necessary to execute them. Most algorithms are designed to work with inputs of arbitrary length. Usually the efficiency or running time of an algorithm is stated as a function relating the input length to the number of steps (time complexity) or

storage locations (space complexity) [12, 15, 17, 23, 27, 33, 34, 35, 36, 37, 39, 93, 94, 121, 122, 129, 130, 142].

Algorithm analysis is an important part of a broader computational complexity theory, which provides theoretical estimates for the resources needed by any algorithm which solves a given computational problem. These estimates provide an insight into reasonable directions of search for efficient algorithms. In theoretical analysis of algorithms it is common to estimate their complexity in the asymptotic sense, i.e., to estimate the complexity function for arbitrarily large input. Big O notation, notation and theta notation are used to this end. For instance, binary search is said to run in a number of steps proportional to the logarithm of the length of the list being searched, or in $O(\log(n))$, colloquially "in logarithmic time". Usually asymptotic estimates are used because different implementations of the same algorithm may differ in efficiency. However the efficiencies of any two "reasonable" implementations of a given algorithm are related by a constant multiplicative factor called a hidden constant.

Exact (not asymptotic) measures of efficiency can sometimes be computed but they usually require certain assumptions concerning the particular implementation of the algorithm, called model of computation. A model of computation may be defined in terms of an abstract computer, e.g., Turing machine, and/or by postulating that certain operations are executed in unit time. For example, if the sorted list to which to apply binary search has n elements, and it can guarantee that each lookup of an element in the list can be done in unit time, then at most $\log_2 n + 1$ time units are needed to return an answer.

Section 1.2 gives the details literature survey, section 1.3 describes the problem domain; section 1.4 illustrates the proposed methodology, section 1.5 states the salient features of this thesis and section 1.6 figures out the organization of this thesis.

1.2 Literature Survey

Diffie and Hellman [43] in the year 1976 gave a new direction in cryptography. Two kinds of contemporary in cryptography are examined. Widening applications of teleprocessing have given rise to the need of new type of cryptographic systems, which minimize the need for secure key distribution channels and supply the equivalent of a written signature. This paper suggests ways to solve these currently open problems. It also discusses how the theories of communication and computation are beginning to provide the tools to solve cryptographic problems of long standing.

Campbell [101] in 1979 proposed a microprocessor based module to provide security in electronic fund transfer. Electronic Fund Transfer (EFT) is expected to grow in importance and to result in national interchange system. The potential for fraud in EFT is quite significant, and can be prevented by the use of cryptographic security techniques. A microprocessor based security module has been developed which serves as a CPU peripheral to perform all cryptographic functions which an EDP facility requires to secure its EFT operations. The cryptographic operations include management of Personal Identification Numbers (PIN), the management of cryptographic "keys", and protection and validation of customer entered PINs.

The uses of microprocessors for implementing some of the new public cryptographic algorithms have been examined by Davida and Wells [102] in the year 1979. Time and space requirements for the various encryption programs are studied and applications where microprocessor based encryption would be able to meet throughput requirements are identified. It is concluded that with currently available microprocessors, micro-based encryption is useful for offline encryption/decryption, electronic mail systems, and most terminal applications.

Best [95] in the year 1980 proposed prevention of software piracy with crypto-microprocessor and it's a wonderful application of cryptography implemented in microprocessor based systems. A crypto-microprocessor executes a program which is stored in cipher to prevent it from being altered, disassembled, or copied for use in unauthorised processors. Each instruction, just before it is executed, is deciphered by the crypto-microprocessor under control of one or more secret encryption keys which are different for each program. Microprocessors lacking these keys can't execute an enciphered program or process enciphered data. Valuable proprietary programs and data files can thus be distributed in cipher along with dedicated crypto-microprocessor for use by numerous and anonymous people, without risk of piracy or unauthorised alteration of programs.

Computer communication systems, local-area networks, interconnected local-area networks, and electronic mail systems are playing an increasingly important role in office automation, telecommunications, and factory automation. A microprocessor based crypto-processor has been proposed by Schloer [104] in 1983. A prerequisite for extensive usage of these services, with full or partial replacement of conventional paper mail by an electronic medium, is security. It must be possible to guarantee the secrecy of a message so that only the addressee is able to read it (i.e. it must be possible to provide the equivalent of a paper envelope). Furthermore, the receiver of a message wants to verify that the indicated and the

real sender are one and the same (i.e., there must be a provision for electronic signatures and signature verification). Recent advances have made the technology of cryptography a viable tool for solving these problems. The DES-Data Encryption Standard as well as public-key systems have also been discussed extensively. In this article an experimental secure communication system and its implementation with a special module-the crypto-processor (CP) is described by the author. The overall system structure and user interface along with an overview of cryptography and a review of the design considerations are also presented by the author. Moreover, the software interfaces to the main component, the crypto-processor, and its data structure, hardware, software, and performance are also described.

A security and performance optimization of a new DES data encryption chip is proposed by Verbauwhede, Hoornaert and Vandewalle [97]. Cryptography applications demand high speed and security both. This paper presents the implementation of a new high-performance Data Encryption Standard (DES) data encryption chip. These are results of close cooperation between cryptographers and chip designers. At the system design level, cryptography optimizations and equivalence transformations lead to a very efficient floor plan with minimal routing, which otherwise would present a serious problem for data scrambling algorithms. These optimizations, which do not compromise the DES algorithm or the security, are combined with a highly structured design and layout strategy. Novel CAD tools are used at different steps in the design process. The result is a single chip of 25 mm^2 in $3\text{-}\mu\text{m}$ double-metal CMOS. Functionality tests show that a clock of 16.7 MHz can be applied, which means that a 32-Mbit/s data rate can be achieved for all eight byte modes. This is the fastest DES chip reported yet, allowing equally fast execution of all four DES modes of operation due to original pipeline architecture.

Ivey et al. [115] described the architecture and design of a public-key encryption processor which implemented the RSA algorithm with key lengths of 512 bits. The chip, which is 6.2 by 4.2 millimetres, has been designed in a 0.7 micron CMOS, silicon on insulator process and has a target clock speed of 150MHz. It is a self contained subsystem which interfaces directly to standard microprocessors and capable of encrypting at rates well in excess of 64k baud (for contractual reasons the authors are unable, at this time, to disclose the exact speed of operation). The chip contains 50,000 gates and has been designed using a custom methodology with the CADENCE design tools.

In recent years, there has been a tremendous upsurge in information and data transfers over the telephone and computer networks. This information ranges from very simple electronic mail to highly complex medical imaging. In any of these data transfers, the security

of the information is a pressing concern. Currently, the solution to the security concern is to use expensive and inefficient private networks and leased lines. The current evolution of the public Integrated Services Digital Network (ISDN) with its complete end to end digital connectivity provides an excellent platform for networking and reliable data communications. The primary objective of this work is to develop and implement methodologies for successful data encryption schemes which can be embedded into the ISDN Customer Premises Equipment (CPE) and require no software upgrades of the switching equipment. This will make the public ISDN network look like a private network to the security conscious user. Furthermore, secure data communication is provided over circuit switched voice or data channels. Widely used encryption and keying schemes based upon Data Encryption Standard (DES), secret key cryptography, and Rivest-Shamir-Adleman (RSA), public key cryptography, algorithms are currently being investigated for their applicability in the ISDN environment. Initial investigations show that a HYBRID cryptographic approach, RSA for authentication, and DES for encryption, may be most appropriate. Efforts are on to develop a hardware and software implementation for the HYBRID approach. Latif, Mahboob and Daram [61] discussed possible standards for ISDN security that will allow data (including voice) transmitted over the ISDN Basic Rate Interface (BRI) line to be encrypted so that only the intended receiver can decipher it. The ideas presented here can easily be transported to the packet switched channels, Primary Rate Interface (PRI) and possibly to Broadband ISDN (BISDN).

Blaze [75] has described an efficient key management in an encrypting file system in the year 1994. As distributed computing systems grow in size, complexity and variety of application, the problem of protecting sensitive data from unauthorized disclosure and tampering becomes increasingly important. Cryptographic techniques can play an important role in protecting communication links and file data, since access to data can be limited to those who hold the proper key. In the case of file data, however, the routine use of encryption facilities often places the organizational requirements of information security in opposition to those of information management. Since strong encryption implies that only the holders of the cryptographic key have access to the clear-text data, an organization may be denied the use of its own critical business records if the key used to encrypt these records becomes unavailable (e.g., through the accidental death of the key holder). This paper describes a system, based on cryptographic "smartcards," for the temporary "escrow" of file encryption keys for critical files in a cryptographic file system. Unlike conventional escrow schemes, this system is bilaterally auditable, in that the holder of an escrowed key can verify that, in

fact, the key is kept to a particular directory and the owner of the key can verify, when the escrow period is ended, that the escrow agent has neither used the key nor can use it in the future.

Rhazr [74] in 1995 again described a simple protocol, the Remotely Keyed Encryption Protocol (RKEP), which enables a secure, but bandwidth limited, cryptographic smart card to function as a high-bandwidth secret key encryption and decryption engine for an insecure, but fast, host processor. The host processor assumes most of the computational and bandwidth burden of each cryptographic operation without ever learning the secret key stored on the card. By varying the parameter of the protocol, arbitrary size blocks can be processed by the host with only a single small message exchange with the card and minimal card computation. RKEP works with any conventional block cipher and requires only standard ECB mode block cipher operations on the smartcard, permitting its implementation with off-the-shelf components. There is no storage overhead. Computational overhead is minimal and includes the calculation of a cryptographic hash function as well as conventional cipher function on the host processor.

Kaps and Paar [85] were the first in its type of cryptography with FPGA described fast DES implementation for FPGAs and its application to a universal key-search machine in the year 1999. Most security protocol and security applications are defined to be algorithm independent, that is, they allow a choice from a set of cryptographic algorithms for the same function. Therefore a key-search machine which is also defined to be algorithm independent might be interesting. Authors searched the feasibility of a universal key-search machine using the Data Encryption Standard (DES) as an example algorithm. Field Programmable Gate Arrays (FPGA) provides an ideal match for an algorithm-independent cracker as they can switch algorithms on-the-fly and run much faster than software. Authors designed, implemented and compared various architecture options of DES with strong emphasis on high-speed performance. Techniques like pipelining and loop unrolling were used and their effectiveness for DES on FPGAs investigated. The most interesting results is that it could achieve data rates up to 403 Mbits/s using standard Xilinx FPGA. This result is by a factor 31 faster than software implementations while authors are still maintaining flexibility. A DES cracker chip based on this design could search 6.29 million keys per second.

Burke, McDonald and Austin [42] gave the architectural support for fast symmetric-key cryptography in the year 2000. The emergence of the Internet as a trusted medium for commerce and communication has made cryptography an essential component of modern information systems. Cryptography provides the mechanisms necessary to implement

accountability, accuracy, and confidentiality in communication. As demands for secure communication bandwidth grow, efficient cryptographic processing will become increasingly vital for good system performance. In this paper, authors explored techniques to improve the performance of symmetric key cipher algorithms. Eight popular strong encryption algorithms are examined in detail. Analysis reveals the algorithms are computationally complex and contain little parallelism. Overall throughput on a high-end microprocessor is quite poor, a 600 MHz processor is incapable of saturating a T3 communication line with 3DES (triple DES) encrypted data. Authors introduce new instructions that improve the efficiency of the analyzed algorithms. The approach adds instruction set support for fast substitutions, general permutations, rotates, and modular arithmetic. Performance analysis of the optimized ciphers shows an overall speedup of 59% over a baseline machine with rotate instructions and 74% speedup over a baseline without rotate. Even higher speedups are demonstrated with optimized substitutions (SBOXes) and additional functional unit resources. Authors' analyses of the original and optimized algorithms suggest future directions for the design of high-performance programmable cryptographic processors.

Canda, Tring and Magliveras [45] introduced a new family of symmetric block cipher based on group bases. The main advantage of this approach is full scalability. It enables to construct, for instance, a trivial 8-bit Caesar cipher as well as strong 256-bit cipher with 312-bit key, both from the same specification. Authors discussed the practical aspects of the design, especially the choice of carrier groups, generation of random group bases and an efficient factorization algorithm. Authors also described how the cryptographic properties of the system are optimized, and analyze the influence of parameters on its security. Finally some experimental results regarding the speed and security of concrete ciphers from the family has also been presented.

Gan, Simmons and Tavares [58] introduced a new family of stream ciphers based on cascaded small S-Boxes in year 2000. Many stream cipher designs based on linear feedback shift registers (LFSRs) with non-linear combining functions are susceptible to various versions of the correlation attack. In this paper authors proposed a new model for stream ciphers which does not make use of LFSRs. Instead, these stream ciphers are based on a cascade of small substitution boxes (s-boxes). Like the RC4 stream cipher designed by Ron Rivest, the cascade stream cipher makes use of evolving s-boxes and pointers. However, instead of using one large s-box authors employ a cascade of several small s-boxes. Two parameters of this family of stream ciphers are the size of the individual s-boxes and the length of the cascade. If n -bit s-boxes is used, then each output of the stream cipher is an n -

bit block. A cascade consisting of 16 2-bit s-boxes would have an effective key length which is adequate for most practical applications. The number of s-boxes in the cascade can be increased if desired more security. Authors' studied indicated that the cascade cipher has good statistical properties. This cascade stream cipher requires relatively little storage and executes efficiently in both hardware and software.

Gaj and Pawel [84] gave comparison of the hardware performance of the AES proposals using reconfigurable hardware in the year 2000. The results of implementations of all five AES systems using Xilinx Field Programmable Gate Arrays are presented and analyzed. Performance of four alternative hardware architectures is discussed and compared. The AES proposals are divided into three classes depending on their hardware performance characteristics. Recommendation regarding the optimum choice of the algorithms for AES is provided.

Raghuram and Chakrabarti [105] proposed a programmable processor for cryptography. Cryptographic protocols have numerous applications in today's world, the most prevalent one being transferring messages safely over the network. Cryptographic algorithms are either implemented in software on a general-purpose processor or in hardware on an application-specific processor. While the software implementations tend to be time consuming, the hardware implementations are too specific and cannot even support small modifications. In this paper, a programmable architecture that can handle a large number of algorithms including DES, RSA, Blowfish, SAFER, et cetera have been developed. The architecture consists of addition, subtraction, modular multiplication, and exponentiation and XOR units and thus can support a majority of the cryptographic algorithms. A high data rate is achieved by applying loop unrolling to the Montgomery algorithm that is used for modular multiplication and exponentiation. The differences in the number of bits, key length, and sequence of operations are handled by the micro-programmed control unit. A VHDL model has been developed and synthesized using Auto-Logic II from Mentor Graphics. The results show a frequency of operation of 77 Megahertz and an area of 23,000 "Optimization COST" units.

Groszschädel [116] proposed the application of Chinese remainder theorem in a high speed RSA crypto chip in 2000. The performance of RSA hardware is primarily determined by an efficient implementation of the long integer modular arithmetic and the ability to utilize the Chinese Remainder Theorem (CRT) for the private key operations. This paper presents the multiplier architecture of the RSA crypto chip, a high-speed hardware accelerator for long integer modular arithmetic. The RSA multiplier data path is reconfigurable to execute either

one 1024-bit modular exponentiation or two 512-bit modular exponentiations in parallel. Another significant characteristic of the multiplier core is its high degree of parallelism. The actual RSA prototype contains a 1056×16 bit word-serial multiplier which is optimized for modular multiplications according to Barrett's modular reduction method. The multiplier core is dimensioned for a clock frequency of 200 MHz and requires 227 clock cycles for a single 1024-bit modular multiplication. Pipelining in the highly parallel long integer unit allows achieving a decryption rate of 560 kbits/sec for a 1024-bit exponent. In CRT-mode, the multiplier executes two 512-bit modular exponentiations in parallel, which increases the decryption rate by a factor of 3.5 to almost 2 Mbits/sec.

Software-efficient stream ciphers have been proposed by Halevi, Coppersmith and Jutla [59] in year 2002. The design of Scream, a new software-efficient stream cipher, which was designed to be more secured seal⁴. The design of Scream resembles in many ways a block-cipher design. The new cipher is as fast as SEAL, but it offers a significantly higher security level. In the process of designing this cipher, authors re-visit the SEAL design paradigm, exhibiting some tradeoffs and limitations.

Chau and Siu [65] proposed an Internet security system for E-Commerce. Network security is increasing in importance as result in the enormous use of electronic communication in business activities. RSA is the most widely used public-key cryptography algorithm in the e-commerce. In this paper, a new system is proposed to provide secure communications through the Internet. The proposed approach focuses on two main technologies. First, a session key approach is used to enhance the security performance of RSA. Second, JDBC tool is used to perform client/server data access efficiently and effectively. The system was implemented provided a reliable and secure Internet environment successfully for business activities around the world.

A tradeoffs analysis of FPGA based elliptic curve cryptography has been proposed by Bednara et al [76] in the year 2002. FPGAs are an attractive platform for elliptic curve cryptography hardware. Since field multiplication is the most critical operation in elliptic curve cryptography, authors have studied how efficient several field multipliers can be mapped to lookup table based FPGAs. Furthermore authors have compared various curve coordinates representations with respect to the number of required field operations, and show how an elliptic curve coprocessor based on the Montgomery algorithm for curve multiplication can be implemented using this generic coprocessor architecture.

Paquet [44] has described 'Sinople', a shared-key (symmetric) block cipher supporting 128-bit data blocks and 128-bit key size in the year 2003. Sinople is designed to

take advantage of the 32-bit operations supported in today's computers and its original design tries to improve security against differential and linear attacks.

Billet and Gilbert [47] proposed a new symmetric block cipher with the following paradoxical traceability properties: it is computationally easy to derive many equivalent secret keys providing distinct descriptions of the same instance of the block cipher. But it is computationally complex, given one or even up to k equivalent keys, to recover the so called meta-key from which they were derived, or to find any additional equivalent key, or more generally to forge any new untraceable description of the same instance of the block cipher. Therefore, if each legitimate user of a digital content distribution system based on encrypted information broadcast (e.g. scrambled pay TV, distribution over the Internet of multimedia content, etc.) is provided with one of the equivalent keys, one can use this personal key to decrypt the content. But it is infeasible for coalitions of up to k traitors to mix their legitimate personal keys into untraceable keys they might redistribute anonymously to pirate decoders. Thus, the proposed block cipher inherently provides an efficient traitor tracing scheme. This algorithm can be described as an iterative block cipher belonging to the class of multivariate schemes. It has advantages in terms of performance over existing traitor tracing schemes and furthermore, it allows restricting overheads to one single block (i.e. typically 80 to 160 bits) per encrypted content payload. Its strength relies upon the difficulty of the "Isomorphism of Polynomials" problem, which has been extensively investigated over the past years. An initial security analysis is supplied.

Kim and Srinivasan [72] proposed simple support for IPsec tunnel model. Many corporate employees (those commonly known as road warriors) often access the resources in protected corporate intranets, while working remotely, through IPsec tunnels between their corporate VPN (Virtual Private Network) gateway and their remote hosts. With the proliferation of Wireless LANs, 3G wireless networks, and mobile workers, it becomes highly desirable for remote hosts to be able to move among multiple networks (IP subnets) freely, even across different air-interface technologies. Currently, IPsec does not support this movement without breaking and re-establishing of IPsec tunnels. Re-establishing IPsec tunnels could cause disruptions to applications currently running across the tunnels, in addition to incurring the overhead of a 3 to 6 roundtrip handshake for a new tunnel establishment. One solution could be to run IPsec tunnels over MobileIP to enable mobility. However, that is inefficient due to the double tunnelling, which is especially an issue for resource-limited wireless networks. Authors explored modifying an IPsec implementation to enable mobility without compromising security and without incurring tunnel-re-establishment

at handoff. Authors do not intend to address the general issue of secure mobility support for the Internet. Instead, it is to focus on a single scenario of VPN remote access via IPsec ESP-only tunnel mode in IPv4, which has a large commercial application of the secure remote access of corporate intranets. Authors' approach is to change the tunnel endpoint IP address of the mobile host at the IPsec VPN gateway via a secure signalling, which is possible with minor modifications to how IPsec operates. To this end, authors modified FreeS/WAN v1.8, an open-source implementation of IPsec. The dependence of identifying a Security Association on the outer header destination address has also been removed so that the same security parameters can be used even in the new network. Two new private messages are added to ISAKMP (Internet Security Association and Key Management Protocol) to enable the required signalling to update new tunnel endpoint addresses. Authors' approach neither compromises the security of IPsec, nor requires changes to the existing IPsec standard, preserving interoperability with mobility-unaware hosts and gateways. Authors describe a working implementation of these modifications, discuss the performance of this approach, and compare with the standard IPsec and IPsec over MobileIP.

Rouvroy et al [77] proposed a compact and efficient encryption/decryption module for FPGA implementation of the AES Rijndael which is very well suited for small embedded application in the year 2003. Hardware implementations of the Advanced Encryption Standard (AES) Rijndael algorithm have recently been the object of an intensive evaluation. Several papers described efficient architectures for ASICs and FPGAs. In this context, the highest effort has been devoted to high throughput (up to 20 Gbps) encryption in designs, fewer works studied low area encryption in architectures and only a few papers have investigated low area encryption/decryption structures. However, in practice, only a few applications need throughput up to 20 Gbps while flexible and low cost encryption/decryption solutions are needed to protect sensible data, especially for embedded hardware applications. This paper proposed an efficient solution to combine Rijndael encryption and decryption in one FPGA design, with a strong focus on low area constraints. The proposed design fits into the smallest Xilinx FPGAs, deals with data streams of 208 Mbps, uses 163 slices and 3 RAM blocks and improves by 68% the best-known similar designs in terms of ratio Throughput/Area. Authors also proposed implementations in other FPGA families (Xilinx Virtex-II) and comparisons with similar DES, triple-DES and AES implementations has also been done.

Chodowiec and Gaj [87] presented a compact FPGA architecture for the AES algorithm with 128-bit key targeted for low-cost embedded applications. Encryption,

decryption and key schedule are all implemented using small resources of only 222 Slices and 3 Block RAMs. This implementation easily fits in a low-cost Xilinx Spartan II XC2S30 FPGA. This implementation can encrypt/decrypt data streams of 150 Mbps, which satisfies the needs of most embedded applications, including wireless communication. Specific features of Spartan II FPGAs enabling compact logic implementation are explored, and a new way of implementing MixColumns and InvMixColumns transformations using shared logic resources is presented.

Network data is, currently, often encrypted at a low level. In addition, the majority of future networks will use low-layer (IP level) encryption. Current trends imply that future networks are likely to be dominated by mobile terminals, thus, the power consumption and electromagnetic emissions aspects of encryption devices will be critical. Sotiriou and Papaefstathiou [100] presents several realisations of the DES algorithm, both in software and in hardware. Authors present software implementations of the algorithm running on the state-of-the-art Intel IXP network processor and seven hardware realisations based on a standard-cell library. The software implementations are based on the Intel MP Network platform, for which authors wrote assembly code implementing the DES algorithm utilising its various features. The hardware implementations consist of seven different hardware implementations, three clocked implementations and four un-clocked (asynchronous) implementations. The efficient realisation of the DES algorithm is a hardware realisations through conventional clocked designs, which is an asynchronous one compared to other four designs. Authors demonstrated that the most efficient realisation of the algorithm is a hardware implementation which is an asynchronous one.

Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication and data origin authentication. Rahman, Talkhan and Shaheen [120] introduced a new system that would help crypto-designer in their work towards implementing unbreakable encryption algorithm in easy way. The system consists of a language called "Cryptography Language" or CL, CL compiler and CL converter. Any designer can use this kit easily, to record any algorithm and implement in either software or hardware product. Using CL, designer can write many algorithms and give their code to CL compiler that will compile the code and extract all the algorithm information. Using CL converters, the algorithm can be generated in any software or hardware languages depending on their use.

Dutta and Mandal [12] proposed a bit-level secret-key block cipher which follows the principle of substitution in 2004. The decimal equivalent of the block under consideration is

evaluated and the modulo-2 operation is performed to check if the integral value is even or odd. Then the position of that integral value in the series of natural even or odd numbers is evaluated. The same process is repeated again with this positional value. This process is carried out recursively for finite number of times, equal to the length of the source block. After each modulo-2 operation, 0 or 1 is pushed to the output stream in MSB to LSB direction depending on whether the integral value is even or odd, respectively. During decryption, bits in the target block are to be considered along LSB to MSB direction after which an integral value is got, the binary equivalent of which is the source block.

Sinha and Mandal [17] proposed a novel block cipher based on a microprocessor system where the encryption is done through Overlapped Modulo Arithmetic Technique (OMAT). The original message is considered as a stream of bits, which is then divided into a number of blocks, each containing 'n' bits, where 'n' is one of 2, 4, 8, 16, 32, 64, 128, and 256. The two adjacent blocks are then added where the modulus of addition is 2^n . The result replaces the second block, first block remaining unchanged. The modulo addition has been implemented in a very simple manner where carry out of the MSB is discarded to get the result. The technique is applied in a cascaded manner by varying the block size from 2 to 256. The whole technique has been implemented through a microprocessor-based system by using a modulo subtraction for decryption.

Diaz-Perez, Saqib and Rodriguez-Henriquez [41] identified the basic characteristics of cryptographic algorithms especially symmetric block ciphers for their implementation on hardware platforms. The basic primitives in symmetric ciphers are discussed and some implementation techniques are suggested for them. As an application, an FPGA implementation of DES is presented which achieves a throughput of 274 Mbits/s occupy just 165 CLB slices for a single round. The same guidelines well hold for other block ciphers like Advanced Encryption Standard (AES).

Linear cryptanalysis has been proven to be a powerful attack which can be applied to a number of symmetric block ciphers. However, conventional linear cryptanalysis is ineffective in attacking ciphers that use key-dependent operations, such as ICE, Lucifer and SAFER. Dojen and Coffey [46] showed conditional linear cryptanalysis, which used characteristics that depended on some key-bit values. This technique has been described in detailed application to symmetric ciphers are also analysed. The consequences of using key-dependent characteristics are explained and a formal notation of conditional linear cryptanalysis is presented. As a case study, conditional linear cryptanalysis is applied to the ICE cipher, which uses key-dependant operations to improve resistance against cryptanalysis.

A successful attack on ThinICE using the technique is presented. Further, experimental work supporting the effectiveness of conditional linear cryptanalysis also detailed in the article.

Kim [70] presented the design and implementation of a crypto processor, a special-purpose microprocessor optimized for the execution of cryptography algorithms. This crypto processor can be used for various security applications such as storage devices, embedded systems, network routers, security gateways using IPSec and SSL protocol, etc. The crypto processor consists of a 32-bit RISC processor block and coprocessor blocks dedicated to the AES, KASUMI, SEED, triple-DES private key crypto algorithms and ECC and RSA public key crypto algorithm. The dedicated coprocessor block permits fast execution of encryption, decryption, and key scheduling operations. The 32-bit RISC processor block can be used to execute various crypto algorithms such as Hash and other application programs such as user authentication and IC card interface. The crypto processor has been designed and implemented using an FPGA, and some parts of crypto algorithms have been fabricated as a single VLSI chip using 0.5µm CMOS technology. To test and demonstrate the capabilities of this chip, a custom board providing real-time data security for a data storage device has been developed.

The national institute of standards and technology (NIST) in U.S. has initiated a process to develop an Advanced Encryption Standard (AES) specifying a private-key algorithm based on a 128-bit block size as a replacement of the Data Encryption Standard (DES). Riaz and Heys [78] investigated the efficiency of two AES candidates, RC6 and CAST-256, from the hardware implementation perspective with Field Programmable Gate Array (FPGA) as the target technology. Authors' analysis and synthesis of the ciphers suggest that it would be desirable for FPGA implementation to have a simpler cipher design that makes use of simpler operation which not only posse's good cryptographic properties, but also make the overall cipher design efficient from the hardware implementation perspective.

Goots et al [98] proposed a fast DDP based cipher in the year 2004. Data-dependent (DD) permutations (DDP) that are very suitable to cheap hardware implementation have been introduced as a cryptographic primitive for the design of fast firmware and software encryption systems. DDP can be performed with so called controlled permutation boxes (CPB) which are fast while implemented in cheap hardware. The latter defined the efficiency of the embedding of CPB in microcontrollers and microprocessors when adding a new fast instruction that allows one to perform DDP. Software and firmware encryption algorithms combining DDP with fast arithmetic operations are described.

Jha, Mandal and Shakya [36] described a bit level symmetric encryption technique through Recursive Transposition Operation (RTO) in 2005 to enhance security of transmission. The technique considers a message as binary string on which a RTO is applied. A block of n bits is taken as a input stream, where n varies from 8 to 256, from a continuous stream of bits and the technique operates on it to generate the intermediate encrypted stream. The same operation is performed repeatedly for different block sizes as per the specification of a session key of a session to generate the final encrypted stream. It is a kind of block cipher and symmetric in nature hence, decoding is done following the same procedure. A comparison of the proposed technique with existing and industrially accepted RSA has also been done in terms of frequency distribution and homogeneity of source and encrypted file.

A fast stream cipher, MAJE# has been designed and developed with a variable key size of 128-bit or 256-bit. The randomness property of the stream cipher is analysed by using the statistical tests. The performance evaluation of the stream cipher is done in comparison with another fast stream cipher called JEROBOAM. The focus is to generate a long unpredictable key stream with better performance, which can be used for cryptographic applications. This stream cipher has been proposed by Mathew and Jacob [57] in the year 2005.

The system PACS (Picture Archiving and Communication System) which handles medical image saves patient's medical image information and transports them. This needs security processor for user's authentication and encrypted information based on PKI (Public Key Infrastructure). The DICOM (Digital Imaging Communications in Medicine) which is a standard of PACS's transmission has adopted using RSA (Rivest Shamir Adleman) in the authentication and transmission which is public key algorithm. However, in embedded medical image system using low power and restricted hardware resource, its long key size is a major problem on the hardware implementation and processing time. Moreover, the public key algorithm, ECC (Elliptic Curve Cryptography) provided higher security than those of RSA in the same key size. Park, Hwang and Kim [63] implemented the DICOM security standard RSA substituted for ECC. ECC is implemented on GF(2) using polynomial base. Finite field used Montgomery algorithm and Brunner Extended Euclidian algorithm. ECC point multiplication operating used Radix-4 scalar multiplication and authors verified it on ISE 6.2 software using the Xilinx Vertex 1000E FPGA. Finally, authors designed and verified it by semi-custom ASIC design.

RSA calculation architecture is proposed for FPGAs which addressed the issues of scalability, flexible performance, and silicon efficiency for the hardware acceleration of

Public Key crypto systems in the year 2005 by Fry and Langhammer [81]. Using techniques based on Montgomery math for exponentiation, the proposed RSA calculation architecture is compared with existing FPGA-based solutions for speed, FPGA utilisation, and scalability. The paper covered the RSA encryption algorithm, Montgomery math, basic FPGA technology, and the implementation details of the proposed RSA calculation architecture.

Jha and Mandal proposed a symmetric block cipher [15] in the year 2006. The technique considered a message as binary string on which a Cascaded Recursive Carry Addition and Key Rotation (CRCAKR) is applied. This technique used binary addition methodology. It is a kind of block cipher and symmetric in nature hence, decoding is done following the same procedure. A comparison of the technique with existing RSA and Triple DES has also been done in terms of frequency distribution and non-homogeneity of source and encrypted files.

An efficient identity-based cryptosystem for end-to-end mobile security has been proposed [66] in the year 2006 by Hsu and Chen. In the next generation mobile telecommunications, any third party that provides wireless data services (e.g. mobile banking) must have its own solution for end-to-end security. Existing mobile security mechanisms are based on public-key cryptosystems. The main concern in a public-key system is the authenticity of the public key. This issue can be resolved by identity-based (ID-based) cryptography where the public key of a user can be derived from public information which uniquely identifies the user. This paper proposed an efficient ID-based encryption algorithm. Authors actually implemented the ID-based encryption schemes and compared the performance to show the advantage of the approach. Study indicates that this solution outperforms a previously proposed algorithm by (20 – 35)%.

Ferrante [68] proposed a new model for maintaining security and privacy for patient information system in 2006. As the global Internet evolved into today's mobile and broadband service, it is anticipated that the applications of the services to support telemedicine and e-Health operations using these techniques will result in increasing healthcare benefits for all using a lower cost based solution. Precautions must be taken while making these changes to ensure that security technology keeps pace with the changes and provides the means by which satisfaction of HIPAA's privacy regulations can be assured. Consider the application of Wireless Communications in connecting medical professionals and patients through the ubiquitous web access arrangements. New products offered to patients and physicians alike are capable of transmitting vital signs, key blood test results for diabetics, blood pressure data, as well as the higher data requirements of X-Rays, MRIs,

ultrasounds, CAT scans, and more. But now things are changing. Cell phones, hotspots (802.11 access arrangements) offer opportunities for others to intercept private information if not protected adequately. Today's security offering for wireless hotspots such as the Wired Equivalent Privacy (WEP) offers some security and privacy but it is known to be broken and useful only temporarily and not for protection of vital medical information protection. Currently, chips are being developed which will offer a much stronger protection for personal wireless networks. This recommended standard, referred to as 802.11's WPA, is already supported within the Microsoft XP Operating System, and it will be enhanced and approved shortly by the latest recommended version of the standard offering WPA2 - for enterprise applications. The final version of 802.11 (WPA2) addresses practically all the vulnerabilities of WEP and more. However, now things are changing once more. A wideband wireless capability in all likelihood will supersede WiFi within the next five years will allow up to 75 Mbps data transfer rates and support connections to systems in the range of 30 miles or more under the right conditions. In addition, speeding ambulances and cars travelling at speeds in excess of 70 MPH will be more readily capable of interfacing at the higher data rates. WIMAX, the much-awaited technology that is expected to provide wireless broadband services on a Metropolitan Area Network (MAN) scale is going to be the next wave of evolution (802.16) as was the case for WiFi, the security is concerned here. Will the WiFi security offerings support that needed at these higher rates? All of this is yet to be assured. Thus privacy is once again of concern if the standards are not adequate. It is understood that the WPA2 will support both WiFi and WiMax security needs. As technology evolved, security must be enhanced and if the manufacturers of products can settle on non-proprietary representative devices to support the needs of the medical field, then it will be fine.

Yang, Dai and Yu [92] described the RCBA architecture, a specialized reconfigurable architecture for block cipher that bridges the cost and performance gap between general purpose and application specific architecture for block cipher. Authors present implementations for representative algorithms of block cipher such as DES, Rijndael and RC6 on RCBA architecture. System performance has been analyzed, and from this analysis it has been demonstrated that the RCBA architecture can achieve both high security and speed.

Eslami et al [114] proposed an area efficient universal cryptography processor for smart cards in 2006. Cryptographic circuits for smart cards and portable electronic devices provide user authentication and secure data communication. These circuits should, in general, occupy small chip area, consume low power, handle several cryptography algorithms, and provide acceptable performance. This paper presented, for the first time, a hardware

implementation of three standard cryptography algorithms on a universal architecture. The micro-coded cryptography processor targets smart card applications and implements both private key and public key algorithms and meets the power and performance specifications and is as small as 2.25 mm^2 in $0.18\text{-}\mu\text{m}$ 6LM CMOS. An algorithm is implemented by changing the contents of the memory blocks that are implemented in ferroelectric RAM (FeRAM). FeRAM allows non-volatile storage of the configuration bits, which are changed only when an algorithm instantiation is done.

A testing scheme has been proposed by Yang, Wu and Karri [118] for crypto-chips in the year 2006. Scan-based design for test (DFT) is a powerful testing scheme, but it can be used to retrieve the secrets stored in a crypto chip, thus compromising its security. On one hand, sacrificing the security for testability by using a traditional scan-based DFT restricts its use in privacy sensitive applications. On the other hand, sacrificing the testability for security by abandoning the scan-based DFT hurts the product quality. The security of a crypto chip comes from the small secret key stored in a few registers, and the testability of a crypto chip comes from the data path and control path implementing the crypto algorithm. Based on this key observation, the authors proposed a novel scan DFT architecture called "secure scan" that maintains the high test quality of traditional scan DFT without compromising the security. They used a hardware implementation of the advanced encryption standard (AES) to show that the traditional scan DFT scheme can compromise the secret key. It is seen that by using secure-scan DFT, neither the secret key nor the testability of the AES implementation is compromised.

Safeguarding intellectual property on FPGAs is a major challenge for its manufactures. It becomes difficult to add more FPGA security features for economical reasons. It is difficult to say whether entire users are ready to pay for these added features. But there can be no question that security features are absolutely essential for FPGA security. Dutta and Dutta [18] addressed some security scenarios in FPGAs and tries to find out why currently existing security features are inadequate. Finally a comparative analysis for hardware implementations of the authentication algorithm has been provided for FPGA as well as ASIC implementation. This paper has been proposed in the year 2007.

Reddy et al [21] explored the idea of protecting one or more machines on a server from other systems by filtering the IP packets. In addition, the IP packets are checked whether they are reaching their proper destination or not and whether the message is corrupt or not by calculating the IP checksum both at the sender and receiver side. All packet processing operations were implemented on a reconfigurable hardware platform i.e. Field

Programmable Gate Array (FPGA). The FPGA resource requirements are reported and the methodology employed for the system design, verification and implementation is described.

Das et al [23] presented an evolutionary approach to develop software ensuring information security in 2007. The system used two ciphering techniques. Recursive positional substitution based on prime-nonprime of cluster and Triangular encryption technique. The evolutionary approach incrementally develops different versions of the system steadily progressing towards making final version deliverable to the customer. Allowing cascaded implementation of two techniques which required a longer key space ensuring the key to be practically impossible to be broken. Both techniques used are block ciphers and of bit-level implementation. The observation on the used evolutionary approach is also presented, stating the possible scope of having an improved performance.

Paul, Dutta and Bhattacharya [24] presented a 191-bit substitution based block cipher which considered a file to be encrypted as a stream of bits. The cipher implements a storage efficient algorithm through which along a reduction in size has also been achieved.

Paul, Dutta and Bhattacharya [25] presented a substitution-based block cipher which considered a file to be encrypted as a stream of bits. The cipher implements a storage efficient algorithm using non-Boolean operations through which a reduction in size is also achieved in addition to encryption.

Cryptographic circuits for smart cards and portable electronic devices provide user authentication and secure data communication. These circuits should, in general, occupy small chip area, consume low power, handle several cryptographic algorithms and provide acceptable performance. Rahimunnisa and Lincy [26] presented a hardware implementation of three standard cryptographic algorithms on a universal architecture. The macro coded cryptography processor targets smart cards application and implements both private key and public key algorithms and meets the power and performance specification.

Iha and Mandal [27] proposed a symmetric block cipher technique in year 2007. The technique considered a message as binary string on which a cascaded recursive key rotation of a session key and addition of blocks (CRKRAB) is applied. A block of 'n' bits is taken as input stream, where 'n' varies from 4 to 256, from a continuous stream of bits and the technique operates on it in two phases, in first phase plain text is encrypted by using recursive key rotation of a session key and then encrypt the output in the second phase to generate the intermediate encrypted stream using addition of blocks. The same operation is performed repeatedly for different block sizes as per the specification of a session key of a session to generate the final encrypted stream. It is a kind of block cipher and symmetric in nature.

hence, decoding is done following the same procedure. A comparison of the proposed technique with the existing and industrially accepted RSA and TDES has also been done in terms of frequency distribution and homogeneity of source and encrypted files.

Jha and Mandal [35] proposed a symmetric block cipher technique in year 2007. The technique considered a message as binary string on which a recursive key rotation (RKR) is applied. A block of 'n' bits is taken as input stream, where 'n' varies from 8 to 256, from a continuous stream of bits and the technique operates on it to generate the intermediate encrypted stream. The basic characteristic of this RKR technique is the use of a key value. This technique directly involves all the bits of blocks in a Boolean operation. The same operation is performed repeatedly for different block sizes as per the specification of a session key of a session to generate the final encrypted stream. It is a kind of block cipher and symmetric in nature hence, decoding is done following the same procedure. A comparison of the proposed technique with existing and industrially accepted RSA has also been done in terms of frequency distribution and homogeneity of source and encrypted files.

Aziz and Ikram [88] proposed an FPGA-based AES-CCM crypto core for IEEE 802.11i architecture. The widespread adoption of IEEE 802.11 wireless networks has brought its security paradigm under active research. One of the important research areas in this field is the realization of fast and secure implementations of cryptographic algorithms. Under this work, implementation has been done for Advanced Encryption Standard (AES) efficient and low power Field Programmable Gate Arrays (FPGAs) whereby computational intensive cryptographic processes are offloaded from the main processor thus results in achieving high-speed secure wireless connectivity. The dedicated resources of Spartan-3 FPGAs have been effectively utilized to develop wider logic function which minimizes the critical paths by confining logic to single Configurable Logic Block (CLB), thus improving the performance, density and power consumption of the design. The resultant design consumes only 4 Block RAMs and 487 Slices to fit both AES cores and its key scheduling.

Chen et al [103] analyzed the reconfigurable design principles of public-key cryptography and the characteristics of modular arithmetic iteration process. According to the analysis of results, a structured-adaptive reconfigurable modular arithmetic unit for Public-key cryptography has been implemented, where architecture is able to support security parameters of both RSA and ECC (Fp) algorithms. Based on 0.18 micrometer standard cell library, the area of the chip is only 42000 μm^2 . Simulation results of post-synthesis indicate that the maximum operating clock frequency is 103.8 MHz, the 1024-bit RSA modular

exponential operation period is about 45ms, and the 192-bit ECC (Fp) point multiplication period is 17ms on average

Lim and Phillips [113] presented a new residue number system implementation of the RSA cryptosystem. The system runs on a low-area, low-power microprocessor where it has been extended with hardware support for residue arithmetic. When compared against a baseline implementation which uses non-RNS multi-precision methods, the RNS implementation executes in 67.7% fewer clock cycles. The hardware support requires 42.7% more gates than the base processor core.

Reberio and Mukhopadhyaya [1] proposed an efficient high speed implementation of an elliptic curve crypto processor (ECCP) for an FPGA platform in the year 2008. The main optimization goal for the ECCP is efficient implementation of the important underlying finite field primitives namely multiplication and inverse. The technique proposes maximum utilization of FPGA resources. Additionally improves scheduling of elliptic curve point arithmetic results in lower number of register files reducing the area required and the critical delay of the circuit. Through several comparisons with existing work it demonstrate that the combination of the above techniques helps realization of one of the fastest and compact elliptic curve crypto processor.

Online ciphers are those ciphers whose plaintext can be computed in real time by using length preservative encryption algorithm. HCBC1 and HCBC2 are two known example of hash based chaining of online block ciphers. The first construct is secure against chosen plaintext attack whereas the later is secure against chosen cipher text attack. Mr. Nandi [2] provided simple security analysis of these online ciphers. Authors also proposed two new more efficient chosen cipher-texts secure online ciphers modified-HCBC (MHCBC) and modified-CBC (MCBC).

The tiny encryption algorithm (TEA) has been developed as simple computer program for encryption. Kaps [3] gave the first design space exploration for hardware implementation of the extended tiny encryption algorithm. It presents efficient implementation of XTEA on FPGAs and ASICs for ultra low power implementation.

Daniel and Peter [4] presented speed records for AES software taking advantage of architecture-dependent reduction of instructions used to compute AES and micro-architecture dependent reduction of cycles used for those instructions. A wide varieties of common CPU architectures – amd64, ppc32, sparcv9 and x86 – are discussed in details, along with several specific micro-architectures.

Gorski and Lucks [6] presented in the year 2008 the two new attacks on round reduced version of AES. Authors presented first application of the related key boomerang attack on 7 and 9 rounds of AES-192. The 7-round attack requires only 2^{18} chosen plaintext and ciphertexts and needs $2^{67.5}$ encryption. Authors extend the attack to nine rounds of AES-192. This leaves data complexity of 2^{67} chosen plaintexts and ciphertexts using about $2^{64.33}$ encryption to break 9 rounds of AES-192.

Recently the concept of proxy re-encryption has been shown very useful in a number of applications, especially enforcing access control policies. In existing proxy re-encryption schemes, the delegates can decrypt all ciphertexts for the delegator after re-encryption by the proxy. Consequently, in order to implement fine-grained access control policies, the delegator needs to either use multiple key pairs or trust the proxy to behave honestly. Tang [8] extended this concept and proposed type-based proxy re-encryption, which enables the delegator to selectively delegate his decryption right to the delegate which only needs one key pair. As a result type-based proxy re-encryption enables the delegator to implement fine-grained policies with one key pair without any additional trust on the proxy. A security model for this concept is provided and formal definition for semantic security is also provided. Author proposed two type-based proxy re-encryption schemes: one is CPA secure with ciphertext privacy while the other is CCA secure without ciphertext privacy.

Jha and Mandal [33] proposed a symmetric block cipher technique in year 2008. The technique considers a message as binary string on which a CRKRKA is performed. A block of 'n' bits is taken as input stream, where 'n' varies from 8 to 256, from a continuous stream of bits and the technique operates on it to generate the intermediate encrypted stream. This technique directly involves all the bits of blocks in a Boolean operation and a session key. The same operation is performed repeatedly for different block sizes as per the specification of a session key of a session to generate the final encrypted stream. It is a kind of block cipher and symmetric in nature hence, decoding is done following the same procedure. A comparison of the proposed technique with existing and industrially accepted RSA and TDES has also been done in terms of frequency distribution and non-homogeneity of source and encrypted file.

Jha and Mandal [37] proposed a symmetric block cipher technique in year 2008. The technique considers a message as binary string on which Cascaded Recursive Bitwise Operation and Carry Addition on Blocks (CRBOCAB) is applied. A block of 'n' bits is taken as an input stream, where 'n' varies from 4 to 256, from a continuous stream of bits and the technique operates on it to generate the intermediate encrypted stream. The basic

characteristic of CRBOCAB technique is the use of a carry in second stage. The technique directly involves all the bits of blocks in a binary addition. The same operation is performed repeatedly for different block sizes as per the specification of a session key of a session to generate the final encrypted stream. It is a kind of block cipher and symmetric in nature hence, decoding is done following the same procedure. A comparison of the proposed technique with existing and industrially accepted RSA and Triple-DES has also been done in terms of frequency distribution and homogeneity of source and encrypted files.

A highly secure symmetric stream ciphering technique based on hopping of chaotic maps and orbits is introduced by Nasir and Zein [52]. The chaotic logistic and quadratic maps used to generate the cipher, while the tent map is used generate map hopping sequence. The key determines the initial conditions, orbits, and orbit hopping patterns. There are eight cipher-generating maps. Each has sixteen orbits. The stream cipher is produced by hopping between both maps and orbits. Detailed examples are provided. The results show that this technique is highly promising.

A novel approach for design of stream ciphers based on a combination of pseudo-randomness and randomness is proposed by Mihajevic and Imai [53] in 2008. The core element of the approach is a pseudo-random embedding of the random bits into the ciphertext. This embedding plays a role of a homophonic encoding and implies an additional communication overhead. Before its output to the public communication channel the ciphertext with the embedded random bits is also intentionally degraded via its exposure to a moderate noisy symmetric channel. The proposed design has potential of providing that complexity of recovering the secret key in the known plaintext attacking scenario is close to the complexity of recovering the secret key via the exhaustive search. Accordingly, the proposed approach can be considered as a trade-off between the increased security and decreased communication efficiency.

Paul, Dutta and Bhattacharya [89] presented a substitution-based block cipher that considers a file to be encrypted as a bit-stream. The cipher implements a storage efficient algorithm through which along with encryption a reduction in size is also achieved. As encryption is done at bit level, this algorithm can be implemented on any kind of files. A tendency of increase in execution time is observed. The proposed technique is compared with the existing International Data Encryption Algorithm (IDEA) with respect to execution time and degree of non-homogeneity. A generalized expression for the key space is formulated.

A novel programmable security processor for cryptography algorithms presented by Han et al [91]. The 16-bit length RISC-like instruction set and 3-stage pipeline provide low

code density, low hardware cost and low power consumption. Parallel on-chip lookup tables are integrated to obtain satisfactory performance of cryptographic processing. Chinese wireless local area network block cipher standard-SMS4 and NIST encryption standard-AES are implemented in this processor, and it is the first implementation of SMS4 based on a domain-specific programmable processor. To resist external attack on memories, a method for secure storage of round key is also proposed.

Saram and Khondram [28] in 2009 presented an in-depth juxtaposition of RSA and Elliptic Curve Cryptosystem (ECC) and provided an overview of the different trade-off involving in choosing between cryptosystems based on them. Authors offer ECC as a suitable alternative to RSA. Authors also presented experimental results quantifying the benefits of using ECC for public-key cryptosystem.

The security of Xwindows is usually divided into authentication/authorization of connections, and authorization of Xclient interactions. The first issue has been well-addressed in research through mechanism such as xhost and xauth. Uppuluri et al [31] discussed the approaches to the last issue, one of authorization. Authors presented taxonomy of different approach and discuss the effectiveness of a light-weight mechanism.

Datta and Mandal [34] presented a 123-bit private key based block cipher in the year 2009, RSBP, which is capable of encrypting files up to 11 MB. It is formulated on the basis of base-10 value corresponding to a block of bits, which is to be checked if it is prime or not. It results in an alteration of size for file being encrypted. Its executable performance is analyzed on the basis of execution time, graphical layout of frequency distribution of characters and Chi-Square values for varying degrees of freedom. RSBP is found to be highly comparable with existing cryptosystems.

Latif, Mahboob and Ikram [61] proposed a parameterized design of modular exponentiation on reconfigurable platforms for RSA cryptographic processor. Modular Exponentiation is at the heart of various arithmetic architectures used in most Public Key cryptography algorithms. Modular Exponentiation of large numbers requires excessive processing. An efficient implementation of Modular Exponentiation may help overcome the speed issues of Public Key Cryptography. In this work, the most promising technique of Montgomery modular exponentiation and its optimizations have been deeply explored. Authors have been able to achieve performances which are better than earlier published results. Full bit length modular exponentiations with different word sizes and radices are implemented and their results are shown.

Ping et al [71] proposed a key management scheme for ad-hoc networks in the year 2009. With initial trust in the system model, a new identity-based distributed key management scheme has been proposed. Verifiable secret sharing technology and blind short signatures are applied into this scheme. The scheme is composed of system initialization, update of a node's private key, share refreshing of system private key, discover of malicious node and key revocation. The model with initial trust makes the scheme more secure. The overhead of storage, communication and computation are reduced since the identity-based public key system is used. The verifiable secret share technology effectively prevents the node from behaving dishonestly. The update scheme of a node's private key provides mutual authentication. The blind short signature ensures the share of private key can be transmitted in the unsecured channel. The key revocation is simple and convenient since the valid time is added to the key. The analysis shows that the scheme outlined not only provides greater security, but also improves the efficiency than previous scheme for ad hoc networks.

Ghosh and Paul [90] presented a process by which one can secure any kind of file. The newly developed encryption algorithm is presented in this paper. With help of generated distinct blocks N -level, again some distinct blocks are regenerated for $(N + 1)$ -level. Thus, source stream or plain text, target stream or encrypted text will be generated and decrypted text will be gotten on the applying the reverse process. The algorithm can be implemented on any kind file as it is implemented in bit-level. The strength of the technique has been analyzed in this paper.

Pal and Mandal [93] proposed a four stage character/bit level encoding technique (RBCMCPCC), where the first three steps take input block of length 128 bit, 192bit and 256 bit, respectively, and generate intermediate blocks of the same lengths using identical lengths of keys in each step. The fourth stage generates final cipher blocks based on random session keys. Before execution of the fourth stage, all 256 bit blocks are passed through the chaining process to ensure the generation of non-identical intermediate stream. A two dimensional matrix based substitution and folding operation is performed in the first stage of encryption, whereas a three dimensional matrix based permutation and substitution operation is done in the next stage. Again, a four dimensional matrix oriented substitution and transposition operation is applied in the third stage. Finally, wrapping operation is done to produce the cipher text. To expound the effectiveness of the algorithm, the obtained results are compared with BAM, TDES, CTSCCT and RSA algorithms.

Lamba [54] proposed a design and analysis of stream ciphers for network security in the year 2010. This paper mainly analysis and describe the design issue of stream ciphers in

Network security as the streams are widely used to protecting the privacy of digital information. A variety of attacks against stream cipher exist; (algebraic and so on). These attacks have been very successful against a variety of stream ciphers. So in this paper efforts have been done to design and analyze stream ciphers. The main contribution is to design new stream ciphers through analysis of the algebraic immunity of Boolean functions and S-Boxes. In this paper, the cryptographic properties of non-linear transformation have been used for designing of stream ciphers. Many LFSR (Linear feedback Shift Register) based stream ciphers use non-linear Boolean function to destroy the linearity of the LFSR(s) output. Many of these designs have been broken by algebraic attacks. Here authors analyzed a popular and cryptographically significant class of nonlinear Boolean functions for their resistance to algebraic attacks.

Chen and Ge [56] proposed a novel stream cipher in the year 2010. Performance and key randomness are two key issues for this stream cipher systems. In this paper, a lightweight stream cipher with good randomness of key stream has been proposed. It utilizes a novel model named tree parity machine to generate streams. The most superiority of this scheme is that 128 bits could be generated based on one time of state rotation, and it passes the full ENT randomness test. Then, implementation of the proposed scheme for wireless sensor networks, TinyStream, has been presented. TinyStream is based on TinySec protocols, and consists of 65024 bytes ROM and 1659184 bytes RAM. Due to simple structure and small computation, TinyStream is considered to be good for secure communication applications in the resource constraint based WSNs.

Smart card is used increasingly and widely, security becomes a primary issue for information transmission. Public key cryptography is the main directions of research in smart card encryption. Analysis of the principles of public key cryptography, illustrates two typical cryptographies RSA and ECC by Peng and Fang [60]. Moreover, comparisons and discussions about public key sizes and the security required of these two cryptographic protocols. In the case of ECC's with smaller keys to provide high security and high speed in a low bandwidth, this paper selected ECC cryptographic protocol to implement the smart card encryption.

A fast public key algorithm of Knapsack type has been proposed in the year 2010 by Zhang and Xiaolin [64]. Public key cryptography system is not only successful in protecting the confidentiality of information transmission, but also a good solution to the key management issues. However, almost all knapsack-type public key cryptography has been proven unsafe. This paper analyzed and improved an easy solution of knapsack problem.

based on the fast public-key cryptographic algorithm. An improved algorithm is proposed due to high density backpack, so that it can resist the low-density subset and attacks. Analysis shows that in the premise of almost same efficiency of encryption and decryption, the improved algorithm is better than the original algorithm in security.

An enhanced FPGA implementation of the hummingbird cryptographic algorithm is proposed in 2010 by San and Nuray [80]. Hummingbird is a novel ultra-lightweight cryptographic algorithm aiming at resource-constrained devices. In this work, an enhanced hardware implementation of the Hummingbird cryptographic algorithm for low-cost Spartan-3 FPGA family. The enhancement is due to the introduction of the coprocessor approach. It is seen that all Virtex and Spartan FPGAs consist of many embedded memory blocks and this work explores the use of these functional blocks. The intrinsic serialise of the algorithm is exploited so that each step performs just one operation on the data. Authors compared the performance results with other reported FPGA implementations of the lightweight cryptographic algorithms. This work presents the smallest and the most efficient FPGA implementation of the Hummingbird cryptographic algorithm.

The concept of binary field arithmetic is used to generate block cipher proposed by Pal and Mandal [94] in 2010. The technique consists of five stages, where in each of first four stages binary field arithmetic based substitution technique along with key association process is used. The lengths of input and output blocks in these four stages are identical and they are 256 bit, 128 bit, 64bit and 32 bit, respectively. The last stage consists of a nonlinear S-box operation which may generate cipher block of length different from its input. In most of the cases the proposed algorithm generates space efficient cipher. At the time of decryption, a set of session keys are used in conjunction with the user input key.

A frame based symmetric key cryptography algorithm has been proposed in the year 2011 by Mandal and PalChoudhury [39]. There are huge numbers of algorithms available in symmetry key block cipher. All these algorithms have been used either complicated keys to produce cipher text from plain text or a complicated algorithms for it. The level of security of all algorithms is dependent on either number of iterations or length of keys. In this paper, a symmetry key block cipher algorithm has been proposed to encrypt plain text into cipher text or vice versa using a frame set. A comparative study have been made with RSA, DES, IDEA, BAM and other algorithms with Chi-square value, frequency distribution, bit ratio to check the security level of proposed algorithm. Finally, a comparison has been made for time complexity for encryption of plain text and decryption from cipher text with the well-known existing algorithms.

A linear cryptanalysis of block ciphers has been proposed in the year 2011 by Bogdanov and Rijmen [48]. Linear cryptanalysis, along with differential cryptanalysis, is an important tool to evaluate the security of block ciphers. This work introduced a novel extension of linear cryptanalysis – zero-correlation linear cryptanalysis – a technique applicable to many block cipher constructions. It is based on linear approximations with a correlation value of exactly zero. For a permutation on n bits, an algorithm of complexity $O(2^{n-1})$ is proposed for the exact evaluation of correlation. Non-trivial zero-correlation linear approximations are demonstrated for various block cipher structures including AES, balanced Feistel networks, Skipjack, CLEFIA, and CAST256. Using the zero-correlation linear cryptanalysis, a key-recovery attack is shown on 6 rounds of AES-192 and AES-256 as well as 13 rounds of CLEFIA-256.

A differential cryptanalysis of block ciphers has been proposed in the year 2011 by Blondeau and Gerard [50]. Differential cryptanalysis is a well-known statistical attack on block ciphers. Authors presented a generalisation of attack called multiple differential cryptanalysis. Authors also study the data complexity, the time complexity and the success probability of such an attack and it is experimentally validate the formulas on a reduced version. Finally, authors proposed a multiple differential cryptanalysis on 18-round PRESENT for both 80-bit and 128-bit master keys.

1.3 Problem Domain

Campbell [101] proposed a microprocessor based module to provide security in electronic fund transfer. Electronic Fund Transfer (EFT) is expected to grow in importance and to result in national interchange system. The potential for fraud in EFT is quite significant, and can be prevented by the use of cryptographic security techniques. A microprocessor based security module has been developed which serves as a CPU peripheral to perform all cryptographic functions which an EDP facility requires to secure its EFT operations.

Computer communication systems, local-area networks, interconnected local-area networks, and electronic mail systems are playing an increasingly important role in office automation, telecommunications, and factory automation. A microprocessor based cryptoprocessor has been proposed by Schloer [104]. Recent advances have made the technology of cryptography a viable tool for providing security. The DES-Data Encryption Standard as well as public-key systems have also been used extensively. The overall system structure and user

interface along with an overview of cryptography and a review of the design considerations are also presented by the author. The performance parameters like non-homogeneity test, frequency distribution graph are not evaluated in this paper.

Dutta and Mandal [12] proposed a bit-level secret-key block cipher which follows the principle of substitution in 2004. Avalanche ratio test is not calculated in this paper. Sinha and Mandal [17] proposed a novel block cipher based on a microprocessor system where the encryption is done through Overlapped Modulo Arithmetic Technique (OMAT) in 2004. The non-homogeneity test calculated in this paper is not good. Jha, Mandal and Shakya [36] described a bit level symmetric encryption technique through Recursive Transposition Operation (RTO) in 2005 to enhance security of transmission. Again avalanche ratio test is not calculated in this paper. Jha and Mandal proposed a symmetric block cipher [15] in the year 2006. The technique considered a message as binary string on which a Cascaded Recursive Carry Addition and Key Rotation (CRCAKR) is applied. The key-length calculated in this paper is not 128-bits which now considered a secure key length. Jha and Mandal [27] proposed a symmetric block cipher technique in year 2007. The technique considered a message as binary string on which a cascaded recursive key rotation of a session key and addition of blocks (CRKRAB) is applied. Jha and Mandal [35] proposed a symmetric block cipher technique in year 2007. The hardware performance parameter like HDL synthesis is not done in this paper. The technique considered a message as binary string on which a recursive key rotation (RKR) is applied. Jha and Mandal [33] proposed a symmetric block cipher technique in year 2008. The technique considers a message as binary string on which a CRKRKA is performed. The technique here is not been implemented in either in microprocessor based systems or FPGA based systems. Jha and Mandal [37] proposed a symmetric block cipher technique in year 2008. The technique considers a message as binary string on which Cascaded Recursive Bitwise Operation and Carry Addition on Blocks (CRBOCAB) is applied. Time-complexity like encryption time and decryption time is found to be non satisfactory in this paper.

These ten references given here specify the problem domain as follows

- Microprocessor can be used to develop crypto hardware or crypto processor for secure electronic fund transfer or as an embedded system to be used for security purpose.

- FPGA can be used to develop crypto hardware or crypto processor for secure electronic fund transfer or as an embedded system to be used for security purpose.
- Develop such techniques which are faster, and this can be achieved by using microprocessor based techniques and FPGA based techniques.
- Develop such techniques which are simpler which means techniques with low computational complexity.
- Development of Non-Fiestel block cipher [150] but still with good cryptographic parameters.
- Development of symmetric block ciphers that is the source stream repeats after some number of iterations.
- The cryptographic parameters like non-homogeneity test using Chi-Square test, frequency distribution, avalanche ratio test and key length have been achieved to satisfactory level in this thesis with respect to RSA.
- The algorithmic parameters like encryption time and decryption time has been achieved to satisfactory level in this thesis with respect to RSA.
- The hardware parameter like HDL synthesis report (both timing and component) has also been achieved to satisfactory level in this thesis with respect to RSA.

Therefore, the problem domain is to develop efficient microprocessor based techniques and FPGA-based techniques to be used in embedded system.

1.4 Proposed Methodology

Any mechanical or electrical system that is controlled by a computer working as part of an overall system is called embedded system. A general-purpose computer is made to perform a variety of functions. An embedded system, which may contain a high performance CPU than in general purpose computers one, has a set of specific tasks for which the system is made.

Embedded systems have grown tremendously in recent years. There are three important reasons of this.

First, integrated circuit (IC) capacities have increased to the point that both software processors and custom hardware processors now commonly coexist on a single IC.

Second, quality compilers and program size increases have led to the common use processor independent C, C++ and Java compilers and integrated design platforms in embedded system design.

Third, synthesis technology has advanced to the point that synthesis tools have become commonplace in the design of digital hardware.

Synthesis tools achieve nearly the same for hardware design as compilers achieve in software design. They allow the designer to describe desired functionality in high-level programming language, and they then automatically generate an efficient custom-hardware processor implementation.

Firstly, the performance of the algorithms is often crucial. One needs encryption algorithms to run at the transmission rates of the communication links. Slow running cryptographic algorithms translate into consumer dissatisfaction and inconvenience. On the other hand, fast running encryption might mean high product costs since traditionally, higher speeds were achieved through custom hardware devices.

Secondly, In addition to performance requirements, guaranteeing security is a formidable challenge. An encryption algorithm running on a general-purpose computer has only limited physical security, as the secure storage of keys in memory is difficult on most operating systems. On the other hand, hardware encryption devices can be securely encapsulated to prevent attackers from tampering with the system. Thus, custom hardware is the platform of choice for security protocol designers. Hardware solutions, however, come with the well-known drawback of reduced flexibility and potentially high costs. These drawbacks are especially prominent in security applications, which are designed using new security protocol paradigms.

Many of the new security protocols decouple the choice of cryptographic algorithm from the design of the protocol. Users of the protocol negotiate on the choice of algorithm to use for a particular secure session. The new devices to support these applications, then, must not only support a single cryptographic algorithm and protocol, but also must be "algorithm agile" that is, able to select from a variety of algorithms. For example, IPSec (the security standard for the Internet) allows to choosing out of a list of different symmetric as well asymmetric ciphers. Some of the symmetric-key algorithms are: DES, 3DES, Blowfish, CAST, IDEA, RC4, RC6, and so on. Thus, software-based systems would seem to be a better fit because of their flexibility. However, the security engineer is faced with a difficult choice.

Fortunately, many embedded processors combine the flexibility of software on general-purpose computers with the near-hardware speed and better physical security than general-purpose computers.

Embedded processors are already an integral part of many communications devices and their importance will continue to increase. Combine this with their flexibility to be programmed and their ability to perform arithmetic operations at moderate speeds, it is easy to see that they are a very promising platform to implement cryptographic algorithms.

Here, the focus is on the basics of cryptography and the implementation of cryptographic applications on embedded systems. In Section 2, it introduces the general theory and concepts of symmetric-key and public-key cryptography as well as the operations, which are most commonly performed. It will be shown that public-key operations are very computationally intensive and therefore require platforms, which have strong arithmetic capabilities. In Section 3, a survey of previous cryptographic implementations on embedded systems is presented, as well as some of the characteristics of the proposed algorithms. An overview of implementations of symmetric-key and public-key algorithms is given. Finally, it ends up with some conclusions.

The field of efficient algorithms for the implementation of cryptographic schemes is a very active one. However, essentially all cryptographic research is being conducted independent of hardware platforms, and little research focuses on algorithm optimization for specific processors.

Crypto algorithm can be implemented in either hardware or software. It is fairly easy to implement crypto algorithms in software, but such approach is typically too slow for real-time applications such as storage devices, embedded system, etc. Hence, for these kinds of applications, hardware always appears to be the ultimate choice of implementation. As coprocessors, they can offload time-consuming algorithms and reduce the computation bottleneck (Lejla et al., 2003). For any same operation and function, hardware implementation will always outperform implementation in timing performance. Crypto hardware accelerators are not only faster in general, but also offer at the same time more intrinsic security. Unlike software implementations, crypto hardware is resistant to physical tampering. This is one of the most important features of the crypto hardware. In addition, crypto hardware also cannot be cloned easily, hacked, modify, etc. Therefore, it is suitable to be used in many of the critical real-time applications.

1.5 Salient Features of the Thesis

The detailed study in literature survey reveals to me the following facts:

- Most of the algorithms/techniques implemented are software based; my intention is for hardware implementation for high speed, low area and much lesser power consumption.
- It is also learnt that still today most of the low end embedded system uses microprocessor as a driving device, so the candidate has also opted for microprocessor-based implementation.
- FPGA is the future of embedded systems and a lot of research is still awaited in this domain, so the work is a step towards the same.
- Most of the algorithms/techniques devised are based on Feistel block cipher [121, 122]; the proposed work is to design non-Feistel ciphers with better cryptographic and algorithmic parameters. These designs are suitable for embedded systems.
- Since symmetric ciphers are much faster and simpler design than asymmetric ciphers having the same properties so my design is based on symmetric ciphers. Symmetric ciphers are also suitable for embedded systems.

An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions, often with real-time computing constraints. It is usually embedded as a part of a complete device including hardware and embedded software. Embedded systems many of the common devices in use today. Embedded systems are commonly used in today's world ranges from portable MP4 player to most common mobile phones; others are iPod, DTH and many more. An embedded system is a combination of hardware and software that may have some mechanical components to perform some specific task. Embedded systems consist of small computerized parts within a large device that serves more general purpose. The programs and instruction written for embedded systems are called firmware and are stored in read-only memory or flash memory. In today's world the use of embedded system is common in everybody's life. Hence, the security concern in embedded systems is growing in exponential terms. Cryptography is one of the ways to provide security

in these embedded systems. So, the realization of this goal can be achieved through microprocessor based solution and also fast growing FPGA based solutions.

Any work is to be accepted widely requires some betterment than the existing systems. In this research work the devised techniques are compared with existing and industrially accepted asymmetric block cipher RSA (Rivest-Shamir-Adleman). Finally a security model is proposed in the end of the thesis. Thus the proposed work can be summarized as follows:

- To devise symmetric key cryptographic technique. As symmetric key cryptography is faster than that of asymmetric key cryptography which can be used for embedded systems. Symmetric key cryptography is also suitable for encryption of large data or files.
- Then these techniques are compared with existing algorithms such as RSA. The parameters are Chi-Square value, Frequency distribution, Encryption time, Decryption time, and Avalanche ratio.
- After satisfying the above parameters a set of techniques are then implemented in 8085 Microprocessor based systems.
- Again satisfying the same another set of techniques are then implemented in FPGA based system, both for the use of these techniques in embedded systems and also in general purpose computers.

1.6 Organization of the Thesis

This thesis consists of three parts, the first part is from Chapter two to Chapter three which contains two proposed microprocessor based solutions, in the second part from Chapter four to Chapter nine which contains a different set of six proposed FPGA based solutions and in the last part from Chapter ten and Chapter eleven models are proposed and conclusions are drawn.

In this thesis following eight new techniques are proposed:-

- Modified Recursive Modulo-2ⁿ And Key Rotation Technique (MRMKRT)
- Recursive Transposition Technique (RTT)
- Two Pass Replacement Technique (TPRT)

- Triangular Modulo Arithmetic Technique (TMAT)
- Recursively Oriented Block Addition and Substitution Technique (ROBAST)
- Shuffle-RAT (SRAT)
- Triple Sagacious Vanquish (TSV)
- Forward-Backward Overlapped Modulo Arithmetic Technique (MFBOMAT)

Among them MRMKRT and RTT are for microprocessor based implementations and TPRT, TMAT, ROBAST, SRAT, TSV and MFBOMAT are for FPGA based implementations.

Chapter two contains a proposal of microprocessor based solutions. In this chapter, a novel block cipher based on a microprocessor system has been proposed where the encryption is done through Modified Recursive Modulo- 2^n and Key Rotation Technique (MRMKRT) [143]. The original message is considered as a stream of bits, which is then divided into a number of blocks, each containing n bits, where n is any one of 2, 4, 8, 16, 32, 64, 128, 256. The two adjacent blocks are then added where the modulus of addition is 2^n . The result replaces the second block, first block remaining unchanged. After that the whole stream of bits is circular left rotated. The modulo addition has been implemented in a very simple manner where the carry out of the MSB is discarded to get the result. The technique is applied in a cascaded manner by varying the block size from 2 to 256. The whole technique has been implemented through a microprocessor-based system by using a modulo subtraction technique for decryption.

Chapter three is the final chapter based on microprocessor based cryptosystem. This chapter contains a generalized approach towards e-Security through a novel variable length block cipher based algorithm termed as Recursive Transposition Technique (RTT) [135]. The plain text is considered as a stream of bits, which is then divided into a number of blocks, each containing k (variable number of bits) bits. As it is a generalized approach so, $k = 2 * n$ or $k = (2 * n - 1)$ that is even or odd numbers of bits per block, where $n = \{\text{set of positive integers}\}$. A matrix is constituted taking two adjacent blocks then the two adjacent blocks are XORED and the result is replaced with second block, the first block remains unchanged. The same process is repeated for whole plain text. The technique is implemented in both microprocessor-based system and in high level programming language. This technique is simple and a comparable result has been found. This proposed technique shows more improvement in heterogeneous point of view than MRMKRT.

This Chapter four deals with FPGA-based solution. It's a generalized approach towards digital content protection through a novel block cipher based algorithm called Two Pass Replacement Technique (TPRT) [132]. The digital message or plain text is considered as a stream of bits, which is then divided into a number of blocks, each containing k bits. As it is a generalized approach so, $k = 2 * n$ or $k = (2 * n - 1)$ that is even or odd numbers of bits per block, where $n \in \{\text{set of positive integers}\}$. The two adjacent blocks are XORED and the result replaces the second block, the first block remains unchanged. The same process is repeated in whole message. The same round is again done in reversible manner that is the result of XORED operation between the last block and 2^{nd} last block replaces the second last block. The technique is implemented in both FPGA-based system and in high level programming language.

Chapter five describes a block cipher based new cryptosystem has been proposed, where the encryption is done through Triangular Modulo Arithmetic Technique (TMAT) [144], which consists of three phases. The original message is considered as a stream of bits. In Phase 1, bit stream is divided into a number of equal size blocks. Then the Triangular algorithm is performed on odd blocks [148, 149], where even blocks are remaining unchanged and together form a bit stream. In Phase 2, modulo arithmetic operation is performed on that bit stream, which is divided into a number of blocks, each containing n bits, n is 2^k , k is 1, 2, 3 and so on. Then modulo addition is performed between first and second block and the content of the second block is replaced by the result, where the first block is remain unchanged. This is continuing till the last block is changed and also for block size n is 2, 4, 8, and so on. In case of modulo addition the carry out of the MSB is discarded. In Phase 3, the Triangular algorithm is performed on even blocks but odd blocks are remaining unchanged and together form output stream. In case of decryption, reverse algorithm is used, where modulo subtraction technique is performed instead of performing modulo arithmetic technique.

Chapter six proposed FPGA based technique where a message is considered as a binary string on which the technique termed as Recursively Oriented Block Addition and Substitution Technique (ROBAST) [138] is applied. A block of n -bits is taken as an input stream, where n ranges from 8 to 256 – bit, then ROBAST is applied in each block to generate intermediate stream through iterative process, any one intermediate stream is considered as a cipher text. The same operation is performed repeatedly on various block sizes. It is a kind of block cipher and symmetric in nature hence decoding is done in similar manner. This chapter also presents an efficient hardware realization of the proposed

technique using state-of-the-art Field Programmable Gate Array (FPGA). The technique is also coded in C programming [129] language and Very High Speed Integrated Circuit Hardware Description Language (VHDL.) Various results and comparisons have been performed against industrially accepted RSA. A good result in terms of encryption time and decryption time has been found for this proposed technique, ROBAST than TPRT and TMAT.

In Chapter seven, an iterative block cipher based on rotational addition technique and butterfly-shuffle for confusion and diffusion respectively has been proposed. The design is an improvement over an existing design based on just rotational addition technique, and it achieves dramatic improvements in terms of diffusion and runtime efficiency over the existing one, this technique is called as Shuffle-RAT (SRAT) [139]. Efficient hardware architecture to implement the proposed design on FPGA, and perform VHDL-based simulation using Xilinx ISE has been constructed here. Using a C implementation in software, this technique has also been compared with popular ciphers like RSA to prove its competence. This proposed technique shows improvements in Chi-Square value and in diffusion than ROBAST, TMAT and TPRT.

Chapter eight also deals with FPGA-based solutions. In this chapter, a new block cipher, TRIPLE SV (3SV / TSV) [140], with 256-bit block size and 112-bit key length has been devised. Generally, stream ciphers produce higher avalanche effect but Triple SV shows a substantial rise in avalanche effect with a block cipher implementation. The CBC mode has been used to attain higher avalanche effect. The technique is implemented in C and VHDL and has been tested for feasibility. This technique shows a high value in avalanche ratio in compared to SRAT, ROBAST, TMAT and TPRT.

Chapter nine is the final FPGA-based solution, a new Cryptosystem based on block cipher has been proposed in this chapter where the encryption is done through Modified Forward Backward Overlapped Modulo Arithmetic Technique (MFBOMAT) [145]. The original message is considered as a stream of bits, which is then divided into a number of blocks, each containing n bits, where n is any one of 2, 4, 8, 16, 32, 64, 128, 256. The first and last blocks are then added where the modulus of addition is 2^n . The result replaces the last block (say N th block), first block remaining unchanged (Forward mode). In the next attempt the second and the N th block (the changed block) are added and the result replaces the second block (Backward mode). Again the second (the changed block) and the $(N-1)$ th block are added and the result replaces the $(N-1)$ th block (Forward mode). The modulo addition has been implemented in a very simple manner where the carry out of the MSB is

discarded to get the result. The technique is applied in a cascaded manner by varying the block size from 2 to 256. The whole technique has been implemented by using a modulo subtraction technique for decryption. MFBOMAT is giving much better result in all respect than previously proposed technique.

Chapter ten gives cryptographic models for microprocessor based systems and FPGA based systems and conclusions are drawn in Chapter eleven.

Section I
Microprocessor Based Solutions

Chapter 2

Modified Recursive Modulo-2ⁿ and Key Rotation Technique (MRMKRT)

2.1 Introduction

In this chapter, a novel block cipher based on a microprocessor system has been proposed where the encryption and decryption is done through Modified Recursive Modulo- 2^n and Key Rotation Technique (MRMKRT). The original message is considered as a stream of bits, which is then divided into a number of blocks, each containing n bits, where n is any one of 2, 4, 8, 16, 32, 64, 128, 256. The two adjacent blocks are then added where the modulus of addition is 2^n . The result replaces the second block, first block remaining unchanged. The modulo addition has been implemented in a very simple manner where the carry out of the MSB is discarded to get the result. After addition one bit left circular rotation is applied. The technique is applied in a cascaded manner by varying the block size from 2 to 256. The whole technique has been implemented through a microprocessor-based system by using a modulo subtraction technique for decryption.

For this implementation the stream size of 512 bits has been taken but the scheme may be implemented for larger stream sizes also. The input stream, S , is first broken into a number of blocks, each containing n bits ($n=2^k$, $k=1,2,3,\dots,8$) so that $S = B_1B_2B_3\dots B_m$, where $m=512/n$. Starting from the MSB, the blocks are paired as (B_1,B_2) , (B_2,B_3) , (B_3,B_4) and so on. The MRMKRT operation with modulo addition is applied to each pair of blocks, the result replaces the second block keeping first block intact. After addition one bit left circular rotation is applied. The process is repeated, each time increasing the block size till $n=256$. So, encryption is a process of converting intelligent message into stupid form. Therefore, decryption is the process of getting back the intelligent message from the stupid one. The proposed scheme has been implemented by using the reverse technique, i.e. modulo subtraction technique, for decryption. The flow of the work is to first implement all the proposed algorithms in C programming, the test for feasibility using frequency distribution test, test for non-homogeneity, time complexity analysis and avalanche ratio test. Then the proposed algorithms are implemented for microprocessor and FPGA.

Section 2.2 described the algorithm of MRMKRT in detail using block diagram, section 2.3 illustrates an example, section 2.4 shows how modulo addition is being done, section 2.5 deals with key generation issues, section 2.6 gives the algorithmic analysis of the scheme, section 2.7 gives generalized routine as a microprocessor based implementation, section 2.8 illustrates various results and its comparisons with existing RSA algorithm and section 2.9 gives a brief discussions.

2.2 The Algorithm of MRMKRT

The algorithm of MRMKRT is based on bit level encryption technique. A plaintext is taken for encryption in the sender side and ciphertext is taken for decryption in the receiver side. It is a bit level cipher so, during encryption plaintext is first broken down into a blocks of bits, let $B1 = \{a_0, a_1, a_2, \dots, a_{n-1}\}$, $B2 = \{b_0, b_1, b_2, \dots, b_{n-1}\}$, $B_m = \{\dots\}$, so here each block is n-bits in size and number of blocks are 'm' then MRMKRT encryption is performed which is again combined, $C1||C2||\dots||C_m$, here block B1 is converted to block C1 after MRMKRT encryption, block B2 is converted to block C2 after MRMKRT encryption and so on to block Bm is converted to Cm after MRMKRT encryption, hereto form ciphertext. During decryption ciphertext is broken down into blocks of bits, let $C1 = \{a_0, a_1, a_2, \dots, a_{n-1}\}$, $C2 = \{b_0, b_1, b_2, \dots, b_{n-1}\}$, $C_m = \{\dots\}$, so here each block is n-bits in size and number of blocks are 'm' then MRMKRT decryption is performed which is again combined, $B1||B2||\dots||B_m$, block B1 is regenerated from block C1 after MRMKRT decryption, block B2 is regenerated from block C2 after MRMKRT decryption and so on to block Bm is regenerated from block Cm, to form plaintext. A generalized approach is taken for explaining the algorithm of MRMKRT.

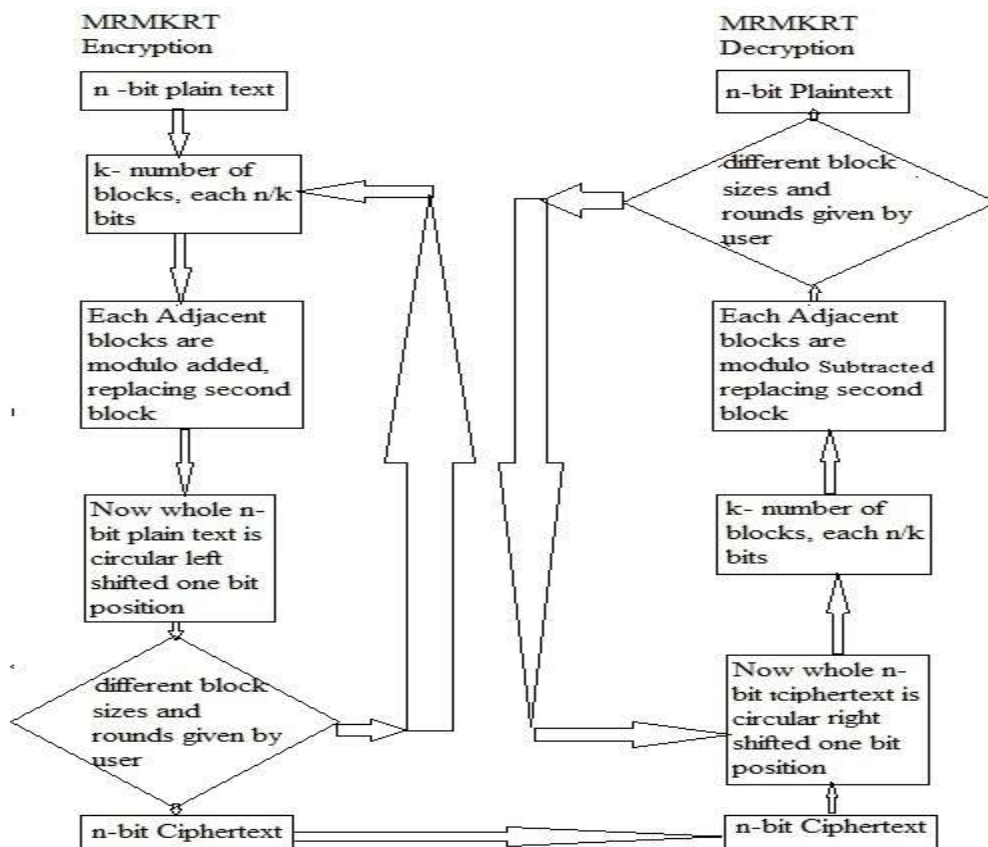


Figure 2.1: Modified recursive modulo- 2^n and key rotation technique (MRMKRT)

Figure 2.1 gives the block diagram of MRMKRT. The MRMKRT is defined with n -bit plaintext which is to be encrypted, ' k ' blocks with ' n/k ' bits per block. It has three main rounds/steps which are explained below:-

- *Round 1:* At first n -bit plaintext has been broken into k number of blocks and each block has n/k bits as given in figure 2.1 of the block diagram of MRMKRT, let the blocks are $B_1, B_2, B_3, \dots, B_k$, the following operations are performed starting from the most significant bit towards least significant bits.
- *Round 2:* In each pair of blocks, the first member of the pair, say block B_1 , is added to the second member, say block B_2 , where the modulus of addition is 2^m for block size m . Therefore for 2-bit blocks, the modulus of addition will be 4.
- *Round 3:* Now the whole n -bit text is left circular shifted/rotation by 1-bit position.

This round is repeated for a finite number of times and the number of iterations will form a part of the session key as discussed in section 2.5, which is given by the user.

So, in general the whole plaintext is broken down into two-bits block size, then modulo addition are performed and at last a one-bit left circular shift is performed. After that the same three operations are performed for 4-bit block size, i.e. the whole n -bit stream is now broken down into block of sizes 4-bits. In this fashion several rounds are completed till it reaches a round where the block size is 256 and the encrypted bit-stream is obtained. Since the original content of block B_i changes due the addition with block B_{i-1} , a new content of B_i is added to block B_{i+1} . This is due to the overlapping nature of the block-pairs, which increases the complexity of the algorithm resulting in the enhancement of security.

During decryption, the reverse operation, i.e. modulo subtraction, is performed instead of modulo addition, starting from the LSB and decreasing the block size from 256 to 2. At first whole n -bit ciphertext in right circular shifted/rotation by one-bit position, as shown in figure 2.1. Then the n -bit ciphertext is first broken into blocks of sizes 256-bits, then the two adjacent blocks, say B_1 , and B_2 , are modulo subtracted instead of addition, these three stems are repeated for a number of iterations and various block sizes given by the user, actually this forms the key.

2.3 Example

As discussed in section 2.2 MRMKRT encrypts n-bits of plaintext with ‘k’ blocks with ‘n/k’-bits per blocks. In this section 32-bit plaintext is considered as an example of MRMKRT, the whole encryption and decryption process is performed by the following four rounds:-

- *Round 1:* In first round, 16-blocks are taken for encryption and decryption, therefore block size is ‘ $32/16 = 2$ bits’ per block.
- *Round 2:* In second round, 8-blocks are taken for encryption and decryption, therefore block size is ‘ $32/8 = 4$ bits’ per block.
- *Round 3:* In third round, 4-blocks are taken for encryption and decryption, therefore block size is ‘ $32/4 = 8$ bits’ per block.
- *Round 4:* In fourth and final round, 2-blocks are taken for encryption and decryption, therefore block size is ‘ $32/2 = 16$ bits’ per block.

Consider a stream of 32 bits, say $S = 11010011000110111010011101000101$. The whole process of MRMKRT is described in four rounds in figure 2.2 to 2.5.

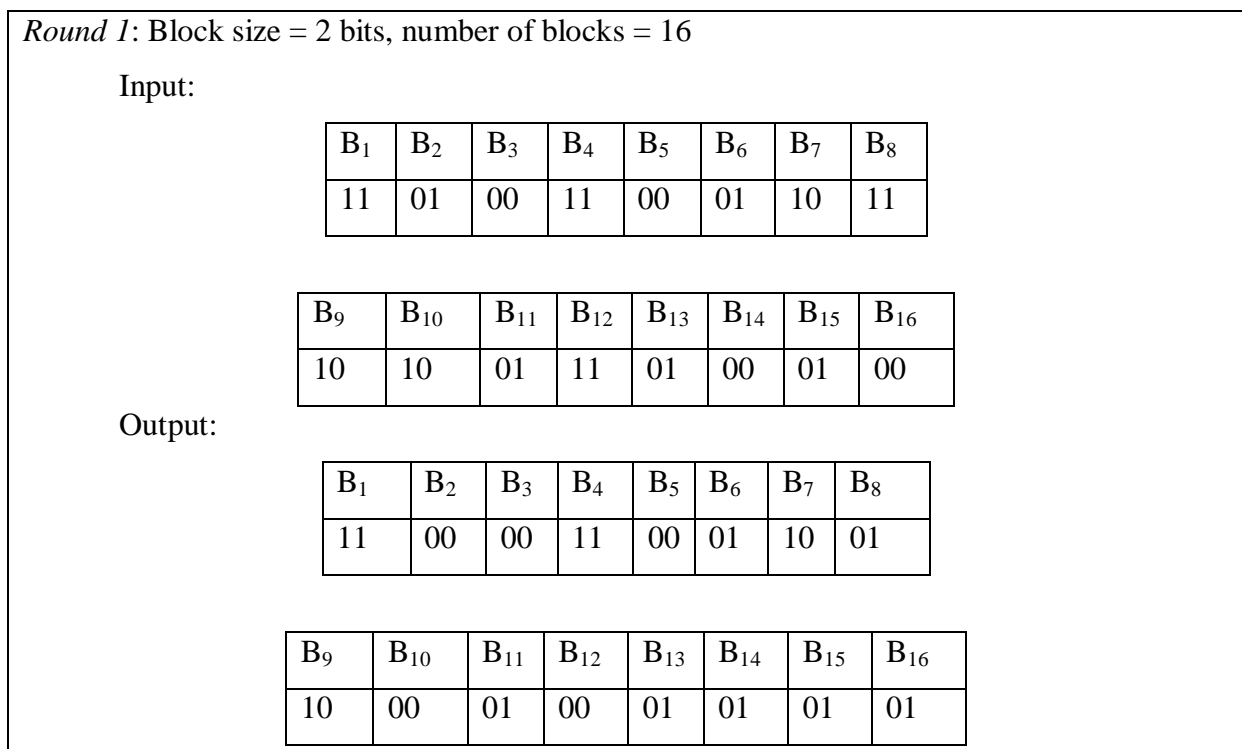


Figure 2.2: First round of MRMKRT

Figure 2.2 shows the first round of MRMKRT, the 32-bit plaintext is broken down into 16 numbers of blocks and each block is of size 2-bits. So, first block B1 has '11', next block B2 has '01' and so on until last block B16 has '00' as value was got, these are reflected in input blocks. Then the two adjacent blocks B1 and B2 are modulo-2 added, so, '11' + '01' mod 2 = '00', which is replacing the second block B2, the first block B1 is unaltered, this is reflected in Output block. The whole operation is performed for all the sixteen blocks. As a result the sub-stream as, X = 11000011000110011000010001010101. Now a one-bit left circular shift is performed. Which generates, X' = 10000110001100110000100010101011. This sub-stream will be now input to the round 2.

<i>Round 2: Block size = 4 bits, number of blocks = 8</i>							
Input:							
B ₁	B ₂	B ₃	B ₄	B ₅	B ₆	B ₇	B ₈
1000	0110	0011	0011	0000	1000	1010	1011
Output:							
B ₁	B ₂	B ₃	B ₄	B ₅	B ₆	B ₇	B ₈
1000	1110	0011	0110	0000	1000	1010	0101

Figure 2.3: Second round of MRMKRT

Figure 2.3 shows the second round of MRMKRT, where, the input blocks from round 1, X', this 32-bit stream is broken down into 8 numbers of blocks and each block is of size 4-bits. So, first block B1 has '1000', next block B2 has '0110' and so on until to get last block B8 has '1011' as value, these are reflected in input blocks. Then the two adjacent blocks B1 and B2 are then added as modulo-2⁴, so, '1000' + '0110' mod 2⁴ = '1110', which is replacing the second block B2, the first block B1 is unaltered, this is reflected in output block. This modulo addition is very clear when adding block B7 and Block B8, here '1010' + '1011' modulo 2⁴ = '0101'. The whole operation is performed for all the eight blocks. After this to get the sub-stream as, Y = 10001110001101100000100010100101. Now a one-bit left circular shift is performed. Y' = 00011100011011000001000101001011 was obtained. This sub-stream will be now input to the round 3.

Round 3: Block size = 8 bits, number of blocks = 4

Input:

B ₁	B ₂	B ₃	B ₄
00011100	01101100	00010001	01001011

Output:

B ₁	B ₂	B ₃	B ₄
00011100	01010010	00010001	01011100

Figure 2.4: Third round of MRMKRT

Figure 2.4 shows the third round of MRMKRT, the input blocks from round 2, Y' , this 32-bit stream is broken down into 4 numbers of blocks and each block is of size 8-bits. So, first block B₁ has '00011100', next block B₂ has '01101100' and so on until to get last block B₄ whose value '01001011' is generated. These are reflected in input blocks. Two adjacent blocks B₁ and B₂ are then modulo- 2^8 added, so, '00011100' + '01101100' mod 2^8 = '01010010', which is replacing the second block B₂, the first block B₁ is unaltered, this is reflected in Output block. The whole operation is performed for all the four blocks. After this to get the sub-stream as, $Z = 00011100010100100001000101011100$. Now a one-bit left circular shift is performed. $Z' = 00111000101001000010001010111000$ is obtained. This sub-stream will be now input to the round 4.

Round 4: Block size = 16 bits, number of blocks = 2

Input:

B ₁	B ₂
0011100010100100	0010001010111000

Output:

B ₁	B ₂
0011100010100100	0101101101011100

Figure 2.5: Fourth round of MRMKRT

Figure 2.5 shows the fourth round of MRMKRT, taking Z' as an input from the round 3, the whole 32-bit stream is now divided into two blocks each of size 16-bits. After modulo- 2^{16} addition to get the 32-bit stream as $C' = 00111000101001000101101101011100$. Now

performing a circular left shift of one bit, to get the final ciphertext as, C=01110001010010001011011010111000.

2.4 The Modulo Addition

An alternative method for modulo addition is proposed here to make the calculations simple. The need for computation of decimal equivalents of the blocks is avoided here since it will get large decimal integer values for large binary blocks. The method proposed here is just to discard the carry out of the MSB after the addition to get the result. For example, if to add 1101 and 1001 the result will be 10110. In terms of decimal values, $13+9=22$. Since the modulus of addition is 16 (2^4) in this case, the result of addition should be 6 ($22-16=6$). Discarding the carry from 10110 is equivalent to subtracting 10000 (i.e. 16 in decimal). So the result will be 0110, which is equivalent to 6 in decimal. The same is applicable to any block size.

2.5 Key Generation

In the proposed scheme, eight rounds have been considered, each for 2, 4, 8, 16, 32, 64, 128, and 256 block size. Each round is repeated for a finite number of times and the number of iterations will form a part of the encryption-key. Although the key may be formed in many ways, for the sake of brevity it is proposed to represent the number of iterations in each round by a 16-bit binary string. The binary strings are then concatenated to form a 128-bit key for a particular key. Example in section 2.5.1 illustrates the key generation process.

Table 2.1: Representation of number of iterations in each round by bits

Round	Block Size	Number of Iterations	
		Decimal	Binary
1.	256	50021	1100001101100101
2.	128	49870	1100001011001110
3.	64	48950	1011111100110110
4.	32	44443	1010110110011011
5.	16	46250	1011010010101010
6.	8	4321	0001000011100001
7.	4	690	0000001010110010
8.	2	72	0000000001001000

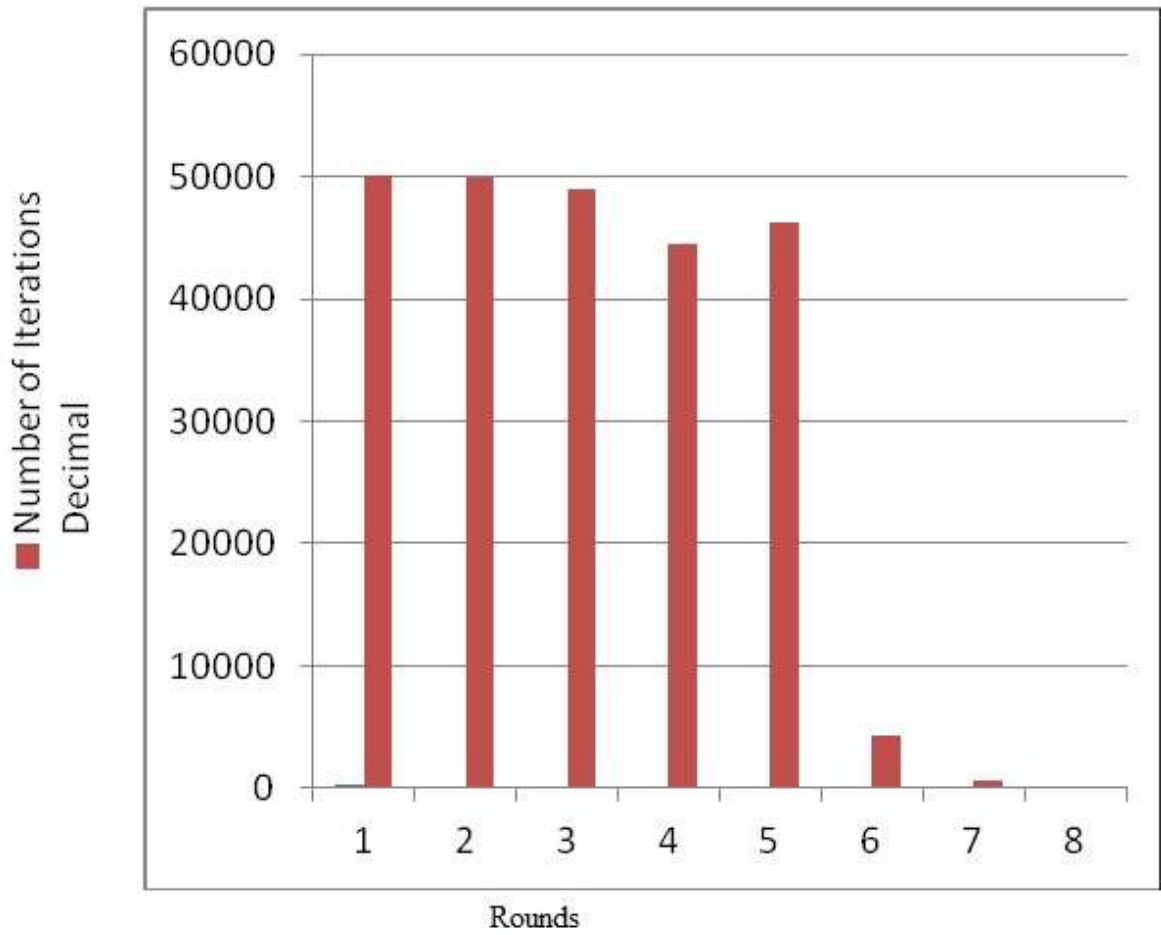


Figure 2.6: Round v/s iteration in MRMKRT

2.5.1 Example of Key Generation

Consider a particular session where the source file is encrypted using iterations for block sizes 2, 4, 8, 16, 32, 64, 128, and 256 bits, respectively. Table 2.1 shows the corresponding binary value for the number of iterations in each round. Figure 2.6 shows the graph for the round v/s iteration. When the block size is 2-bit then the process of MRMKRT is applied 72 times, for 4-bit block size the process of MRMKRT is applied 690 times, similarly when the block size is 256-bit then the process of MRMKRT is applied 50021 times. The number of times of iteration is solely decided by the user or sender of the secret message.

These numeric values are converted to equivalent binary strings and these binary strings are concatenated together to form the 128-bit binary string:

110000110110010111000010110011101011111100110110101011011001101110110100101
01010000100001110000100000010101100100000000001001000.

This 128-bit binary string will be the key for encryption for a particular session.

During decryption, the same key is taken to iterate each round of modulo-subtraction for the specified number of times.

2.6 Analysis

MRMKRT described here is symmetric in nature, that is same key is required for encryption and decryption. Symmetric ciphers are also those where the number of iterations/steps involved is the same in the decryption that is if ‘i’ is the number of iterations performed during encryption then the number of iteration required during decryption is also ‘i’, but this is not the case for MRMKRT encryption and decryption if modulo addition is considered in both cases. MRMKRT is off-course symmetric key/private-key cryptography where same key is used for both encryption and decryption.

Table 2.2: Plaintext and ciphertext pair in hex for single iteration of MRMKRT

Block Size	Input Plaintext	Output Ciphertext	Number of Iteration to Get Back the Original Plaintext
2-bits	D31BA745	863308AB	16
4-bits	863308AB	476844363	256
8-bits	476844363	38A422B8	4096
16-bits	38A422B8	7148B6B8	65536

Table 2.2 gives the plaintext and the corresponding ciphertext obtained based on executing single iteration of MRMKRT and also the number of iteration required to get back the original plaintext when modulo addition is performed. When block size is 2-bits, the plaintext is ‘D31BA745’ and the corresponding ciphertext is ‘863308AB’. The number of iteration required to get back the original plaintext with modulo addition is 16. In the next round, when block size is 4-bits, the plaintext is ‘863308AB’ and the corresponding ciphertext is ‘476844363’. The number of iteration required to get back the original plaintext with modulo addition is 256. In the next round, when block size is 8-bits, the plaintext is ‘476844363’ and the corresponding ciphertext is ‘38A422B8’. The number of iteration

required to get back the original plaintext with modulo addition is 4096. In the next round, when block size is 16-bits, the plaintext is '38A422B8' and the corresponding ciphertext is '7148B6B8'. The number of iteration required to get back the original plaintext with modulo addition is 65536. Thus by observing the table the order of time complexity of MRMKRT is $O(n^4)$. Therefore to minimize this time complexity the modulo-subtraction is proposed for decryption.

After the first rotation by one bit, msb has taken the lsb position and all other bits are shifted left by one bit. So eight such rotations are required to regenerate the original string for an 8 bit string and one of the seven intermediate strings can be used as encoded string. If the string generated after 2nd rotation is used as encoded string, then $(8 - 2) = 6$ more rotations are to applied on the encoded string to get back the original string.

The principle can be extended to n byte string. The number of rotation required to get back the original string for n byte string $(m) = n \times 8$, where n is the number of bytes in the string.

The total number of intermediately generated string, $(k) = (n \times 8 - 1)$

For $n = 1, k = 7$.

Considering that after i-th rotation, the generated string is used as encoded string. Then the number of rotations to be applied on the encoded string at the time of decoding, $l = n \times 8 - i$.

For $n = 1$ and $i = 2$, then $l = 6$.

When a large number of bytes are taken into consideration in the string, the rotational encoding will not be very effective. On 8th rotation, the MS byte will go to the LS byte position and all other bytes will be moved to the right. The characters in the string will appear again in the shifted condition and MS byte character will come to the LS byte position. On 16th rotation the same thing will happen. So after 8 and its multiple rotations the part of the message will reappear with cut and paste condition. This is the disadvantage with the rotational encoding.

On rotational encoding a modification is suggested here with a view to eliminate the disadvantage with the rotational encoding. Before applying the rotational encoding, a particular bit (say, lsb) of each byte of the string under consideration is complemented. This additional feature is very effective and will eliminate the disadvantage of re-appearing the bytes after 8 and its multiple rotations. This will also be very effective for any number of bytes. The encoding with large number of bytes with a particular bit inverted will be more effective. The complexity will be high with large number of bits in the string.

2.7 Implementation

The 8085 microprocessor has been used for realizing the Modified Recursive Modulo-2ⁿ and Key Rotation Technique (MRMKRT). MRMKRT is first implemented with 8-bits block size, then with 16-bits block size, then with 24-bits block size continuing up-to 256-block size. In this section generalized routine with block size 8-bit or more has been discussed. The HL-pair is loaded with memory location where the bytes will be stored, register C is stored with the value of number of iterations to be performed, and register D is stored with the value of block size. To realize the encoder, following four routines are written. The following four routines are called from the main routine during execution of the algorithm.

- Routine '**addblocks**' – This routine will add the two adjacent blocks in general.
- Routine '**rot**' – This routine will rotate the string of n bytes by one bit in left circular shift.
- Routine '**store**' – This routine will store the string as well as the intermediate strings generated.
- Routine '**subblocks**' – This routine will subtract the second block from the first block in general.

Four routines are discussed followed by diagram. These subprograms are illustrated from section 2.7.1 to 2.7.4 and main program is illustrated in section 2.7.5.

2.7.1 Algorithm of 'addblocks' routine

This routine has used HL pair as memory pointer, it will add two consecutive memory locations pointed by HL-pair, and C register as counter, representing the number of times the addition of blocks will be performed. The register D is stored with the value of the block size, this value can be set to 08h means the block size is 8-bit or with any higher value, say 10h means the block size is 16-bit and so on.

- Step 1: Clear register C and CARRY flag.
- Step 2: Load C with counter value, say 05H.
- Step 3: Load D and E with block size, say 10H.
- Step 4: Load HL pair to point to memory location F900H.
- Step 5: Move the content of memory to A register.
- Step 6: Increment HL pair as many a times the value stored in D.
- Step 7: Move the content of memory to B register.
- Step 8: Add A and B registers without CARRY.
- Step 9: Move the result stored in A to memory location pointed to by HL pair.
- Step 10: Decrement D.
- Step 11: If D = 0, then go to Step 16.
- Step 12: Load again HL pair with F900H.
- Step 13: Decrement C register.
- Step 14: Add the content of C to HL pair and store the result in HL pair.
- Step 15: Repeat from Step 5.
- Step 16: Increment HL pair by 10H.
- Step 17: If C is zero then go to Step 18 else go to Step 6.
- Step 18: Return.

By changing the value in register D the block size can be changed, register C here store the number of iteration this modulo addition can be done, the result is stored in the consecutive memory locations pointed by the HL-pair. After each addition the register C is decremented by 1H and the HL-pair is incremented by block size, in this routine HL-pair is incremented by 10H. The whole operation is repeated until register C becomes zero.

2.7.2 Algorithm of 'rot' routine

This routine rotates the string anticlockwise by one bit, containing n bytes. It is assumed that the string is stored from F900H onwards, LSB in F900H. The ls bit of the string stored in F900H is checked for 0 or 1 and set the carry accordingly. Then, the memory pointer is set to the last location, containing the MSB, and rotates the byte left by one bit through carry. The process is continued till the first location, containing the LSB, is reached.

- Step 1: Register C is initialized as counter
- Step 2: HL pair, used as memory pointer, is set to F900h
- Step 3: The memory content is moved to A.
- Step 4: 01h is ANDed with A.
- Step 5: If the zero flag is set, register B is loaded with 00h, otherwise with 01h.
- Step 6: The memory pointer is set to the last location.
- Step 7: The 1s bit in B is shifted to carry bit.
- Step 8: The memory content is rotated through carry.
- Step 9: The pointer is decremented.
- Step 10: The byte counter, C is decremented.
- Step 11: Till the counter is exhausted, go to step 8.
- Step 12: Returned.

By changing the count value in C, the bit length in string can be changed, that is the number of bits to be left circular rotated.

2.7.3 Algorithm of 'store' routine

This routine is used for storing the string as well as the intermediate string generated from F900h onwards during encoding or decoding. Here the HL pair is used the pointer of the memory from where the bytes will stored. The initialization of the HL pair is made through the main program and will be used as parameter to the routine 'store'.

- Step 1: The BC and DE pair is saved in the stack.
- Step 2: The D is initialized with byte counter.
- Step 3: The BC pair is initialized with F900H.
- Step 4: The content of memory pointed by BC pair is moved to A.
- Step 5: The content of A is moved to the memory pointed by HL pair.
- Step 6: The HL and BC pairs are incremented.
- Step 7: The D, byte counter is decremented.
- Step 8: Till it is zero, go to step 4.
- Step 9: BC and DE pairs are incremented.
- Step 10: Returned.

By changing the counter value in D, the byte length can be changed. Thus this routine stores the intermediate results on memory location F900H onwards.

2.7.4 Algorithm of ‘subblocks’ routine

This routine has used HL pair as memory pointer, it will subtract second block from the first block of two consecutive memory locations pointed by HL-pair, and C register as counter, representing the number of times the subtraction of blocks will be performed. The register D is stored with the value of the block size, this value can be set to 08h means the block size is 8-bit or with any higher value, say 10h means the block size is 16-bit and so on.

- Step 1: Clear register C and CARRY flag.
- Step 2: Load C with counter value, say 05H.
- Step 3: Load D and E with block size, say 10H.
- Step 4: Load HL pair to point to memory location F900H.
- Step 5: Move the content of memory to A register.
- Step 6: Increment HL pair as many a times the value stored in D.
- Step 7: Move the content of memory to B register.
- Step 8: Subtract A from B registers.
- Step 9: Move the result stored in A to memory location pointed to by HL pair.
- Step 10: Decrement D.
- Step 11: If D = 0, then go to Step 16.
- Step 12: Load again HL pair with F900H.
- Step 13: Decrement C register.
- Step 14: Add the content of C to HL pair and store the result in HL pair.
- Step 15: Repeat from Step 5.
- Step 16: Increment HL pair by 10H.
- Step 17: If C is zero then go to Step 18 else go to Step 6.
- Step 18: Return.

By changing the value in register D the block size can be changed, register C here store the number of iteration this modulo subtraction can be done, the result is stored in the consecutive memory locations pointed by the HL-pair. After each subtraction the register C is

decremented by 1H and the HL-pair is incremented by block size, in this routine HL-pair is incremented by 10H. The whole operation is repeated until register C becomes zero.

2.7.5 Main Program of MRMKRT

The generalized implementation is discussed here; MRMKRT is first implemented with 8-bits block size, then with 16-bits block size followed by with 24-bits block size continuing up-to 256-block size.

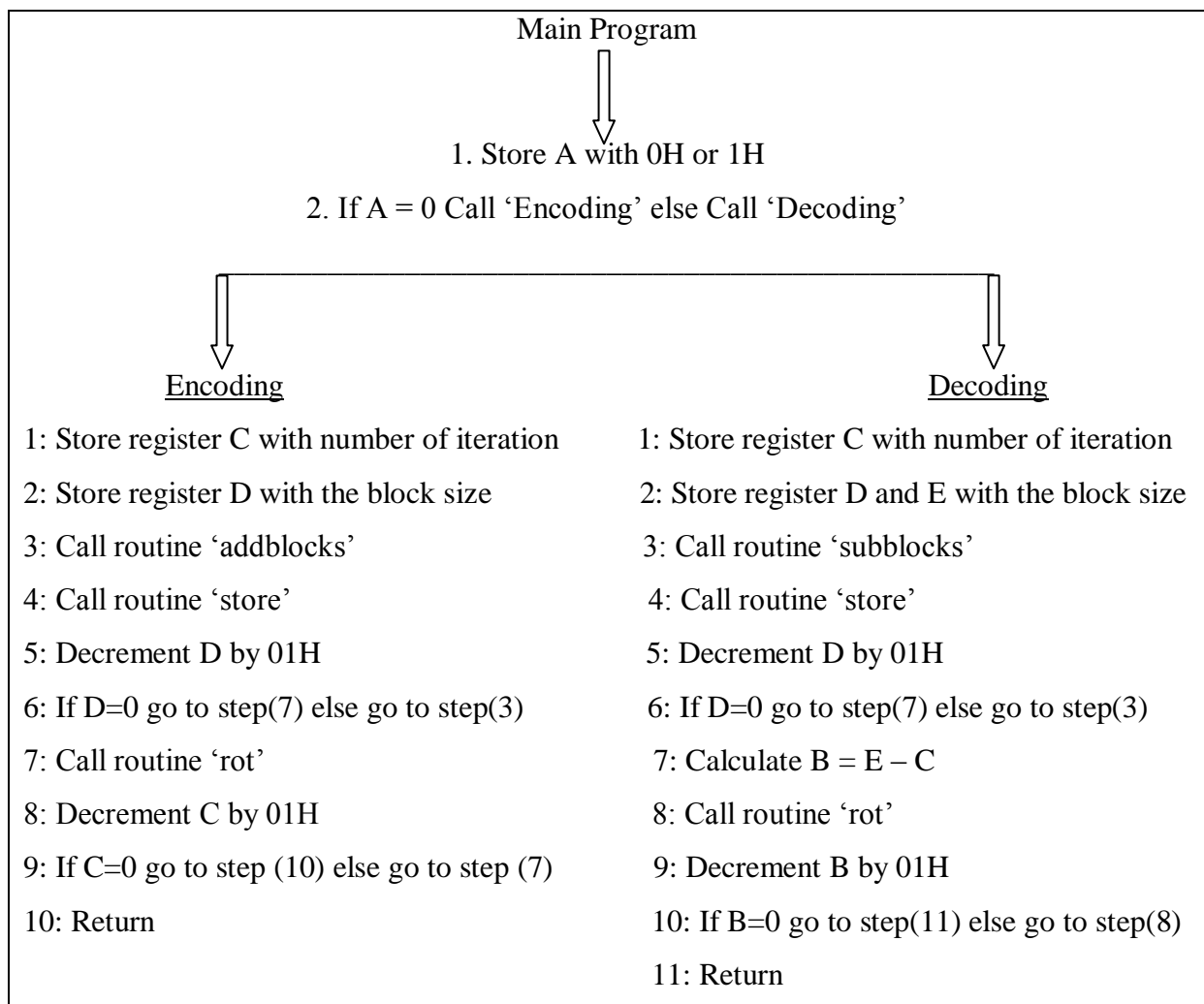


Figure 2.7: MRMKRT encryption and decryption algorithm

Figure 2.7 gives the generalized routine for MRMKRT encryption and decryption. Initially Accumulator (or any other register) is stored with the value 0H or 1H, 0H value for encryption and 1H value for decryption.

In encryption routine, register C is stored with the number of iteration to be performed and register D is stored with the block size. Then routine 'addblocks' is called to add consecutive blocks given by HL-pair and after that routine 'store' is called to store the intermediate results in stack. These processes will continue for given number of block sizes as given by register D, thereafter routine 'rot' is called to rotate the blocks by circular left rotation and the number of rotation is given by the value stored in register C.

In decryption routine, register C is stored with the number of iteration to be performed and register D is stored with the block size. Then routine 'subblocks' is called to subtract the second block from the first block of consecutive blocks given by HL-pair and after that routine 'store' is called to store the intermediate results in stack. These processes will continue for given number of block sizes as given by register D, thereafter routine 'rot' is called to rotate the blocks by circular left rotation and the number of rotation is given by the value stored in register B which is equal to E (block size) – C (number of iterations performed during encryption).

2.8 Results and Comparisons

MRMKRT is also implemented in high-level C-programming language and some of the results are extracted after encrypting some plaintext files then these are compared with existing algorithms, RSA, to prove the feasibility of MRMKRT. Finally it is encoded for microprocessor based system using 8085-assembly language programming. The acceptance of any solution must satisfy some test parameters, here MRMKRT is tested for feasibility in five dimensions, these are implementation based results, frequency distribution graph analysis, Chi-Square test for non-homogeneity, time complexity analysis taking encryption and decryption time and finally the avalanche ratio test. Section 2.8.1 discuss the implementation based results, section 2.8.2 illustrates the frequency distribution analysis, section 2.8.3 test for non-homogeneity, section 2.8.4 gives the time complexity analysis and section 2.8.5 illustrates the avalanche ratio test.

2.8.1 Implementation Based Results

MRMKRT is encoded in 8085-assembly language program, MRMKRT is encoded for 4-bit block size, then 8-bit block size continuing up-to 256-bit block size and finally a

generalized coding has been done. This section explains some of the implementation based results.

Table 2.3: Implementation based results of MRMKRT

Characteristics ↓	Proposed Techniques →	MRMKRT
Block Cipher		√
Fixed Length Block Cipher		√
Variable Length Block Cipher		-
Implementation in Bit-Level		√
Implementation other than Bit-Stream		-
Private/Symmetric Key System		√
Substitution Technique		√
Transposition Technique		-
Boolean as Basic Operation		√
Non-Boolean as Basic Operation		√
No Alteration in Size		√
Formation of Cycle		√
Non-formation of Cycle		-
Number of sub-programs used		4
Number of IO/M operations per block of encryption/decryption		9
Number of Boolean operations used per block of encryption/decryption		1
Number of Non Boolean operations used per block of encryption/decryption		5
Calculated T-states per block of encryption/decryption		760

MRMKRT is fixed length block cipher techniques and these are encoded with fixed length block size say 8-bit, 16-bit, 24-bit continuing up-to 256-bit block size and finally a generalized coding has been done. Technique is implemented in bit-level with private/symmetric key cryptography. MRMKRT is substitution cipher, MRMKRT uses both modulo addition (non Boolean) and Boolean as a basic operation. The plaintext size and ciphertext size remains same for MRMKRT. MRMKRT forms cycle where the plaintext regenerates after some finite number of iteration depends on block size and number of iteration used during encryption. MRMKRT used 4 sub-programs and MRMKRT used 9 IO/M operations per block encryption/decryption. MRMKRT used one Boolean operation per block of encryption/decryption but MRMKRT also used 5 non Boolean operations per block of encryption/decryption. So, T-states calculated for MRMKRT is 760. Thus it can be said that in microprocessor based implementation perspective MRMKRT is successfully realized. Table 2.3 summarizes these discussions.

2.8.2 Frequency Distribution Analysis

The variation of frequencies of all the 256 ASCII characters between the source file and the encrypted file are given in this section. The evenly distribution of character frequencies over the 0-255 region of the encrypted file against the source file ensures better security provided by the proposed algorithm, MRMKRT, and it also shows the heterogeneity between the two files.

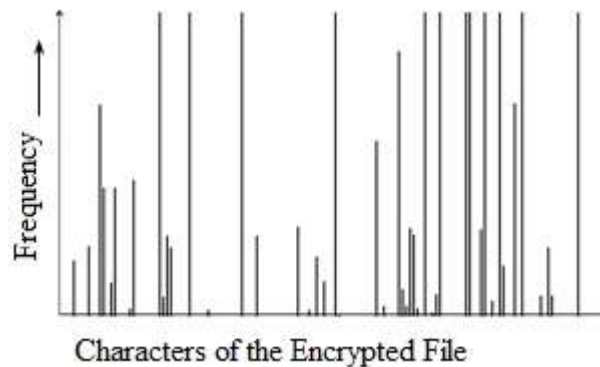


Figure 2.8: Frequency distribution of ASCII characters in the RSA encrypted file

The frequency distribution graph of RSA encrypted file is drawn in figure 2.8. According to the percentage of occurrence of a particular character, not the total number of occurrence. In the frequency distribution graph of RSA encrypted file it can be clearly seen that the frequencies are scattered in some regions and not well distributed throughout the region.

Although ten different files were encrypted and decrypted using both RSA and MRMKRT, only one such file is considered here for analyzing the results. Figure 2.9 illustrates the frequencies of occurrence of all the 256 ASCII characters in the source file and encrypted file with MRMKRT. A close observation will reveal that the characters in source file are distributed in a particular region where as in the encrypted file using MRMKRT the characters are fairly well distributed throughout the character space. Thus if comparing the same with RSA, shown in figure 2.8 and MRMKRT, shown in figure 2.9, to find that the characters of the MRMKRT encrypted file is well distributed than that of RSA encrypted file.

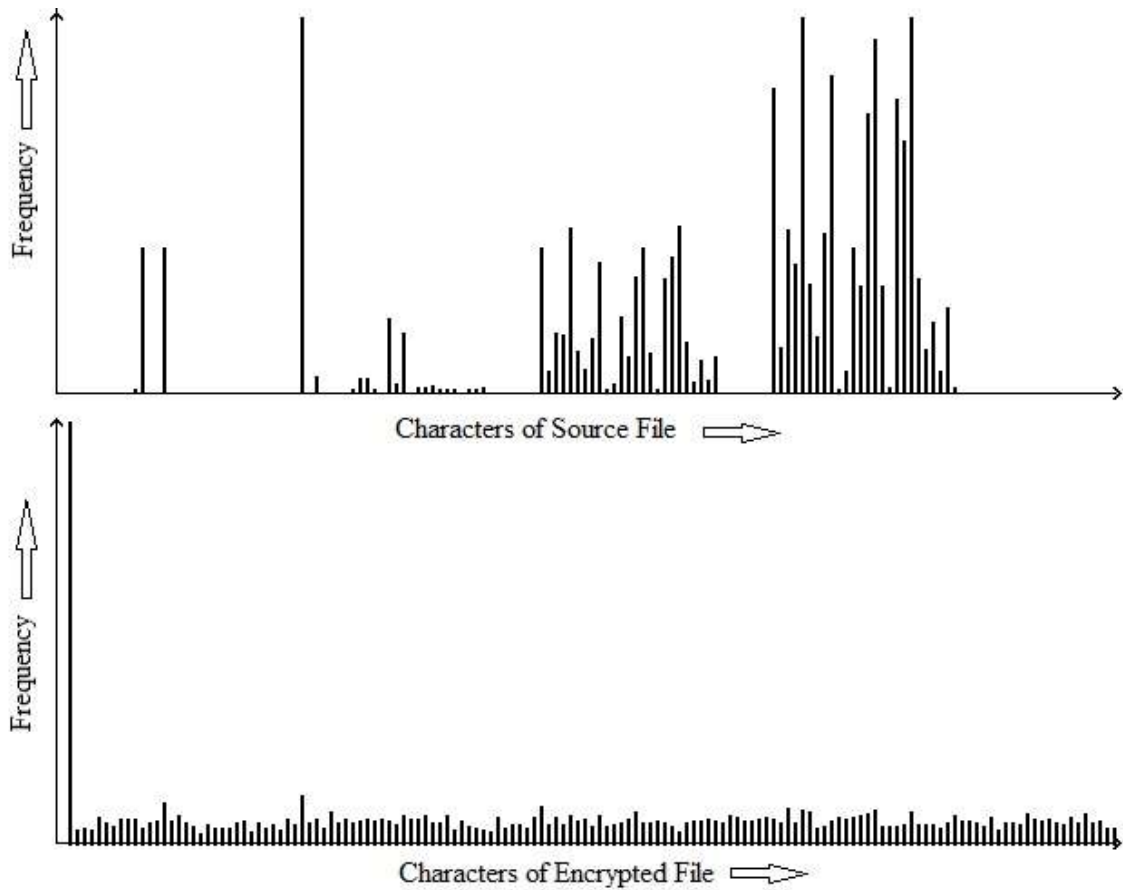


Figure 2.9: Frequency distribution of source file and MRMKRT encrypted files

Hence the MRMKRT scheme may be comparable with RSA in terms of frequency distribution graph.

2.8.3 Non-Homogeneity Test

Non-homogeneity test illustrates how far the plaintext differs from ciphertext. This test is carried out with the help of Chi-Square test. It's basically a statistical test where obtained frequency is compared with the expected frequency and thus giving the extent of non-homogeneity between obtained frequency and expected frequency. In this section the non-homogeneity between plaintext/source file and ciphertext/encrypted files is given.

Table 2.4: Chi-Square values of RSA and MRMKRT

Source File	File Size (Bytes)	Chi-Square Value		Degree of Freedom	
		MRMKRT	RSA	MRMKRT	RSA
license.txt	17,632	221484	40159	255	64
cs405(ei).doc	25,422	295480	199354	255	66
acread9.txt	35,121	420836	179524	255	73
deutsch.txt	47,829	555127	344470	255	77
genesis.txt	49,600	657591	416029	255	75
pod.exe	69,981	886397	751753	255	76
mspaint.exe	136,463	1213869	1204193	255	88
cmd.exe	152,028	1792759	585857	255	73
d3dim.dll	193,189	4351663	328677	255	10
clbcattq.dll	403,901	3823423	328511	255	11

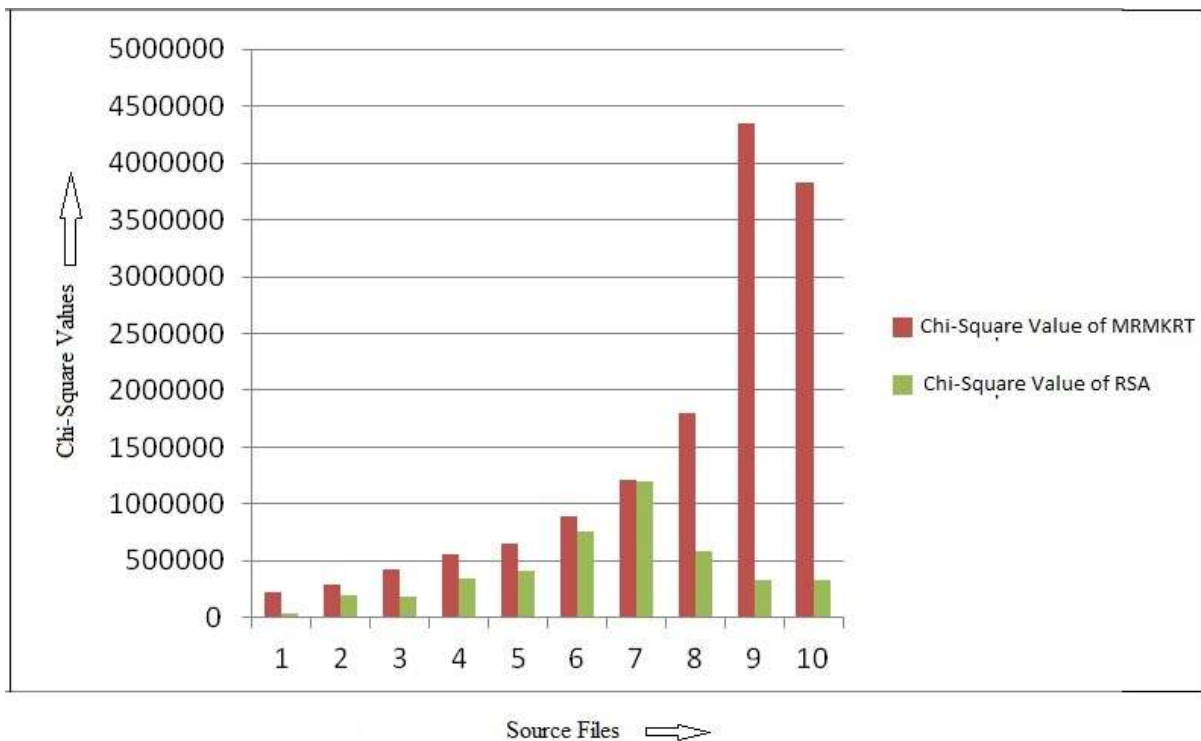


Figure 2.10: Chi-Square values for MRMKRT and RSA encrypted files

Table 2.4 and figure 2.10 show the file size and the corresponding Chi-Square values for ten different files. The Chi-Square values of the proposed algorithm, MRMKRT, are coming to be in the range of twenty thousand, thirty thousand, forty thousand, fifty thousand and so on, which are very good results indeed. It is observed that the Chi-Square values for

MRMKRT are larger compared to RSA. Further, the high values prove that Chi-Square is highly significant at 1% level of significance. Hence the source and the corresponding encrypted files are considered to be heterogeneous. Also it has been noted that the time taken to encrypt a file using MRMKRT is very small compared to that using RSA. One can decide from this observation that MRMKRT is comparable to RSA from the heterogeneity point of view.

Table 2.4 also gives values of degree of freedom; in this context the degree of freedom means the different type of characters present in the encrypted file. If observing this table the degree of freedom of MRMKRT encrypted files are coming to be 255 and that of RSA encrypted file is quite less. It means that all the ASCII characters are present in MRMKRT encrypted file and this result is at par with the frequency distribution graph, where it is also seen that frequency of MRMKRT encrypted file is well distributed.

2.8.4 Time Complexity Analysis

Time complexity analysis is another vital algorithmic parameter, time complexity is basically is the amount of time required for an algorithm to complete. The time complexity analysis is basically done by two ways, first one is a priori estimates and second one is a posteriori estimates. Second one is taken for time complexity analysis of MRMKRT. This section shows the time complexity analysis by taking encryption time and decryption time.

Table 2.5: The time complexity analysis of MRMKRT and RSA

Source File	File Size (Bytes)	Encryption time (in Seconds)		Decryption time (in seconds)	
		MRMKRT	RSA	MRMKRT	RSA
license.txt	17,632	0.01	0.01	0.12	0.28
cs405(ei).doc	25,422	0.01	0.03	0.13	0.30
acread9.txt	35,121	0.15	0.21	0.15	1.67
deutsch.txt	47,829	0.18	0.35	0.18	3.51
genesis.txt	49,600	0.23	0.40	0.20	5.06
pod.exe	69,981	0.39	0.39	0.33	4.34
mspaint.exe	136,463	0.40	0.65	0.43	8.37
cmd.exe	152,028	0.44	0.61	0.51	6.59
d3dim.dll	193,189	0.57	0.75	0.52	10.15
clbcattq.dll	403,901	0.60	0.95	0.55	11.70

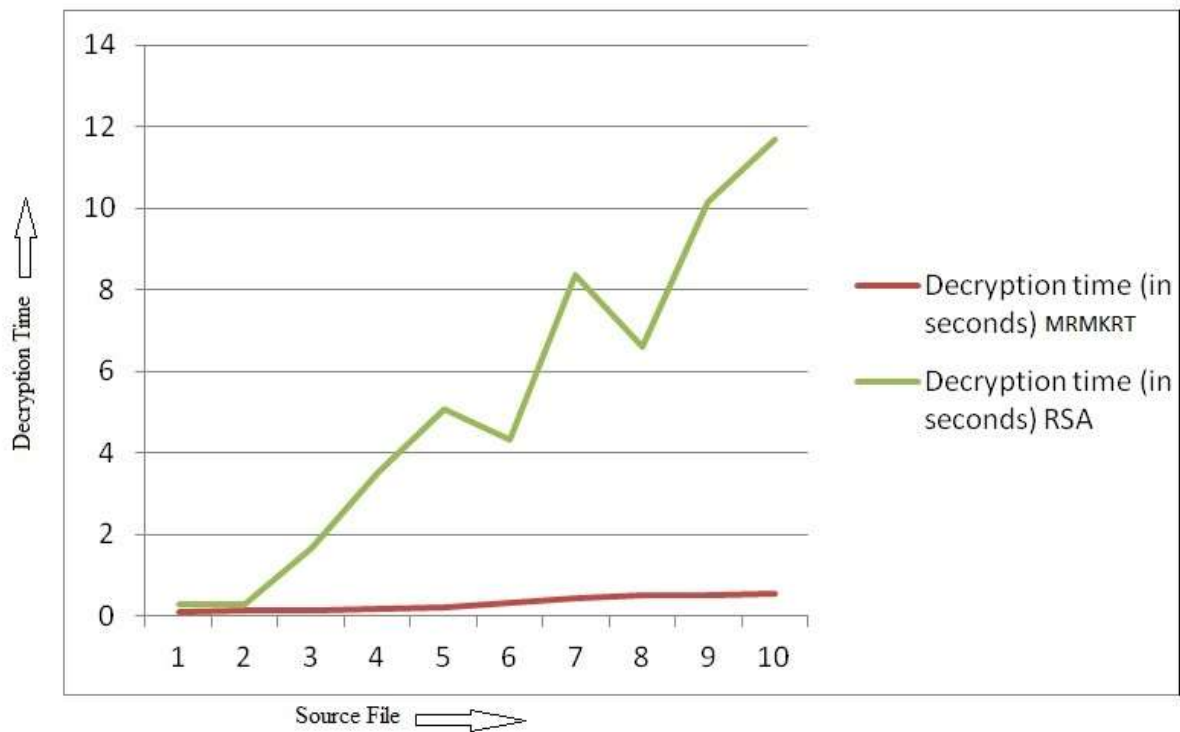
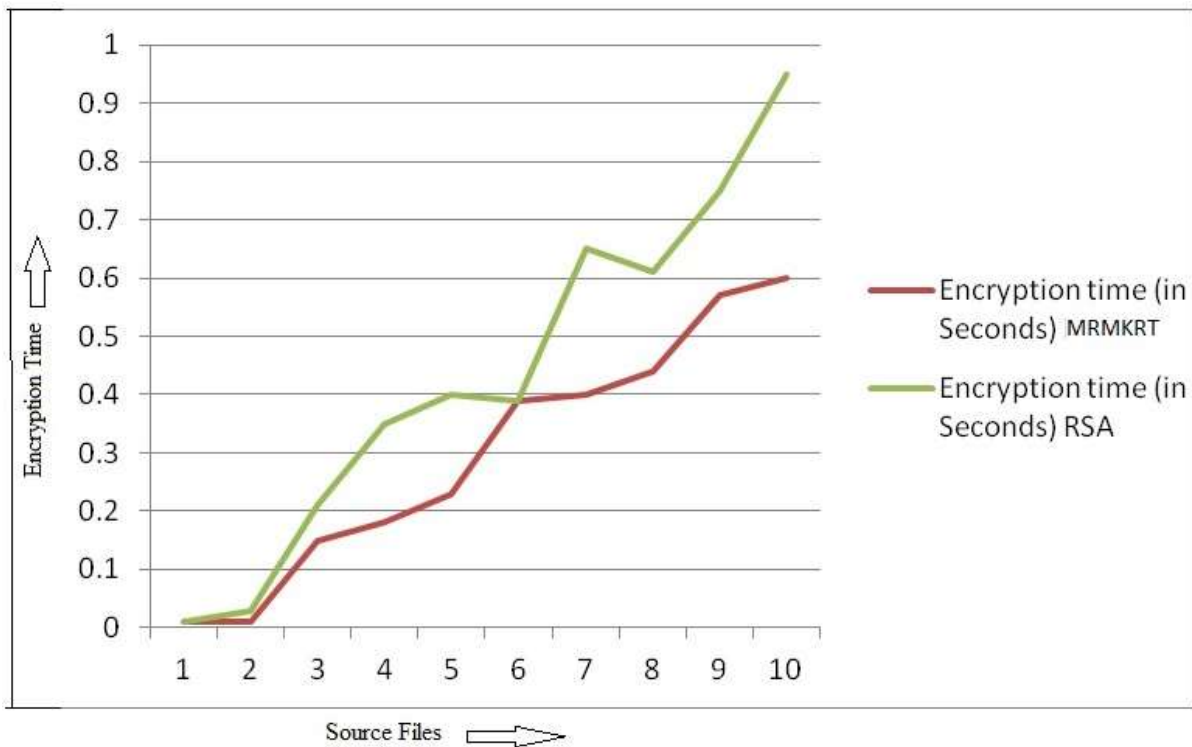


Figure 2.11: Encryption and decryption time of MRMKRT and RSA

Table 2.5 shows the encryption time and decryption time of the proposed technique and that of RSA. Figure 2.11 represent the same graphically. The time complexity analysis is one of the important factors in algorithm design. Here both encryption time and decryption time is tabulated and shown in the figure. The green line shows the time complexity of RSA

and pink line gives the time complexity of this proposed technique, MRMKRT. If observing the encryption time, MRMKRT time of encryption is marginally lower than that of RSA, and observing the decryption time than it is seen that MRMKRT time of decryption is quite less than that of RSA. The cumulative encryption time of MRMKRT is 2.98 seconds and RSA is 4.35 seconds. The cumulative decryption time of MRMKRT is 3.12 seconds and RSA is 51.97 seconds. Hence it can be concluded that the time complexity of the proposed technique, MRMKRT, is quite less than that of RSA.

2.8.5 The Avalanche Test

The Avalanche ratio is another important parameter for the cryptographic security. The Avalanche is the ratio of difference between the simple encrypted file and one bit modified source/key file. The avalanche ratio is the degree of measure for cryptanalysis. In general terms it is the measure that in what extent the characters/bits in the encrypted file will differ if to modify some characters/bits in the source file or in the session key.

Table 2.6: Avalanche ratio values of MRMKRT and RSA

Source File	File Size (Bytes)	Avalanche Ratio (in Percentage)	
		RSA	MRMKRT
license.txt	17,632	58.0	77.7
cs405(ei).doc	25,422	60.0	80.0
acread9.txt	35,121	75.0	88.8
deutsch.txt	47,829	78.9	89.0
genesis.txt	49,600	80.9	87.0
pod.exe	69,981	58.0	77.0
mspaint.exe	136,463	58.9	76.0
cmd.exe	152,028	67.0	77.0
d3dim.dll	193,189	67.9	82.9
clbcattq.dll	403,901	68.0	88.5

Table 2.6 illustrates the result of avalanche ratio of the proposed technique, MRMKRT. During this test some characters/bits in the source file are modified and then again these modified source files are encrypted. Then the percentage of the difference

between the original encrypted files and the modified encrypted files are taken. It is observed from table 2.5 that the avalanche ratio of the proposed technique is nearly 80% and that of RSA is 65%, hence in terms of avalanche ratio analysis MRMKRT is quite comparable with RSA.

2.9 Discussions

The technique proposed takes little time to encode and decode though the block length is high. The encoded string will not generate any overhead bits. The block length may further increased beyond 256 bits, which may enhance the security. Selecting the block pairs in random order, rather than taking in consecutive order may enhance security. The proposed scheme may be applicable to embedded systems. Since it is giving very good results for text files so this proposed technique can be applicable in text based messaging, to encrypt and decrypt the text messages. The main advantages of this proposed technique are its heterogeneity and even frequency distribution. The main disadvantage is since it substitutes the second block and the first block remains unaltered, so this leads to the weakness of these techniques which are going to overcome in the next proposed technique.

Chapter 3
Recursive Transposition Technique (RTT)

3.1 Introduction

This is another method of Encoding as described in earlier chapters. It is also a symmetric and block cipher type in connection with the encryption. MRMKRT described in the previous chapter is a substitution type cipher where the modulo addition of two consecutive blocks replaces the second block, RTT described in this chapter is a permutation and substitution type cipher where permutation of plaintext bits are performed first and then XOR operation is performed between two consecutive matrices and the result replaces the second matrix. In MRMKRT modulo addition is the main component of this technique whereas in RTT XOR is the main component of this technique. Considering that a k-bit string is passed through the RTT encoder, which encodes a string of same length at its output as shown in figure 3.1.

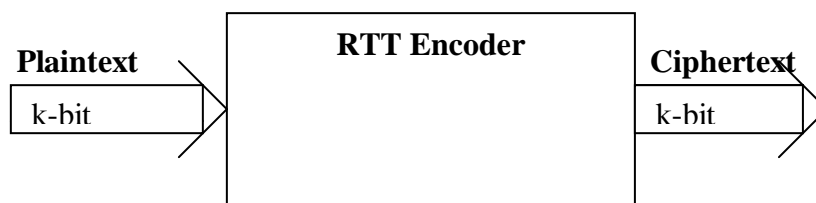


Figure 3.1: RTT encoder

Let X be the string of k-bit. It is supplied as an input to the RTT Encoder. The encoder will generate a string X' of k-bit at the output. This is the first cycle of encoding. If the generated string is allowed to pass to the input of the encoder again, then the encoder will again generate a string X'' . This is called the 2nd cycle and so on.

The process is repeated and checked each time at the output, whether the output is identical with the string supplied initially (i.e. X) or not. It is assumed that the original string is generated after i cycles. Then the intermediately generated one of ($i-1$) strings can be used as encoded string.

Let consider that after m ($m < i$) cycles the generated string is used as encoded string. The original string X can be decoded by applying ($i-m$) cycles on the encoded string.

This encoder has been tested and verified with the help of a microprocessor based system, the specification of which is given the previous chapter, with a string of 256 bit maximum. The length of string, as recommended presently, is sufficiently high for decryption.

Section 3.2 describe the algorithm of RTT, section 3.3 explain key generation process, section 3.4 performs an analysis, section 3.5 give the implementation details, section 3.6 illustrates results and comparisons and section 3.7 gives a short discussions and chalk out the future work (the next part of this thesis with FPGA-based solutions).

3.2 The Algorithm of RTT

The plaintext is first broken down into blocks of bits then RTT encryption is performed, same is done during RTT decryption. The number of iterations in encryption and decryption is also same. A generalized approach has been consider to describe RTT.

RTT is proposed here which shows comparable result in terms of Non-Homogeneity test, time complexity analysis and avalanche ratio test than that of RSA, and MRMKRT. RTT is also a bit level symmetric key cryptography.

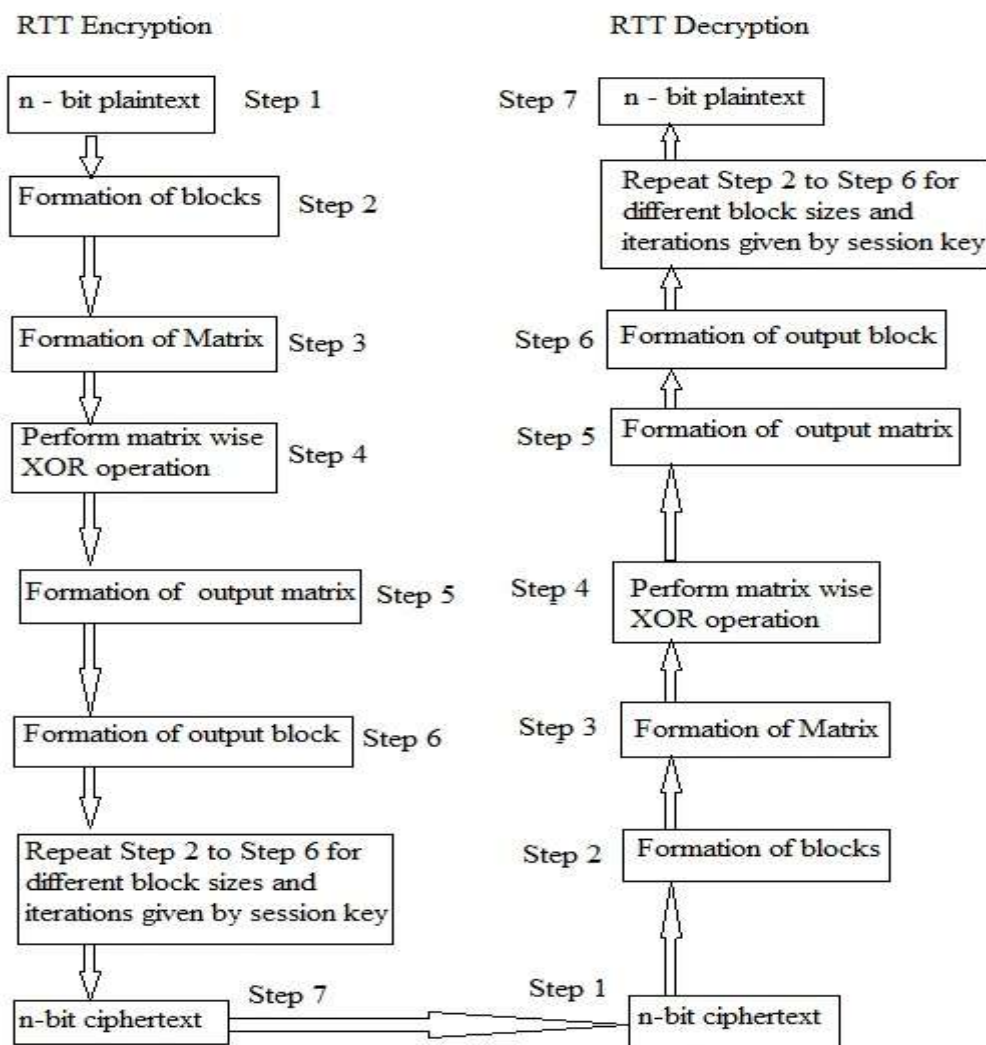


Figure 3.2: Block diagram of Recursive Transposition Technique (RTT)

Encryption and decryption is broadly divided into seven steps, firstly n-bit source stream is formed into blocks, then these blocks are formed into matrix, XOR operation is performed into two consecutive matrices, result in formation of output matrix, then these matrix are again formed into blocks, these steps are repeated for various block sizes and iteration given by session key, finally blocks are merged to form output stream.

Figure 3.2 shows the block diagram of RTT. Section 3.2.1 describes the encryption process in details, section 3.2.2 describes the decryption process in details, and section 3.2.3 illustrates an example.

3.2.1 The Encryption Process

The block diagram of RTT is shown in figure 3.2. The input stream is rounded into blocks of n- bits each the n may be even or it may be odd and pairing the blocks as explained in section 3.1, the following operations are performed starting from the most significant side. This is a recursive type algorithm/technique.

Initially, the whole plain text is considered as a stream of bits and it is broken down into a finite number of k blocks. As it is generalized approach so block size of $k = 2 * n$ or $(2*n + 1)$ where $n = \{1,2,3,\dots\}$ is a set of positive integer. The block size varies between even and or odd numbers of bits. As shown in figure 3.2, let n-bit plaintext form two blocks be $[a_1, a_2, a_3, \dots, a_9]$ and $[a_{10}, a_{11}, a_{12}, \dots, a_{18}]$, here it can be seen that the block size is 9-bit which odd number of bits is. The block length may be odd or even bits which is the strength of this technique.

Now, the blocks are formed into a matrix of $n * m$ size where 'n' is the number of rows and 'm' is the number of columns respectively of the matrix. As shown in the figure the two nine bit blocks forms two 3 X 3 matrices.

Then, two matrices are XORED, $b_1 = a_1 \text{ XOR } a_{10}$, $b_2 = a_2 \text{ XOR } a_{11}$, $b_3 = a_3 \text{ XOR } a_{12}$, $\dots, b_9 = a_9 \text{ XOR } a_{18}$.

Next, the resultant matrix replaces the second matrix remaining the first matrix as it is. So, as shown in the block diagram of RTT, now get the two matrices $[a_1, a_2, a_3, \dots, a_9]$ and $[b_1, b_2, b_3, \dots, b_9]$. The two matrices are noted down in row major order to get the cipher text. Here to get the n-bit ciphertext as $[a_1, a_2, a_3, \dots, a_9, b_1, b_2, b_3, \dots, b_9]$.

After that, the whole operation is performed on k numbers of blocks that is 0th block to (k-1) the block.

Lastly, the whole operation is performed for various block sizes, matrix sizes and number of iteration. The block sizes, matrix sizes and iterations will form a part of the session key as discussed in section 3.3. The different values of block size, matrix size and iteration number will give total different ciphertext output for same plaintext. This is another strength of this technique, which is the flexibility described in section 3.4.

3.2.2 The Decryption Process

The technique is symmetric in nature so the decryption is done in similar manner. The decryption is nothing but the iteration of the same encryption process until the source stream is got. The number of iteration requires for the decryption depends upon the block size, matrix size and the number of iterations performed during encryption.

Firstly, n-bit ciphertext is again broken down into two blocks, [a1,a2,a3,a9] and [a10,a11,a12,a18].

Now, this two block is now formed into two 3 X 3 matrices and XOR operation is performed. So, get b1 = a1 XOR a10, b2 = a2 XOR a11, b3 = a3 XOR a12,b9 = a9 XOR a18.

Then the result replaces second matrices, last get the n-bit plaintext as [a1,a2,a3,a9,b1,b2,b3,b9].

3.2.3 Example

RTT is also a variable length block-cipher, that is block size is not restricted to 2^n , where $n = \{\text{Set of positive integers}\}$. Here block sizes are of odd number of bits. This property gives the programmer the flexibility to encode RTT technique in many different ways/solutions. This is explained elaborately in section 3.4.

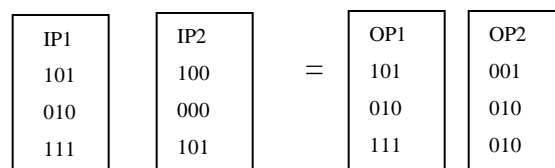


Figure 3.3: Algorithmic flow in RTT

As shown in figure 3.3 let consider a source stream of $S = 101010111100000101$. This source stream is formed into two 3×3 matrix. In the figure IP1 and IP2 are the two input matrices. Now, bit wise XOR is performed between matrix IP1 and matrix IP2. Here considering only the first row of the two input matrices, $a_1 = 1, a_2 = 0, a_3 = 1$ in IP1 and $a_{10} = 1, a_{11} = 0, a_{12} = 0$ in IP2. The exclusive operation will result in, $b_1 = 1 \text{ XOR } 1 = 0, b_2 = 0 \text{ XOR } 0 = 0, b_3 = 1 \text{ XOR } 0 = 1$. So, the output bits are 0, 0, 1 which is the first row of the output matrix OP2, OP1 is same as IP1. Similar operation will be performed for the second and third input matrices. As given in figure 3.3 after the encryption process the resultant cipher text is obtained as $S' = 101010111001010010$. Following the same steps during decryption then plaintext is regenerated.

3.3 Key Generation Process

In the proposed RTT, the key generation process is given for both fixed block size and also for variable block size. The fixed block size are those where the plaintext and ciphertext are grouped into 2^n block sizes, where 'n' is the set of positive integers and the variable block size are those where the plaintext and ciphertext are grouped into odd number of bits block sizes.

Table 3.1: Number of iteration against block sizes

Round	Block Size	Number of Iterations	
		Decimal	Binary
8.	256	50021	1100001101100101
7.	128	49870	1100001011001110
6.	64	48950	1011111100110110
5.	32	44443	1010110110011011
4.	16	46250	1011010010101010
3.	8	4321	0001000011100001
2.	4	690	0000001010110010
1.	2	72	0000000001001000
Tag field			0

In the key generation process of fixed block size eight rounds have been considered, each for 2, 4, 8, 16, 32, 64, 128, and 256-block sizes. As given in table 3.1, each round is

repeated for a finite number of times, for example, for block size of 8-bits (round 3) the iteration is for 4321 times, so, the number of iterations will form a part of the encryption-key. Although the key may be formed in many ways, for the sake of brevity it is proposed to represent the number of iterations in each round by a 16-bit binary string. Since there are eight rounds so, the binary strings are then concatenated to form a $16 \times 8 = 128$ -bit key for a particular session. The tag field is also a part of key as given in table 3.1, tag value 0 means RTT encryption and 1 means RTT decryption, so, the tag field is concatenated at LSB to get the key of 129-bit length.

Table 3.2 gives the key generation process for variable block size operation. Here, if to see the round 4, so, the block size here is of 61-bits and the number of iteration for this block size is 38 times. Hence, the block size here is also a part of session key since its value is variable.

In this process the block size is taken as 8-bit value and iteration is also an 8-bit value per round. Therefore for each round total bits is $8 + 8 = 16$ bits. There are eight rounds so total size is $16 \times 8 = 128$ -bits.

Table 3.2: Key generation for variable block length technique, RTT

Round	Block Size		Number of Iterations	
	Decimal	Binary	Decimal	Binary
8.	253	11111101	203	11001011
7.	103	01100111	101	01100101
6.	99	01100011	83	01010011
5.	70	01000110	55	00110111
4.	61	00111101	38	00100110
3.	33	00100001	20	00010100
2.	17	00010001	10	00001010
1.	3	00000011	2	00000010
Tag field				1

Adding the tag field get total session key length as 129-bits. So, in either or both cases the key bit length is 128 bits + 1 tag bit = 129 bits.

3.4 Analysis

This technique is very much flexible and has a generalized approach. As this technique uses the concept of matrix so a little alteration in the algorithm produces a larger avalanche. The alteration comes up with the following specifications:-

- The block sizes can be changed to get a different solution. In the example given in section 3.2.3, the block size is 9-bits, if to change the block size to 16-bits then there will be different solution for the same plaintext.
- Matrix size can also be altered to get a different solution, for example the 16-bit block size can get 4 X 4 matrix and 8 X 2 matrix sizes.
- The 1st matrix and or 2nd matrix can be transposed to get a different cryptographic solution.
- The orientation among the rows and or columns of the either and or both matrix also leads to another cryptographic solution. Such as after formation of two input matrices, the first row or column of the first input matrix is swapped with third row or column. This will result another cryptographic solution for the same plaintext.

So, there are many ways of alteration possible to generate new ciphers.

Considering a k-bit string is passed through the Recursive Transposition Technique (RTT) encoder, which encodes a string of same length at its output. Let X is the string of k-bit. It is supplied as an input to the RTT Encoder. The encoder will generate a string X^1 of k-bit at the output. This is the first cycle of encoding. If the generated string is allowed to pass to the input of the encoder again, then the encoder will generate a string X^2 . This is called the 2nd cycle and so on. The process is repeated and checked each time at the output, whether the output is same as the string supplied initially (i.e. X) or not. It is assumed that the original string is generated after i cycles. Then the intermediately generated one of (i-1) strings can be used as encoded string. Consider that after m (m<i) cycles the generated string is used as encoded string. The original string X can be decoded by applying (i-m) cycles on the encoded string.

In microprocessor based implementation MRMKRT used three routines which are then called by main program of MRMKRT. Whereas RTT uses eight subroutine which is

then called by main program (as it will be seen shortly in section 3.5), so, the space and time complexity of microprocessor implementation of RTT is quite more than MRMKRT. MRMKRT there is replacement of only one block, that is, only second block was replaced keeping the first block intact. The RTT can also be implemented for the replacement of two blocks; it is one of the flexibility of RTT. In this technique the number of iteration needed for decryption is same as the number of iteration needed for encryption. The first routine will clear the memory locations for storing the counter needed for regeneration of stream, so if to consider three iterations for encryption then there will be three more iteration for decryption; hence the total value of this counter is six. Therefore, in microprocessor implementation perspective the RTT is more complex in terms of time and space than MRMKRT.

RTT consist of matrix operation for both encryption and decryption so the algorithmic complexity found to be $O(n^2)$ which is much less than MRMKRT where the algorithmic complexity is $O(n^4)$.

3.5 Implementation

RTT is a variable length block size so it is first implemented in 9-bits, then 17-bits, then 35-bits and continuing up-to 255-bits block size, in this section the generalized implantation has been discussed.

The routines are developed for realizing the RTT Encoder. The routines are generalized in nature. With proper change in parameter in the routines, these may be used for any bit stream. The algorithms are written for 255 bit string. The registers described in algorithms below are A (Accumulator), B, C, D, E, H, L, SP (Stack Pointer) and PC (Program Counter). The BC, DE and HL are used as a pair of registers.

The routines are:

- Save: This routine saves the final result.
- B: This routine saves the intermediate results.
- A: This routine form the block size.
- C: This routine form the matrix.
- Prg: This routine performs XOR operation.
- Outp: This routine forms the output matrix.
- Supply: This routine regenerates the blocks.

The subroutines are described from section 3.5.1 to section 3.5.7 and the main program routine is given in section 3.5.8.

3.5.1 Routine 'save'

This routine saves the final result. This routine saves the string stored from FA00h onwards to the save area which starts from F9B0h onwards. Here 'D' register is used as counter, HL and BC pair is the memory pointer, loop is used to store the final data byte by byte. Since it is a generalized approach, RTT has been implemented for 256 bits, so the loop will iterate for 32-times as there are 32 bytes. All the subroutines are called in main program described in section 3.5.8.

Step 1: The D register is used as counter, loaded with 20h.

Step 2: The HL pair is used as pointer pointed to F9B0h, the destination

Step 3: The BC pair is used as pointer pointed to F9B0h, the source.

Step 4: The memory content pointed by BC pointer is moved to A.

Step 5: The content of A is moved to destination.

Step 6: The HL and BC pairs are incremented

Step 7: The counter register, D is decremented, till the counter is exhausted, go to step 4

Step 8: Return

3.5.2 Routine 'b'

This routine saves the intermediate results. This routine clears the temporary result area starts from FA20h onwards for 20h bytes. HL pair is used as a memory pointer, routine first clears the temporary result area then stores the intermediate results. The register C is the counter which is loaded with 32 byte.

Step 1: The HL pair is used as memory pointer pointed to FA20h

Step 2: The register A is cleared.

Step 3: The register C, used as counter is loaded with 20h.

Step 4: The content of A is moved to memory.

Step 5: The memory pointer, HL pair is incremented.

Step 6: The counter is decremented.

Step 7: Till the counter is exhausted, go to step 4

Step 8: Return

3.5.3 Routine 'a'

This routine forms the blocks with given block size, register A is the counter, HL and BC pairs are used as memory pointer, register D is used for loop, register E is the block size.

Step 1: The register A is loaded with count value, 20h and saved in FE00h

Step 2: The HL pair used as pointer is pointed to memory location FE00h.

Step 3: The BC pair is pointed to memory location FA00h.

Step 4: The register D is cleared.

Step 5: The register E is loaded with 08h.

Step 6: The content of the memory pointed by the BC pair is moved to the register A.

Step 7: The content of A is rotated right through carry.

Step 8: Jump on no-carry to step 11.

Step 9: On carry, the content of D will move to the memory pointed by the HL pair.

Step 10: The HL pair is incremented.

Step 11: The D register is incremented.

Step 12: The E register is decremented.

Step 13: Till the register E is exhausted, go to step 7.

Step 14: Else the BC pair is incremented.

Step 15: The count value is retrieved and decremented and pushes back to the stack.

Step 16: Till the count value is exhausted, go to step 5.

Step 17: The content of L is moved to the memory location FDFH.

Step 18: Return

3.5.4 Routine 'c'

This routine forms the matrices of the input stream, HL pair is used as memory pointer, register C is the matrix size, MSB 4-bits is the row size of the matrix and the LSB 4-bits are the column size. This routine also calls routine 'prg' to perform matrix wise XOR operation.

- Step 1: The HL pair used as pointer is set to the memory location FDFFh.
- Step 2: The content of the memory location FDFFh is moved to C register.
- Step 3: Is the memory content zero?
- Step 4: If yes, return.
- Step 5: Else, the HL pointer is incremented to FE00h.
- Step 6: The routine 'prg' is called.
- Step 7: The register C is decremented.
- Step 8: Till the content of register C is exhausted, go to step 5.
- Step 9: Return.

3.5.5 Routine 'prg'

This routine performs the XOR operation of the two consecutive matrices. The register H is loaded with FBh which is the matrix location, XOR operation is performed bit by bit of the two consecutive memory locations. The row and column information is obtained from register A and C. Thus this routine performs the main function of RTT Encoding.

- Step 1: The content of the memory is moved to the register L.
- Step 2: The register H is loaded with FBh.
- Step 3: The memory content is moved to register A and C register. This 8 bit data gives the row and column information of the position of the target. The 5 most significant bits give the row and the 3 least significant bits the column information.
- Step 4: The row information is derived from the data, stored in register E and in memory FAFh.
- Step 5: The column information is derived from the data and stored in FAFh.
- Step 6: The BC pair is set to FAFh.
- Step 7: The HL pair is set to FA20h, the base address of the result area.

Step 8: The content of the memory, pointed by BC pair is moved to register A.
Step 9: If the content of A is zero, go to step 11.
Step 10: The HL pair is incremented and the A register is decremented till the content of A is zero.
Step 11: The BC pair is pointed to the memory location FAFh.
Step 12: The content of FAFh is moved to the register A.
Step 13: The register D is loaded with 00000001b.
Step 14: If the content of A is zero, go to step 20.
Step 15: Else, the content of A is moved to the register C.
Step 16: The content of register D is moved to the register A.
Step 17: The content of A is rotated left and the register C is decremented.
Step 18: Till the content of C is exhausted, go to step 17.
Step 19: The content of A is moved to D register.
Step 20: The content of A is XORed with that of the memory and the result is in A.
Step 21: Return.

3.5.6 Routine 'outp'

This routine compares the data from location F9B0h onwards with that of FA20h onwards for 20h bytes. This routine regenerates the output matrix, HL and BC pair is used for memory pointer, register D is counter, as it is a generalized implementation of 256-bits source stream so there are 32 bytes which is 20h and loaded into register D.

Step 1: The HL pair and BC pair are pointed to F9B0h and FA20h respectively.
Step 2: The register D used as counter is loaded with 20h.
Step 3: The content of memory pointed by BC pair is moved to A.
Step 4: The content of A is compared with that of the memory, pointed by the HL.
Step 5: If not zero, go to step 9.
Step 6: Else, the HL pair and BC pair are incremented.
Step 7: The register D is decremented.
Step 8: If not zero, go to step 3.

Step 9: Return.

3.5.7 Routine ‘supply’

This routine is used to supply the generated string at FA20h onwards to FA00h onwards for 20h bytes. This routine regenerates the output blocks. HL and BC pair is used as memory pointer. Register D is loaded with 20h (32 bytes) to regenerates the output blocks.

Step 1: The HL and BC pair are pointed to FA00h and FA20h respectively.

Step 2: The register D used as counter is loaded with 20h.

Step 3: The content of the memory pointed by BC pair is moved to A.

Step 4: The content of A is moved to the memory pointed by the HL pair.

Step 5: Both the BC and HL pairs are incremented.

Step 6: The content of D register is decremented.

Step 7: Till the content of D is exhausted, go to step 3.

Step 8: Return.

3.5.8 Algorithm of the Main Program for RTT Encoder

The main program for RTT Encoding calls the routines described above for the transformation. When the routines are called, the registers used for it are properly saved in the stack and at the time of leaving the routine the previous condition is restored by popping.

Step 1: The stack pointer SP is initialized at the highest address of the usable RAM.

Step 2: A register pair HL is used as pointer for iteration / cycle is initialized.

Step 3: Another pointer used for storing the result (the transformed block) is saved in memory.

Step 4: The routine ‘**save**’ is called to save the string from FA00h onwards to F9B0h onwards.

Step 5: The routine ‘**b**’ is called the temporary result area.

Step 6: The routine ‘**a**’ is called to find the positions of 1s in the string and store from FE00h onwards.

Step 7: The routine 'c' is called to check for presence of 1s in the string and calls 'prg' for the transformation consulting the table stored FB00h onwards.

Step 8: The content of the memory pointed by the pointer is incremented.

Step 9: The routine '**outp**' is called to compare the generated string with the saved string.

Step 10: The transformed block is equal to the original block, the result is displayed and go to step 12.

Step 11: Else, the generated result is supplied to the location from where the transformation will begin, by calling the routine '**supply**' and go to step 5.

Step 12: Stop and end.

The above algorithm is implemented in assembly level for 15 bit initially in order to verify the transformation. The same algorithm is extended to 255 bit block with the presently available microprocessor based kit in the laboratory. It is worth pointing that there is no limitation in increasing the length of the block, if the microprocessor based system supports with large memory. For the bit-length higher than 255 bit, the destination of the target has to be coded for more than 9 bits. The developed assembly level programs are available. Therefore the theoretically computed iterations / cycles conforms the experimental value.

3.6 Results and Comparisons

RTT is also implemented in C-programming language and some of the results are taken to compares with RSA and MRMKRT, to prove the feasibility of RTT and then finally implemented for microprocessor based solutions. Section 3.6.1 discuss implementation based issues, section 3.6.2 illustrates the frequency distribution graph, section 3.6.3 test for non-homogeneity, section 3.6.4 gives the time complexity analysis and section 3.6.5 illustrates the avalanche ratio test.

3.6.1 Implementation Based Results

MRMKRT and RTT are encoded in 8085-assembly language program, MRMKRT is encoded for 4-bit block size, then 8-bit block size continuing upto 256-bit block size and

finally a generalized coding has been done. RTT is variable length techniques and these are encoded with variable length block size say 9-bit, 15-bit, 25-bit continuing upto 255-bit block size and finally a generalized coding has been done. Techniques are implemented in bit-level with private/symmetric key cryptography. MRMKRT is substitution cipher where as RTT is substitution and transposition technique, RTT uses Boolean as basic operation and MRMKRT uses both modulo addition (non Boolean) and Boolean as a basic operation.

Table 3.3: Comparisons of MRMKRT and RTT

Characteristics ↓	Proposed Techniques →	MRMKRT	RTT
Block Cipher		√	√
Fixed Length Block Cipher		√	-
Variable Length Block Cipher		-	√
Implementation in Bit-Level		√	√
Implementation other than Bit-Stream		-	-
Private/Symmetric Key System		√	√
Substitution Technique		√	√
Transposition Technique		-	√
Boolean as Basic Operation		√	√
Non-Boolean as Basic Operation		√	-
No Alteration in Size		√	√
Formation of Cycle		√	√
Non-formation of Cycle		-	-
Number of sub-programs used		4	7
Number of IO/M operations per block of encryption/decryption		9	5
Number of Boolean operations used per block of encryption/decryption		1	1
Number of Non Boolean operations used per block of encryption/decryption		5	0
Calculated T-states per block of encryption/decryption		760	544

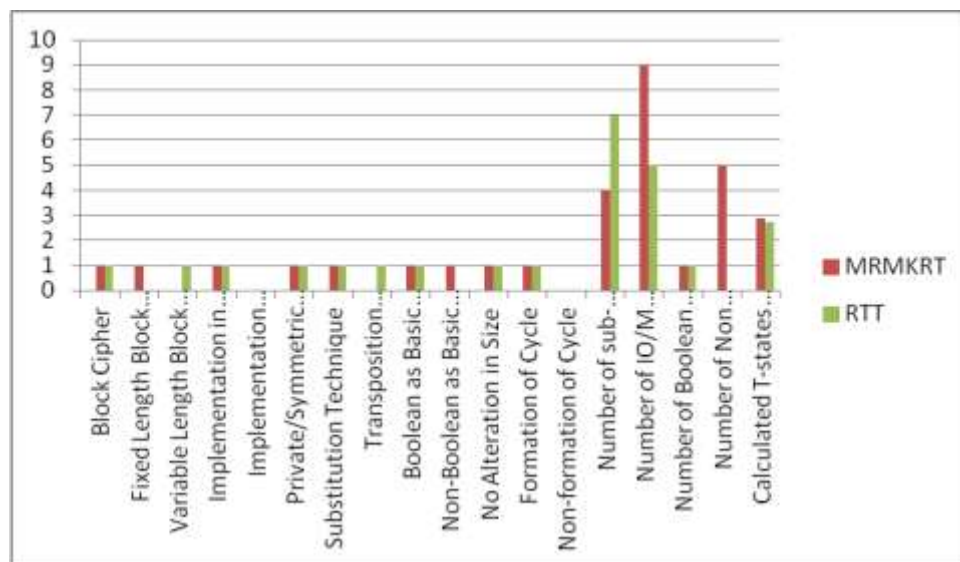


Figure 3.4: Graphical representation of comparisons of MRMKRT and RTT

The plaintext size and ciphertext size remains same for both proposed techniques. MRMKRT and RTT forms cycle where the plaintext regenerates after some finite number of iteration depends on block size and number of iteration used during encryption. MRMKRT and RTT used 4 and 7 sub-programs respectively. MRMKRT used 9 IO/M operations, and RTT used 5 IO/M operations per block encryption/decryption. MRMKRT and RTT used one Boolean operation per block of encryption/decryption but MRMKRT also used 5 non Boolean operations. So, T-states calculated for MRMKRT and RTT are 760 and 544 respectively. Thus it can be said that in microprocessor based implementation perspective RTT is the faster than MRMKRT in terms of execution speed per block of encryption/decryption. Table 3.3 and figure 3.4 summarize these discussions.

3.6.2 Frequency Distribution Graph

This section illustrates frequency distribution graph obtained after encrypting source file/plaintext file with RSA, MRMKRT and RTT. The frequency distribution graph shows the percentage of occurrences of 256-ASCII characters in both plaintext/source file and ciphertext/encrypted file. Though there are ten files encrypted with all four algorithms/techniques but here one source file and the corresponding encrypted file is taken for analysis as other nine files shows the same result. This analysis is one of the important statistical analyses for any cryptographic solutions.



Figure 3.5: The frequency distribution graph of RSA encrypted file

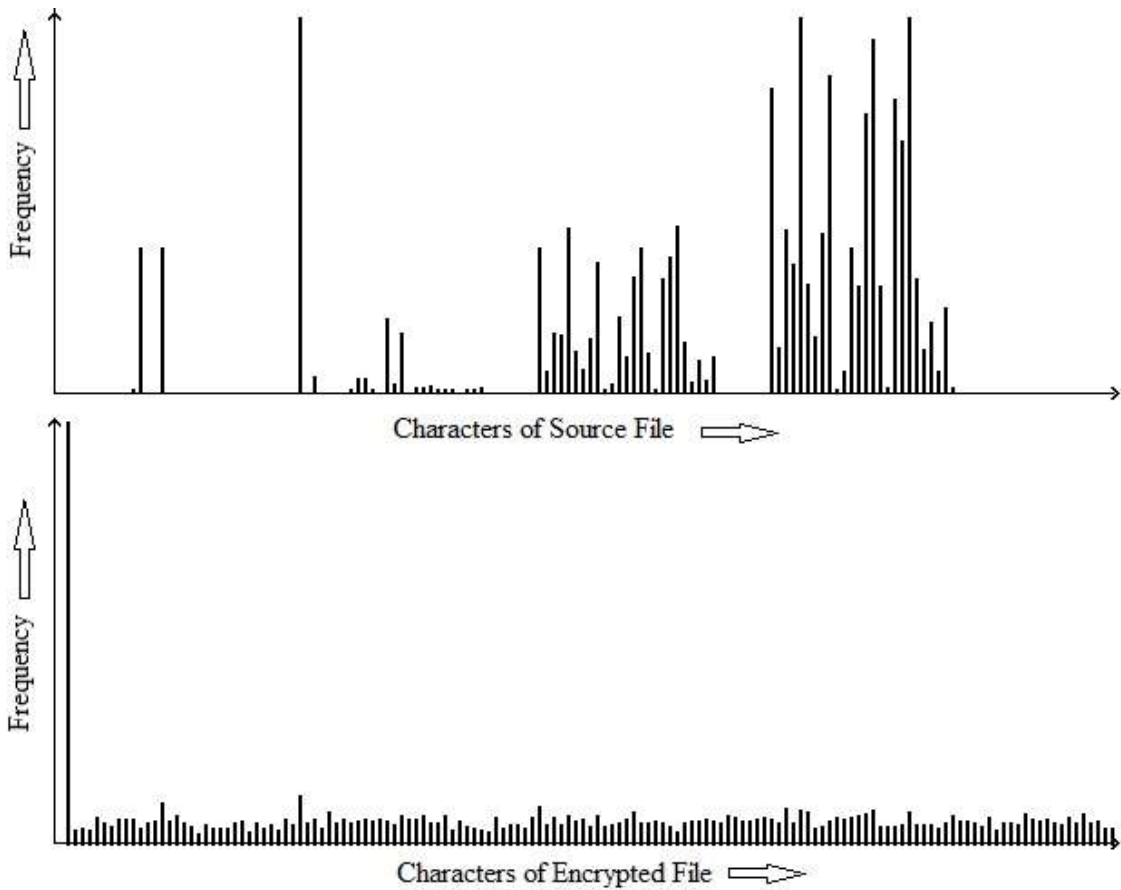


Figure 3.6: The frequency distribution graph of source file and MRMKRT encrypted file



Figure 3.7: Frequency distribution graph of RTT encrypted file

The variation of frequencies of all the 256 ASCII characters between the source file and the encrypted file are given in above figures. Over the 0-255 region of the encrypted file against the source file ensures better security provided by the proposed algorithm and it also shows the heterogeneity between the two files. These variation frequencies ensure against

brute force attack. Figure 3.5 shows the frequency distribution of RSA encrypted file. Figure 3.6 shows the frequency distribution of source file and frequency distribution of MRMKRT encrypted file. The upper half is the frequencies of source file and lower half is the frequencies of MRMKRT encrypted file. Figure 3.7 shows the frequency distribution of RTT encrypted file. The frequency distribution of this proposed technique, RTT, is well distributed. It is also observed that the frequency of MRMKRT is also well distributed. But it is evident that the frequency distribution of RSA is not well distributed. So it can be say that the proposed technique, RTT, shows a marginal improvement over RSA but this result is same for MRMKRT. Thus it can be said that RTT is well comparable with RSA and MRMKRT in terms of frequency distribution analysis.

3.6.3 Non-Homogeneity Test

Chi-Square test is carried out to perform non-homogeneity test, the observed frequency is here the plaintext file and the expected frequency is here the ciphertext file. Chi-Square test basically gives the non-homogeneity between observed frequency and expected frequency, therefore giving the non-homogeneity between plaintext and ciphertext.

Table 3.4: Chi-Square values of RSA, MRMKRT and RTT

Source File	File Size (Bytes)	Chi-Square Value			Degree of Freedom		
		RTT	MRMKRT	RSA	RTT	MRMKRT	RSA
license.txt	17,632	240550	221484	40159	255	255	64
cs405(ei).doc	25,422	270080	295480	199354	255	255	66
acread9.txt	35,121	449011	420836	179524	255	255	73
deutsch.txt	47,829	582499	555127	344470	255	255	77
genesis.txt	49,600	688115	657591	416029	255	255	75
pod.exe	69,981	916577	886397	751753	255	255	76
mspaint.exe	136,463	1340770	1213869	1204193	255	255	88
cmd.exe	152,028	1990000	1792759	585857	255	255	73
d3dim.dll	193,189	4350880	4351663	328677	255	255	10
clbcattq.dll	403,901	4425780	3823423	328511	255	255	11

The Chi-Square values are used to analyze the scheme to test the non-homogeneity of the source and the encrypted file. Table 3.4 gives the file size and the corresponding Chi-

Square values for ten different types of files. It is evident that Chi-Square values for RTT are greater compared to RSA and MRMKRT. The average Chi-Square values of RTT, MRMKRT and RSA are 1525426, 1421863 and 437853 respectively. Hence the source and the corresponding encrypted files of RTT are considered to be more heterogeneous than rest of the techniques/algorithm, RSA and MRMKRT.

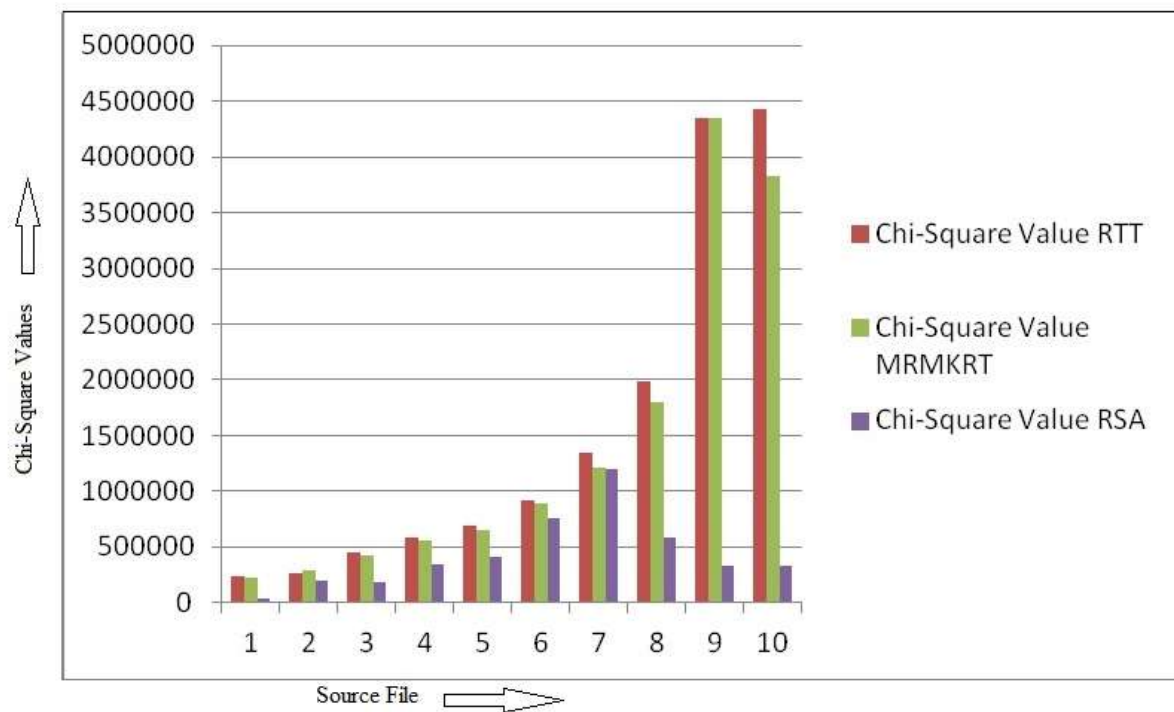


Figure 3.8: Graphical comparisons of Chi-Square values of RTT, MRMKRT and RSA

Figure 3.8 shows the Chi-Square value graph of RTT, MRMKRT and RSA encrypted files of the corresponding files. Ten files are encrypted with RTT, MRMKRT and RSA and their values are tabulated and shown in figure. It is obvious that Chi-Square value of RTT is greater than that of MRMKRT and RSA. Therefore it can be said that RTT shows more heterogeneous result than MRMKRT and RSA.

3.6.4 Time Complexity Analysis

Time complexity analysis is another important algorithmic parameter. A posteriori estimate method of time complexity analysis has been done, in this method an algorithm is encoded first and then the time of executing is noted down with test data. In this section

encryption time and decryption time is taken as parameters for performing time complexity analysis.

Table 3.5: Comparisons of time complexity analysis of RTT, MRMKRT and RSA

Source File	File Size (Bytes)	Encryption time (in Seconds)			Decryption time (in seconds)		
		RTT	MRMKRT	RSA	RTT	MRMKRT	RSA
license.txt	17,632	0.01	0.01	0.01	0.00	0.12	0.28
cs405(ei).doc	25,422	0.01	0.01	0.03	0.01	0.13	0.30
acread9.txt	35,121	0.05	0.15	0.21	0.05	0.15	1.67
deutsch.txt	47,829	0.12	0.18	0.35	0.10	0.18	3.51
genesis.txt	49,600	0.20	0.23	0.40	0.20	0.20	5.06
pod.exe	69,981	0.37	0.39	0.39	0.35	0.33	4.34
mspaint.exe	136,463	0.40	0.40	0.65	0.38	0.43	8.37
cmd.exe	152,028	0.42	0.44	0.61	0.42	0.51	6.59
d3dim.dll	193,189	0.45	0.57	0.75	0.45	0.52	10.15
clbcattq.dll	403,901	0.55	0.60	0.95	0.55	0.55	11.70

Table 3.5 illustrates time complexity data taking encryption time and decryption time. It is observed that the cumulative time of encrypting all the ten files of RTT is 2.58 seconds, MRMKRT is 2.98 seconds and RSA is 4.35 seconds. It is also observed that cumulative time of decrypting all the ten files of RTT is 2.51 seconds, MRMKRT is 3.12 seconds and RSA is 51.97 seconds. Thus in terms of time complexity analysis RTT is far better than MRMKRT and RSA.

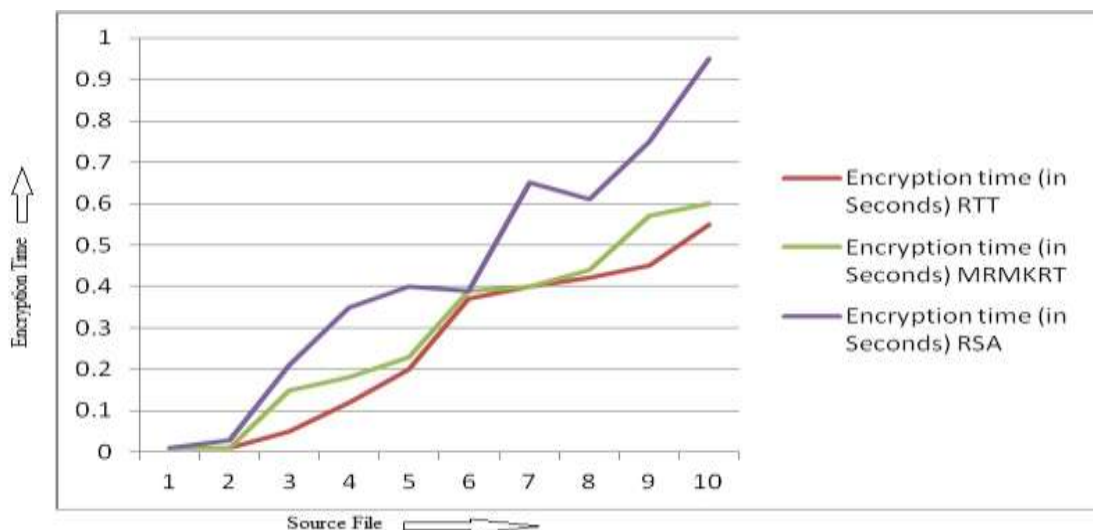


Figure 3.9: Pictorial representation of time graph of RTT, MRMKRT and RSA encryption

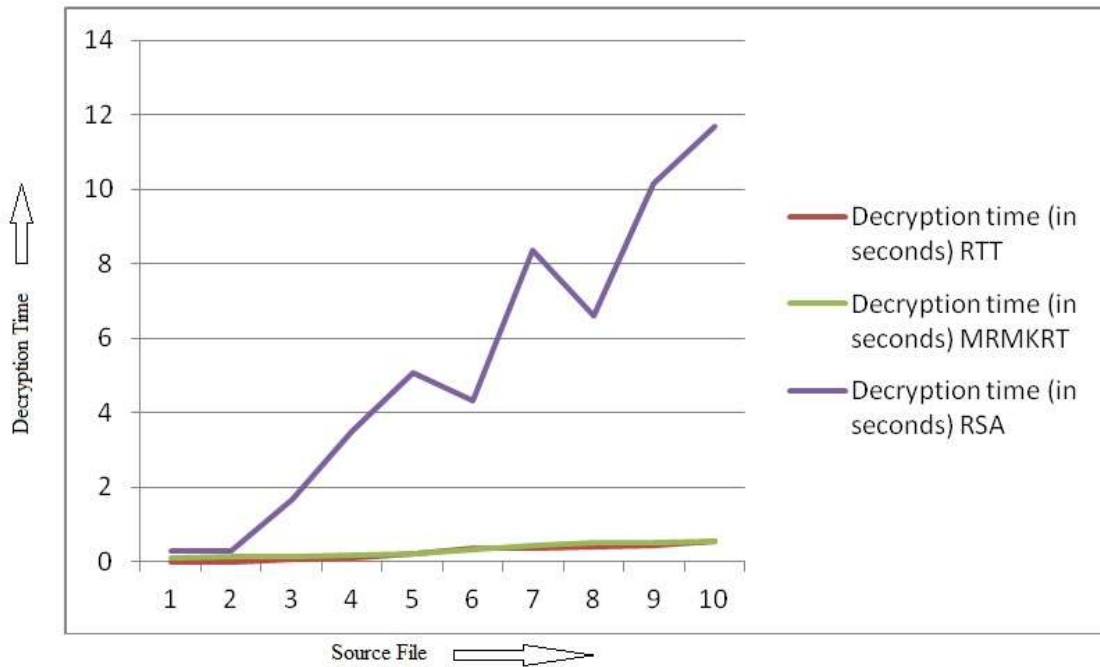


Figure 3.10: Pictorial representation of time graph of RTT, MRMKRT and RSA decryption

Figure 3.9 shows the encryption time of RTT, MRMKRT and RSA. It has been seen that encryption time complexity of RTT is quite comparable with MRMKRT and RSA. Figure 3.10 shows the decryption time of RTT, MRMKRT and RSA. The decryption time of RSA is so large that rest of the techniques almost falls in same line. But still it can be said that time complexity of RTT is quite comparable with MRMKRT and RSA.

3.6.5 The Avalanche Ratio Test

The extent of dependency between many bits of plaintext, ciphertext and key is shown with the help of Avalanche ratio test. If a single bit in plaintext or key is modified then it will alter many bits of the ciphertext. Thus avalanche ratio test is another important cryptographic parameter, this section performs this test.

Table 3.6 shows the avalanche ratio test results of RTT, MRMKRT and RSA. This test doesn't show any significant result of RTT. MRMKRT involves left circular rotation after modulo addition so avalanche ratio test is greater in MRMKRT than RTT where only one block (second block) is replaced during encryption. The avalanche ratio test of RTT is comparable with RSA.

Table 3.6: Comparisons avalanche ratio of RTT, MRMKRT and RSA

Source File	File Size (Bytes)	Avalanche Ratio(in Percentage)		
		RSA	MRMKRT	RTT
license.txt	17,632	58.0	77.7	60.0
cs405(ei).doc	25,422	60.0	80.0	65.0
acread9.txt	35,121	75.0	88.8	68.5
deutsch.txt	47,829	78.9	89.0	73.0
genesis.txt	49,600	80.9	87.0	75.5
pod.exe	69,981	58.0	77.0	80.0
mspaint.exe	136,463	58.9	76.0	81.5
cmd.exe	152,028	67.0	77.0	70.0
d3dim.dll	193,189	67.9	82.9	73.5
clbcattq.dll	403,901	68.0	88.5	65.0

3.7 Discussions

The technique proposed giving satisfactory result in heterogeneous point of view. The average Chi-Square values of RTT, MRMKRT and RSA are 1525426, 1421863 and 437853 respectively. So, it can be concluded that this proposed RTT, has the highest average Chi-Square value and is most heterogeneous. The block length may further increased beyond 256 bits, which may enhance the security. The future scope of this work is to incorporate the algorithm / Technique in embedded systems. The satisfactory results have been found after implementation and testing. So, this technique is can be used in future for achieving security in electronic devices.

The future scope of work is to propose various cryptographic solutions/techniques in FPGA based systems. As FPGA based system is a hot research topic now a day so the candidate developed some FPGA based systems. Six set of algorithms/techniques has been proposed for FPGA-Based solutions. These are given in next sections of this thesis.

Section II
FPGA Based Solutions

Chapter 4
Two Pass Replacement Technique (TPRT)

4.1 Introduction

Corporate objectives, such as increasing profits and sales revenue while utilizing research and development efficiently, are putting severe pressure on today's research and design engineering teams. The resulting system level challenges—creating new products and lowering the cost of existing “successful” products with fewer people and resources in less time—can be addressed by using a design philosophy based on FPGAs. A system architecture using FPGAs as a key component not only reduces new product-development research and development costs but also the total cost of the organization of a product's entire life cycle. It is the new low cost, power devices, the FPGA family, can reduce total system costs in addition to improving the quality of the products.

Global competition and economic factors are squeezing profits and sales of high-tech products, putting tremendous pressure on design engineer to bring to market products with lower cost. Investing in research and development in new product development presents two different system challenges: creating completely new products that take advantage of the latest technologies, features, or solutions available in the market, and developing the same for low cost. For high-tech companies in today's cost-conscious and power-sensitive “green” environment, the first challenge translates into creating a completely new product with some specific functionality not offered by anyone else, while having a lower priced entry point and/or lower power footprint. The cost reduction of existing successful products is typically handled by driving down the cost of the components from the product's bill of materials is another challenge. Another option is for design teams to redesign the product, not for new functionalities, but also to achieve more significant reduction of costs.

These goals can be achieved now with a new technological solution namely “FPGA-based system design”. Keeping views with all these section II of thesis deals with cryptographic solutions based on FPGA.

In previous section, chapter two and chapter three MRMKRT and RTT were designed and proposed respectively for microprocessor based systems. As FPGA has revolutionised the hardware design so the next six techniques are proposed based on FPGA systems.

Section 4.2 discussed the algorithm of TPRT with a block level diagram, section 4.3 gives a detailed example of encryption and decryption process, section 4.4 discussed the implementation issues with key generation, section 4.5 gives a brief analysis, section 4.6 discussed the results obtained based on implementation and discussions are given in section 4.7.

4.2 The Algorithm of TPRT

The proposed technique is a type of replacement technique or substitution technique. A substitution cipher is one in which each symbol of the plaintext is exchanged for another symbol. If this is done uniformly this is called a mono-alphabetic cipher or simple substitution cipher. If different substitutions are made depending on where in the plaintext the symbol occurs, this is called a poly-alphabetic substitution. This proposed technique is a poly-alphabetic cipher.

The original message is considered as a stream of bits, which is then divided into a number of blocks, each containing k (variable number of bits) bits. As it is a generalized approach so, $k = 2 * n$ or $k = (2*n+1)$ that is even or odd numbers of bits per block, where $n = \{\text{set of positive integers}\}$. The technique is implemented in both FPGA-based system and in high level programming language. The two adjacent blocks of a given size are XORed, the result replaces the second block, and the first block remains unchanged. In next iteration the two adjacent blocks are again XORed, now result replaces the first block, and the second block remains unchanged. Then writing the adjacent two blocks gives the target stream. The same process is repeated in whole message using a variable size of stream. The round is repeated for a finite number of times and the intermediate stream is considered as an encrypted stream.

The technique is symmetric in nature so the decryption is done in similar manner. After decomposing the encrypted stream into number of blocks, the two adjacent blocks are XORed, the result replaces the first block, and the second block remains unchanged. In next iteration the two adjacent blocks are again XORed, now result replaces the second block, and the first block remains unchanged. The decryption is nothing but the iteration until the source stream is got. The number of iteration requires for the decryption depends upon the block size and the number of iterations performed during encryption. The flow is during encryption in first iteration second block is changed with the XORed result and in the second iteration first block is changed with the XORed result. During decryption in first iteration first block is changed with the XORed result and in the second iteration second block is changed with the XORed result.

TPRT is a bit-level symmetric key block cipher. The number of iterations required for TPRT decryption is same as the number of iterations used in TPRT encryption. A generalized approach is taken for explaining the algorithm of TPRT.

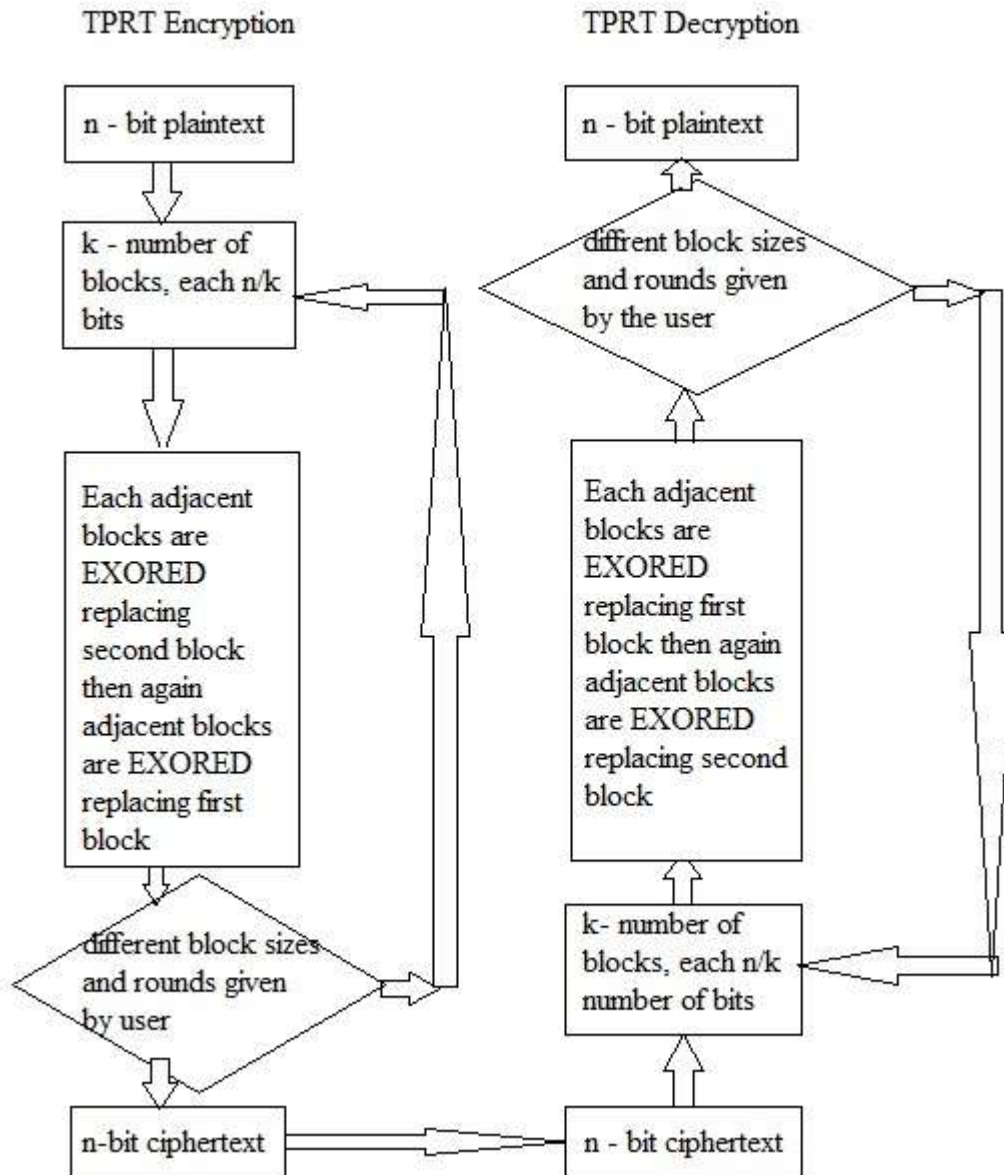


Figure 4.1: Block diagram of Two Pass Replacement Technique (TPRT)

4.2.1 The Encryption Process

The whole message is considered as a stream of n-bits and it is broken down into a finite number of k blocks, so the size of each block, $m = n/k$ bits. As it is generalized approach so block size of $m = 2 * i$ or $(2*i + 1)$ where $i = \{1,2,3,\dots\}$ is a set of positive integer. The block size varies between even and or odd numbers of bits. The block diagram of the encryption and decryption process of TPRT is given in figure 4.1.

Two successive blocks of a given length are XORED to get the result. The result replaces the second block remaining the first block intact. The whole operation is performed

on 'm' numbers of blocks that is 0th block to (m-1) the block. As discussed above is repeated in reversible manner that is the result of the XORED operation between the first block and second block is replaces the first block, keeping the second block intact.

Varying the block sizes performs the whole operation again, let the initial TPRT operation is for block size of 2-bits, then the next TPRT operation could be for block size of 4-bits. The number of blocks must be even, to obtain this successive zeroes are added in LSB position. This round is repeated for a finite number of times and the number of iterations will form a part of the session key as discussed in section 4.4.

4.2.2 The Decryption Process

The technique is symmetric in nature so the decryption is done in similar manner. At the receiver end the n-bit ciphertext stream is broken into k-number of blocks each having, $m=n/k$ number of bits. As it is generalized approach so block size of $m = 2 * i$ or $(2*i + 1)$ where $i = \{1,2,3,\dots\}$ is a set of positive integer. So, the block size varies between even and or odd numbers of bits. The flow diagram of the encryption and decryption process of TPRT is given in figure 4.1. Now, the first block is XORED with second block and the result replaces the first block keeping second block intact, in this way the XOR operation is performed for the all k-blocks. In the next iteration the first block and second block are XORED and now replacing the second block keeping first block intact. The number of iteration requires for the decryption depends upon the block size and the number of iterations performed during encryption.

4.3 Example

Two Pass Replacement Technique (TPRT) is an approach towards e-security through a variable length block cipher based symmetric encryption technique implemented in FPGA based systems. The term 'variable length' block cipher means that TPRT is not restricted to 2^n block sizes, where $n = \{0,1,2,\dots\}$, TRRT can also have odd number bits block sizes and with this TPRT encryption and TPRT decryption can be carried out successfully.

Table 4.1: Encryption process of TPRT

Encryption Phase	Source Stream 10101100			
Pass1	10	10	11	00
Pass2	Source Stream after Pass2			
	10	00	11	11
Pass3	Source Stream after Pass3			
	10	00	00	11
Encrypted stream 10000011				

The encryption process of TPRT is illustrated in table 4.1. Let the source stream be, $S=10101100$, now in the encryption pass1 this stream is broken down into four blocks each of having 2-bits size. So, there are four blocks, '10', '10', '11' and '00' which is illustrated row one of table. Then in encryption process, first block is XORED with second block and the result is replacing the second block keeping the first block intact, similarly the third block and fourth blocks are XORED and the replacing the fourth block keeping third block intact. Now to get the blocks, '10', '00', '11' and '11', which is depicted in row two of table. In the encryption pass3 the same operation is performed but here the result of XOR between first block and second block replaces the first block keeping second block intact, similarly the result of XOR between the third block and fourth block replaces the third block keeping fourth block intact. Now the blocks, '10', '00', '00' and '11' are generated, which is given in row three of table. Concatenating blocks gives the target stream, finally depicted in row four of table. Here, the encrypted stream is generated, $S'=10000011$. In this section only 8-bit source stream is considered for understanding the technique, but during actual implementation the block size taken as 256-bits.

Table 4.2: Decryption process of TPRT

Decryption Phase	Encrypted Stream 10000011			
Pass1	10	00	00	11
Pass2	Source Stream after Pass2			
	10	00	11	11
Pass3	Source Stream after Pass3			
	10	10	11	00
Decrypted Stream 10101100				

The decryption process of TPRT is illustrated in table 4.2. The ciphertext is, $S'=10000011$, now in the decryption pass1 this stream is broken down into four blocks each of having 2-bits size. So, there are four blocks, '10', '00', '00' and '11', which is depicted in row one of table. Then in decryption process, first block of a given length is XORED with second block and the result is replacing the first block keeping the second block intact, similarly the third block and fourth blocks are XORED and the replacing the third block keeping fourth block intact. Now to get the blocks, '10', '00', '11' and '11', which is illustrated in row two of table. In the decryption pass3 the same operation is performed but here the result of XOR between first block and second block replaces the second block keeping first block intact, similarly the result of XOR between the third block and fourth block replaces the fourth block keeping third block intact. Now got the blocks, '10', '10', '11', and '00', which is depicted in row three of table. Then writing the adjacent blocks gives the target stream. Here, the decrypted stream is generated, $S''=10101100$, which is finally given in row four of table. Therefore, if compare $S = S''$, that is, the source stream is again regenerated.

4.4 Implementation and Key Generation

To analyze the performance, TPRT has been implemented both in FPGA and C programming language. This has been implemented in VHDL for RTL design to be embedded in the FPGA based systems. A good synthesis and simulation been generated in Xilinx ISE 8.1i software. Section 4.4.1 discusses key generation process.

```

library IEEE, STD;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.all;
use work.rajdeep.all;
use std.textio.all;
use std.standard.all;
use IEEE.std_logic_textio.all;

entity TPRT_Final_VHDL is
    Port ( IN_DATA : in BIT_VECTOR (255 downto 0);
          OUT_DATA : out BIT_VECTOR (255 downto 0);
          EN_DN : inout BIT;
          ITERATION: in BIT_VECTOR(7 downto 0);
          BLOCK_SIZE : in BIT_VECTOR(7 downto 0));
end TPRT_Final_VHDL;

```

Figure 4.2: Top-level design of TPRT

This proposed technique has been implemented in IEEE VHDL using 256-bit block size. The block-size can be altered just by altering the size of bit_vector type variables, signals and ports from 255 downto 0 to n-1 downto 0 where 'n' is the block size. The modular design approach is taken while coding this cipher.

Figure 4.2 shows the top-level design of TPRT. The main features of the implementation are as follows:-

- TPRT Encryption and Decryption using same RTL design.
- Coded using Behavioural model.
- This program is implemented for text file input and output and also for RTL design for FPGA-chip.
- Encryption and decryption available for all the block size.
- This top-level module has five ports, in_data, out_data, block_size, iteration and EN_DN.

- EN_DN = 0, means encryption is being done, EN_DN = 1, means decryption is being done.
- The chip entity, in_data is the input bit stream; out_data is the output bit stream.
- block_size selects block size upon which encryption to be performed or decryption to be performed.
- Iteration selects the number of iterations to be performed during encryption and decryption for particular block size.
- EN_DN will tell the receiver side that encryption or decryption is being done.
- As this program will also work for text data files so there are three types of TEXT files used in this implementation, “in.txt” for Source block (SB), “out.txt” for Target Block (TB) and “block_size.txt” for selecting block size for encryption/decryption operation.

The rest of the coding is done by defining the package which contains functions and procedures.

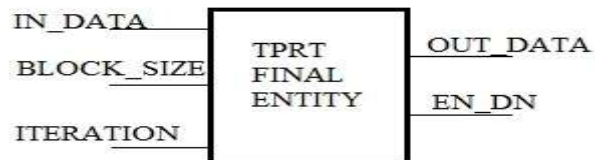


Figure 4.3: Top-level RTL design of TPRT

Figure 4.3 shows the top level RTL design of TPRT. TPRT mainly consist of three functions namely TPRT_ENDN, TPRT_Encryption_ALL, TPRT_Decryption_ALL. TPRT also consist of one procedure namely TPRT_Formation. The function in IEEE VHDL has many input parameters but only one output parameter and procedure in IEEE VHDL has many input and or output parameters. The functions and procedures which are used to realize TPRT are as follows:-

- Function TPRT_ENDN:- This is the main function which performs encryption/decryption using given block size and iteration options.

- Function TPRT_Encryption_ALL:- This is the function which performs encryption using given block size and iteration options.
- Function TPRT_Decryption_ALL:- This is the function which performs decryption using given block size and iteration options.
- Procedure TPRT_Formation:- This is the common VHDL procedure which forms the cone according to the Source Block (SB), before performing encryption/decryption.

Therefore, by using the modular design approach and behavioral approach this proposed cipher has been successfully realized in IEEE VHDL.

4.4.1 Key Generation

In the proposed TPRT, the key generation process is given for both fixed block size and also for variable block size.

Table 4.3 illustrates the key generation process for fixed block size or in other words blocks sizes of 2^n , where 'n' is any integer.

Table 4.3: Representation of number of iterations in each round by bits for 2^n

Round	Block Size	Number of Iterations	
		Decimal	Binary
8.	256	50021	1100001101100101
7.	128	49870	1100001011001110
6.	64	48950	1011111100110110
5.	32	44443	1010110110011011
4.	16	46250	1011010010101010
3.	8	4321	0001000011100001
2.	4	690	0000001010110010
1.	2	72	0000000001001000
Tag field			0

In the fixed block size key generation process eight rounds have been considered, each for 2, 4, 8, 16, 32, 64, 128, and 256-block sizes. As given in table 4.3, each round is repeated for a finite number of times, for example, for block size of 2-bits (round 1) the iteration is for 72 times, for block size of 4-bits (round 2) the iteration is for 690 times, for block size of 8-bits (round 3) the iteration is for 4321 times, for block size of 16-bits (round 4) the iteration is for 46250 times, for block size of 32-bits (round 5) the iteration is for 44443 times, for block size of 64-bits (round 6) the iteration is for 48950 times, for block size of 128-bits (round 7) the iteration is for 49870 times, and for block size of 256-bits (round 8) the iteration is for 50021 times, so, the number of iterations will form a part of the encryption-key. Although the key may be formed in many ways, for the sake of brevity it is proposed to represent the number of iterations in each round by a 16-bit binary string. Since there are eight rounds so, the binary strings are then concatenated to form a 16 X 8 = 128-bit key for a particular session. The tag field is also a part of key as given in table 4.3, tag value 0 means TPRT encryption and 1 means TPRT decryption, so, the tag field is concatenated at LSB to get the key of 129 –bit length. So, got the key as,

K=110000110110010111000010110011101011111100110110101011011001101110110001010101000010000111000010000001010110010000000000100100000.

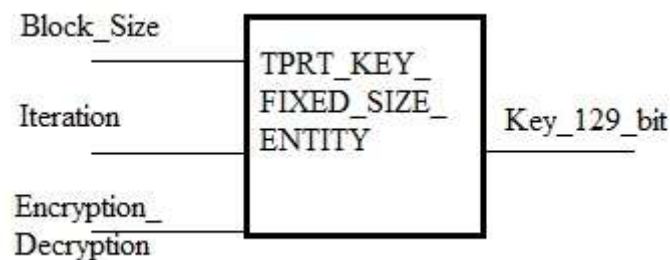


Figure 4.4: Top-level RTL design of TPRT fixed size key generation

Figure 4.4 illustrates the top level RTL design for the TPRT fixed size key generation, here it can be seen that there are four ports, the three input ports are Block_Size, Iteration, Encryption/Decryption option. The output port is the generated session key of 129-bits in size.

Table 4.4: Key generation for variable block length technique for TPRT

Round	Block Size		Number of Iterations	
	Decimal	Binary	Decimal	Binary
8.	253	11111101	203	11001011
7.	103	01100111	101	01100101
6.	99	01100011	83	01010011
5.	70	01000110	55	00110111
4.	61	00111101	38	00100110
3.	33	00100001	20	00010100
2.	17	00010001	10	00001010
1.	3	00000011	2	00000010
Tag field				0

Table 4.4 gives the key generation process for variable block size operation. Here if to see the round 1, so, the block size here is of 3-bits and the number of iteration for this block size is 2 times, for round 2 the block size is 17-bits and the number of iteration is 10 times, for round 3 the block size is 33-bits and the number of iteration is 20 times, for round 4 the block size is 61-bits and the number of iteration is 38 times, for round 5 the block size is 70-bits and the number of iteration is 55 times, for round 6 the block size is 99-bits and the number of iteration is 83 times, for round 7 the block size is 103-bits and the number of iteration is 101 times, and for round 8 the block size is 253-bits and the number of iteration is 203 times. Hence, the block size here is also a part of session key since its value is variable. In this scheme the block sizes are taken as 8-bit value and iteration is also an 8-bit value per round. Therefore for each round total bits is $8 + 8 = 16$ bits. There are eight rounds so total size is $16 \times 8 = 128$ -bits. Adding the tag field got total session key length as 129-bits. Tag value 0 means TPRT encryption and 1 means TPRT decryption. So, in either or both cases the key bit length is 128 bits + 1 tag bit = 129 bits. So, got the key as,

$K=111111011100101101100111011001010110001101010011010001100011011100111101001001100010000100010100000100010000101000000011000000100.$

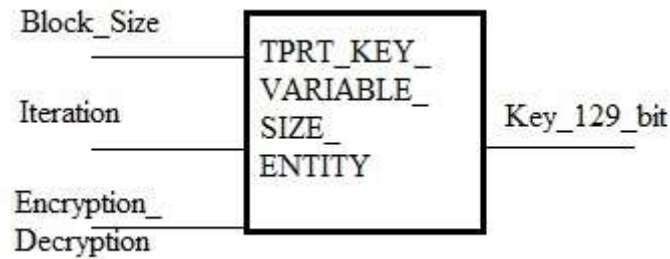


Figure 4.5: Top-level RTL design of TPRT variable size key generation

Figure 4.5 illustrates the top level RTL design for the TPRT variable size key generation, here it can be seen that there are four ports, the three input ports are Block_Size, Iteration, Encryption/Decryption option. The output port is the generated session key of 129-bits in size.

4.5 Analysis

Block ciphers are cryptographic primitives that operate on fixed size texts (blocks). Most designs aim towards secure and fast encryption of large amounts of data. The number of iterations of TPRT encryption and TPRT decryption is same so, the order of complexity of TPRT is $O(n^2)$ where 'n' is the block size, this means the encryption time and decryption time varies linearly with the block size, this is illustrated in result and comparison section. Block ciphers also serve as the building block of a number of hash functions and message authentication codes (MAC). The TPRT is a simple block cipher to implement so, it can be used to generate encryption based Message Authentication Codes (MAC). The task of cryptanalysis is to ensure that no attack violates the security bounds specified by generic attack namely exhaustive key search and table lookup attacks. The non-homogeneity using Chi-Square value is also illustrated in result and comparison section. Since, the key length is 129-bits so; brute force attack is somehow difficult. Most general types of block cipher cryptanalysis has been discussed concentrating on the algebraic attacks. While the algebraic techniques have been successful on certain stream cipher the application to block ciphers has not shown any significant results so far.

Table 4.5: Plaintext and equivalent Hex code

Plaintext	Hex code	Plaintext	Hex code	Plaintext	Hex code	Plaintext	Hex code
A	41	O	4F	<space>	20	T	54
T	54	S	53	U	55	O	4F
T	54	T	54	N	4E	M	4D
A	41	P	50	T	54	O	4F
C	43	O	4F	I	49	R	52
K	4B	N	4E	L	4C	R	52
<space>	20	E	45	L	4C	O	4F
P	50	D	44	<space>	20	W	57

Let it encrypt P = “ATTACK POSTPONED UNTILL TOMORROW”. This plaintext has been encrypted using the key obtained in section 4.4.1. During encryption the letters are converted into ASCII which is then the equivalent hex code is fed into FPGA-based implemented routine described in section 4.4. Then to get the encrypted hex value which is again converted to equivalent ASCII letters. Table 4.5 shows the plaintext letters and the corresponding hex codes, the plaintext letters are taken in column-major order.

Table 4.6: Hex code and equivalent ciphertext

Hex Code	Ciphertext	Hex Code	Ciphertext	Hex Code	Ciphertext	Hex Code	Ciphertext
10	►	5C	\	75	U	5E	^
06	♠	C3	┆	73	S	DC	—
10	►	FC	H	96	Û	8B	Ï
5D]	58	X	C3	┆	20	<space>
1A	→	68	H	F3	≤	2D	-
06	♠	A8	ı	66	F	5C	\
07	•	82	É	0F	☼	CF	±
06	♠	33	3	F3	≤	22	“

Table 4.6 shows the hex codes obtained after encryption and the corresponding ciphertext letters. Thus got the Ciphertext as, C = “►♠►]→♠♠\|ηXhçé3 usû|≤f☼≤^—Ï - \±“”.

4.6 Results and Simulations

This section gives the results obtained based on various parameters. The main results that are described here, RTL based result, frequency distribution graph, Chi-Square test for non-homogeneity, time complexity analysis and the avalanche ratio test. These are described in respective sub sections.

Section 4.6.1 gives the RTL based results got after implementation in FPGA-based systems, section 4.6.2 discuss the frequency distribution graph, section 4.6.3 gives the test for non-homogeneity with Chi-Square values, section 4.6.4 analyze the time complexity of the proposed technique and finally section 4.6.5 discuss the avalanche ratio test.

4.6.1 RTL Simulation Based Results

In this section some of the results obtained on implementing the proposed technique in VHDL. This code has been simulated and synthesized in Xilinx ISE 8.1i. The main objective is to find an efficient FPGA-based cryptographic technique for implementation in embedded systems.

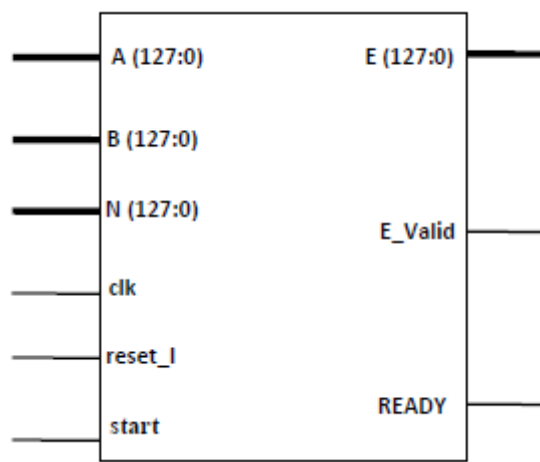


Figure 4.6: RTL diagram of RSA

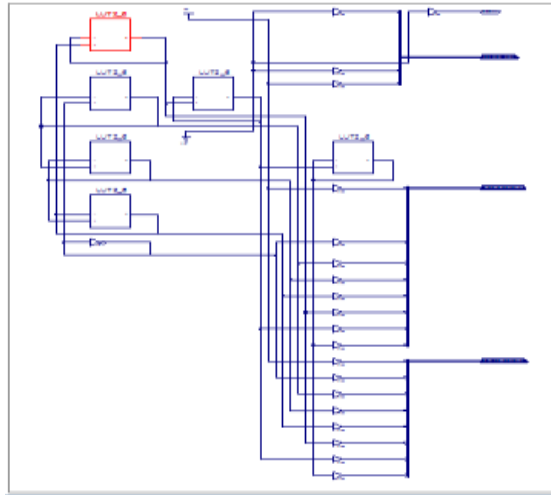


Figure 4.7: RTL diagram for Spartan 3E of the proposed TPRT

Table 4.7: HDL synthesis report (netlist generation of RSA and TPRT)

Sr No.	Netlist Components	Number	
		RSA	TPRT
1	ROMs/RAMs	430	10
2	Adders/Subtractions	3	0
3	Registers	420	20
4	Latches	80	0
5	Multiplexers	120	0

Table 4.8: HDL Synthesis Report (Timing Summary of RSA and TPRT)

Sr No.	Timing Constraint	Values	
		RSA	TPRT
1	Speed Grade	-5	-5
2	Minimum period (ns)	9.895	5.66
3	Maximum Frequency (MHZ)	101.06	101.06
4	Minimum input arrival time before clock (ns)	6.697	4.33
5	Maximum output required time after clock (ns)	4.31	3.33

Figure 4.6 gives the RTL schematic of RSA and figure 4.7 gives the RTL schematic for Spartan 3E of TPRT. If closely observing figure 4.6, it can be seen that there are many look-up tables used in the RSA. Figure 4.7 reveals that six look-up tables are used for TPRT which is quite less than that of RSA.

Table 4.7 gives the HDL synthesis report for netlist generation of TPRT and RSA. Number of ROMs used in RSA is 430 and that of TPRT is 10, number of adder/subtractor used in RSA is 3 and NIL that of TPRT, number of registers used in RSA is 420 and that of TPRT is 20, number of latches in RSA is 80 and that of TPRT is NIL and number of multiplexers used in RSA is 120 and that of TPRT is NIL. So, it is inferred that TPRT uses least number of resources than that of RSA in view of FPGA implementation.

Table 4.8 gives the HDL synthesis report for timing summary of RSA and TPRT. The minimum period of RSA is 9.86ns and TPRT is 5.66ns. Minimum input arrival time before clock of RSA is 6.66ns where for TPRT is 4.33ns. Maximum output required time after clock of RSA is 4.31ns and that of TPRT is 3.33ns. So, it is also seen here that TPRT uses much less timing summary than that of RSA.

So this implantation is synthesizable and can be burn into the Spartan 3E FPGA-chip. After synthesis of the design, the design translation, design mapping, placement of I/Os and routing has also been done successfully. The conclusions has been described in the next section.

4.6.2 Frequency Distribution Graph

This section illustrates the frequency distribution graph of RSA and TPRT. The frequency graph is the collection of different ASCII characters present in plaintext as well as in ciphertext. Although ten different files are encrypted but here the frequency distribution graph of only one such file is given, the rest gives the similar result.

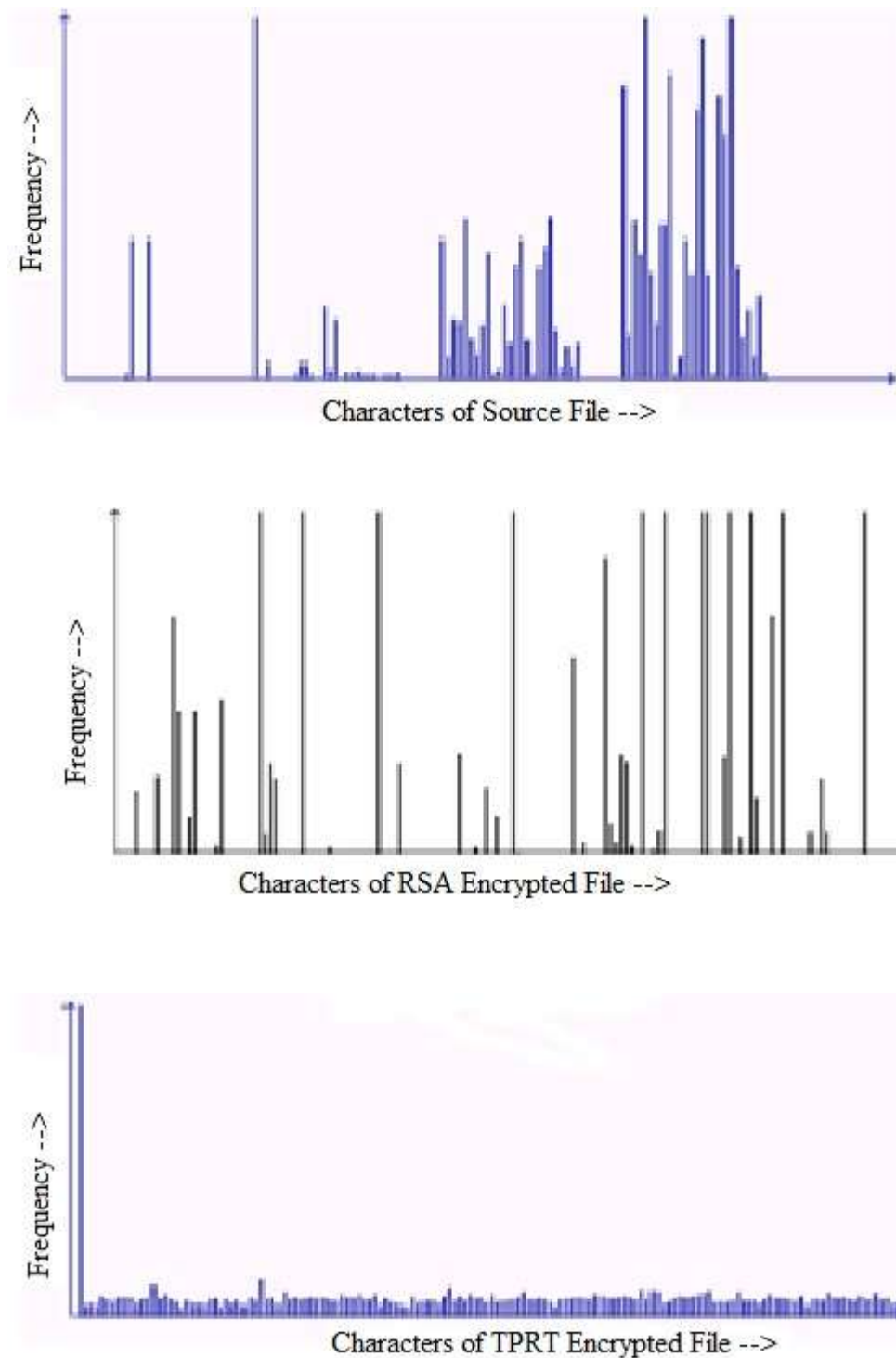


Figure 4.8: The frequency distribution graph of source, RSA encrypted and TPRT encrypted files

Figure 4.8 gives the frequency distribution graphs of source file, RSA encrypted file and TPRT encrypted file. This frequency distribution illustrates the percentage of each characters present in the file, source and encrypted. Since, ASCII character coding is used

here so, the X-axis region is 0 to 255, ASCII got 8-bit character coding. The Y-axis is the percentage of occurrences of each character. Although ten different files, with different file types, are encrypted with RSA as well as TPRT, and only frequency distribution of a single file is illustrated here. The file “GENESIS.TXT” is taken here for analysis. Observe from figure 4.8 that the frequency distribution of source file is in the region 0-127. The frequency distribution of RSA encrypted file is also not well distributed, where as the frequency distribution of TPRT encrypted file is well distributed in the region 0-255. This result illustrates the frequency of the proposed technique is well distributed than that of RSA. Hence, in terms of frequency distribution analysis this proposed technique, TPRT, is well comparable with RSA.

4.6.3 The Non-Homogeneity Test

This section computed the extent of non-homogeneity between source file and encrypted file. The parameter taken for this test is Pearsonian Chi-Square test.

Table 4.9: Chi-Square values of RSA and TPRT

Source File	File Size (Bytes)	Chi-Square Value		Degree of Freedom	
		TPRT	RSA	TPRT	RSA
license.txt	17,632	191382	30148	255	64
cs405(ei).doc	25,422	253470	185351	255	66
acread9.txt	35,121	410735	169424	255	73
deutsch.txt	47,829	505121	334371	255	77
genesis.txt	49,600	638592	396128	255	75
pod.exe	69,981	896405	761842	255	76
mspaint.exe	136,463	1203665	1053183	255	88
cmd.exe	152,028	1692655	545752	255	73
d3dim.dll	193,189	4250652	307565	255	10
clbcatq.dll	403,901	3922143	327510	255	11

Table 4.9 gives the Chi-Square values of the proposed technique (TPRT) and that of RSA. Figure 4.9 illustrates the same graphically. The Chi-Square value gives the extent of non-homogeneity between source file and encrypted file.

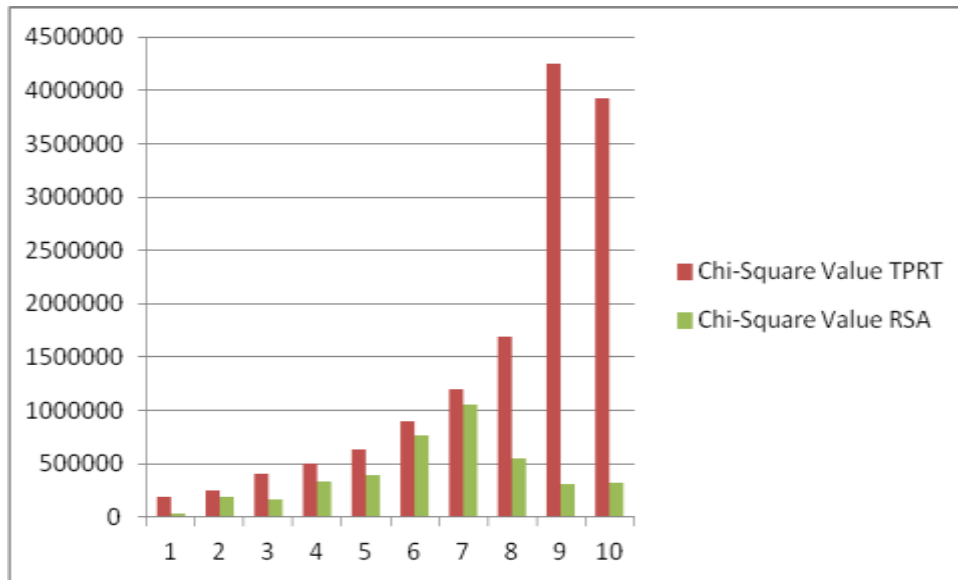


Figure 4.9: Graphical representation of Chi-Square values of RSA and TPRT

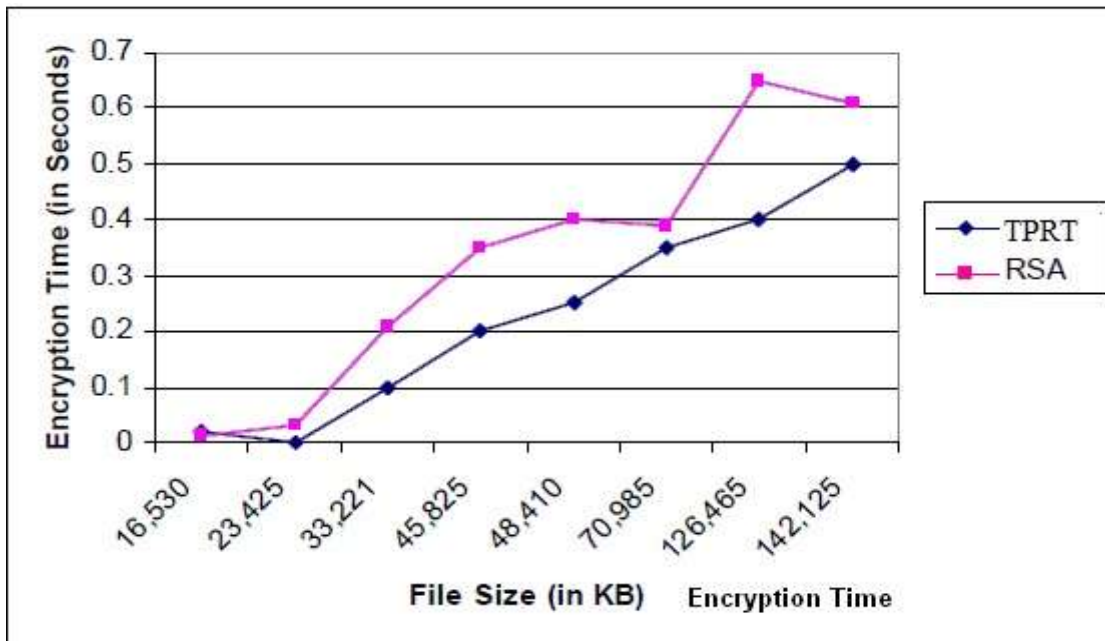
The Pearsonian Chi-Square value has been computed which is already described in Chapter 1. If looking at the table, the Chi-Square value of, first source file of TPRT comes 1,91,382 and RSA comes 30,148, for second source file TPRT comes 2,53,470 and RSA comes 1,85,351 and so on. So observing the above table and figure it has been seen that Chi-Square values of the proposed technique is quite higher than RSA and also the degree of freedom comes to be at a value of 255 in TPRT which says a well distribution of characters present in the TPRT encrypted files than that of source file. The degree of freedom of RSA comes under 100 in all the source files. Hence, from the study of the degree of freedom it is seen that the character distribution of TPRT encrypted file is well distributed than that of RSA which is at par with the result of frequency distribution already discussed in section 5.2.1. Hence, in terms of Chi-Square value analysis this proposed technique, TPRT, is well comparable with RSA.

4.6.4 The Time Complexity Analysis

This section illustrates the time complexity analysis and for the purpose encryption time and decryption time is taken into account.

Table 4.10: Comparisons of time complexity analysis of TPRT and RSA

Source File	File Size (Bytes)	Encryption time (in Seconds)		Decryption time (in seconds)	
		TPRT	RSA	TPRT	RSA
license.txt	17,632	0.02	0.01	0.10	0.28
cs405(ei).doc	25,422	0.00	0.03	0.00	0.30
acread9.txt	35,121	0.10	0.21	0.10	1.67
deutsch.txt	47,829	0.20	0.35	0.11	3.51
genesis.txt	49,600	0.25	0.40	0.20	5.06
pod.exe	69,981	0.35	0.39	0.35	4.34
mspaint.exe	136,463	0.40	0.65	0.40	8.37
cmd.exe	152,028	0.50	0.61	0.42	6.59
d3dim.dll	193,189	0.52	0.75	0.50	10.15
clbcatq.dll	403,901	0.60	0.95	0.55	11.70



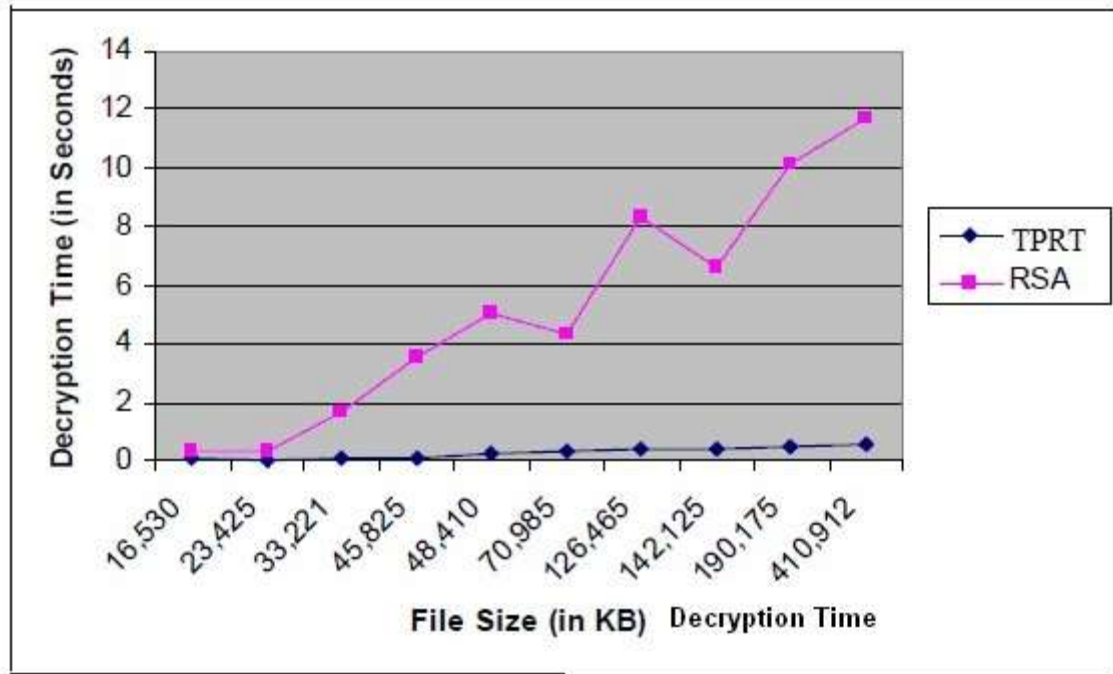


Figure 4.10: Graphical comparisons of encryption and decryption time of TPRT and RSA

From table 4.10 it is seen that the encryption time and decryption time of the proposed technique and that of RSA within same frame graphically in figure 4.10. The time complexity analysis is one of the important factors in algorithm design. Here both encryption time and decryption time is tabulated and shown in the figure. The pink line shows the time complexity of RSA and blue line gives the time complexity of this proposed technique, TPRT. If observing the encryption time, TPRT time of encryption is marginally lower than that of RSA, and observing the decryption time than it is seen that TPRT time of decryption is quite less than that of RSA. Hence it may be concluded that the time complexity of the proposed technique, TPRT, is quite less than that of RSA.

4.6.5 The Avalanche Ratio

The Avalanche ratio is another important parameter for the cryptographic security. Ten files have been taken for this analysis. Some bits of plaintext files have been modified and these ten files again encrypted. The difference between original encrypted files and modified encrypted files has been recorded as avalanche ratio in percentage.

Table 4.11: Comparisons of avalanche ratio of TPRT and RSA

Source File	File Size (Bytes)	Avalanche Ratio (in Percentage)	
		RSA	TPRT
license.txt	17,632	58.0	77.7
cs405(ei).doc	25,422	60.0	80.0
acread9.txt	35,121	75.0	88.8
deutsch.txt	47,829	78.9	89.0
genesis.txt	49,600	80.9	87.0
pod.exe	69,981	58.0	77.0
mspaint.exe	136,463	58.9	76.0
cmd.exe	152,028	67.0	77.0
d3dim.dll	193,189	67.9	82.9
clbcatq.dll	403,901	68.0	88.5

Table 4.11 illustrates the result of avalanche ratio of the proposed TPRT. During this test some characters/bits in the source file have been modified and then again these modified source files are encrypted. Then the percentage of the difference between the original encrypted files and the modified encrypted files are taken. It is observed from table 4.11 that the avalanche ratio of the proposed technique is nearly 80% and that of RSA is 65%, hence in terms of avalanche ratio analysis TPRT is quite comparable with RSA.

4.7 Discussions

The technique given here is easily implemented in high level language and also in VHDL. This technique is very easy and it's implemented in FPGA-based systems, the goal of fast execution and strong cryptanalysis requirements are also obtained here. Moreover this technique can be fabricated in chip to be used in embedded systems. The main goal of the author is to develop an efficient FPGA-based crypto hardware and this proposed technique is the first step towards this.

Chapter 5
Triangular Modulo Arithmetic Technique (TMAT)

5.1 Introduction

Unlike the TPRT, Triangular Modulo Arithmetic Technique (TMAT) is designed in such a manner that neither any cycle is formed nor the process of decryption is the same as that of the encryption. There is no positional orientation of bits. In TMAT a generating function is used to covert plaintext to ciphertext, thus $C = f_k(P)$, where 'P' is plaintext block, 'C' is the ciphertext, f_k is the generating function and 'k' is the secret. The generating function of TMAT is directly related with the different bits present in the plaintext. The source block is considered as a stream of bits, it is then divided into blocks of bits of same size and then generating function is applied to each of the blocks to get the ciphertext. The generating function of TMAT has two parts; first one is the modulo- 2^n addition operation and second one is triangular operation.

In contrast to TPRT technique discussed in chapter 4 and the TMAT technique there is application of Boolean algebra as well as non Boolean operation during encryption as well as decryption. During encryption, the decimal equivalent of the block of bits under consideration is one integral value from which the recursive modulo- 2^n operation starts; this operation is sandwiched between two triangular operations. The modulo- 2^n operation is performed between successive two blocks, before and after modulo operation the triangular operation is performed. These three processes is operated in whole plaintext considering different block sizes and iterations therefore recursively these processes is carried out to a finite number of times, which is exactly the length of the source block. During encryption the flow of these three processes is from MSB-to-LSB direction. To generate the source code during decryption, bits in the target block are to be considered along LSB-to-MSB direction. In second iteration one triangular operation is sandwiched between two modulo- 2^n addition operations.

Section 5.2 discussed the algorithm of TMAT with a block level diagram, section 5.3 gives a detailed example of encryption and decryption process, section 5.4 discussed the implementation issues with key generation, section 5.5 gives a brief analysis, section 5.6 discussed the results obtained based on implementation and a brief discussions are given in section 5.7.

5.2 The Algorithm of TMAP

The proposed scheme has been developed in conjunction with two algorithm, MAT [17] and Triangular algorithm [148, 149]. The source file is taken as binary streams. The input stream size and input key size have been considered 512 bits and 128 bits for the implementation, though the scheme can be implemented for larger input stream sizes as well as any input size also. Section 5.2.1 briefly discuss the modulo addition.

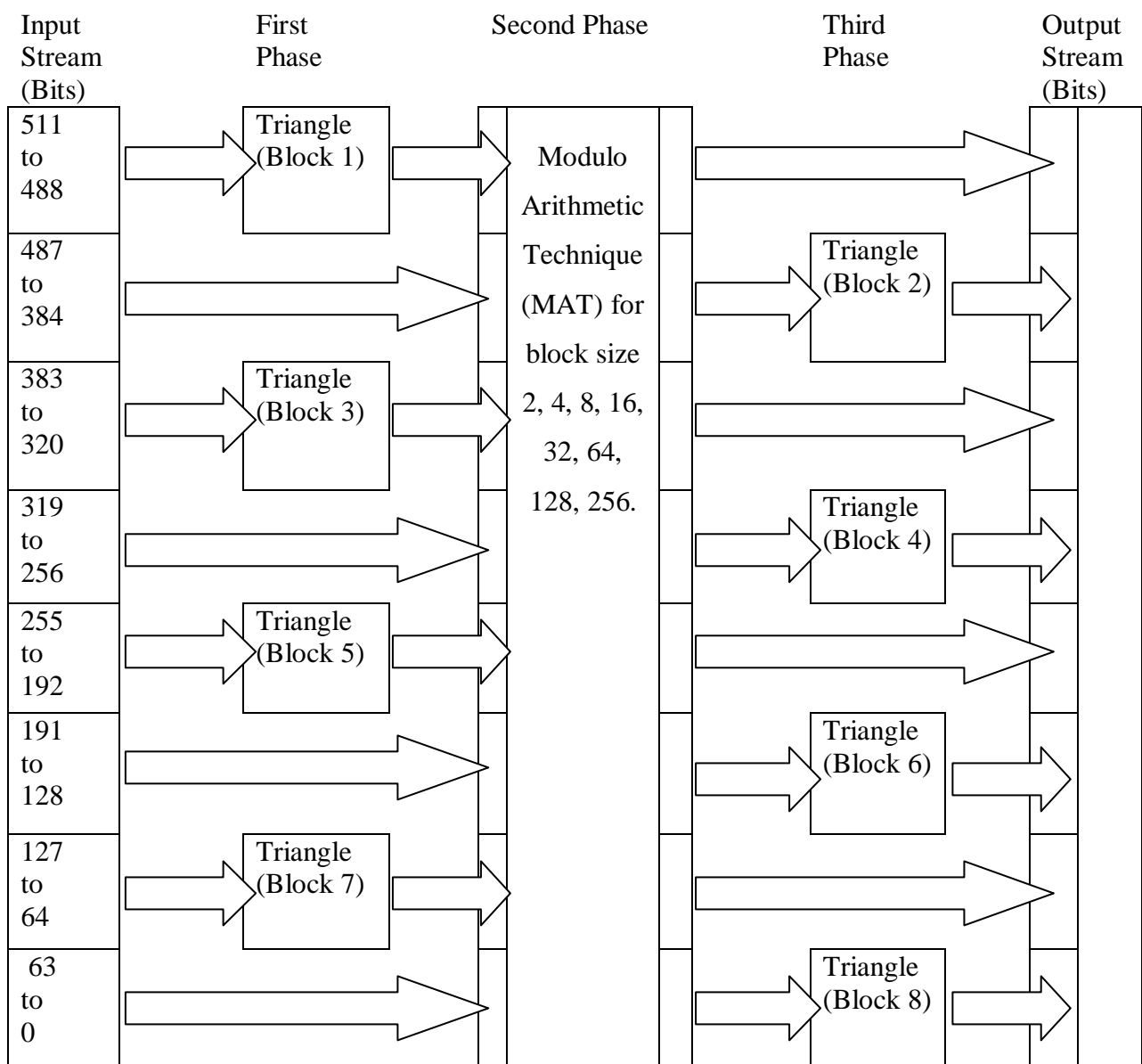


Figure 5.1: Block diagram of TMAP

The proposed algorithm is consisted of three phase where the Triangular algorithm is performed in Phase 1 and Phase 3 and that of MAT is performed in Phase 2. The key

generation and low level implementation will be discussed in section 5.4. Figure 5.1 shows the block diagram of TMAP.

In Phase 1, 512 bits input stream, S , is broken into 8 numbers of equal size blocks, each containing 64 bits and the Triangular algorithm is implemented on block number 1, 3, 5, 7 (i.e. odd blocks) where the rest of the blocks are (even blocks) remain unchanged. Consider that the source block size t (here 64). In the Triangular Encryption technique an intermediate block of size $(t-1)$ is generated from the source block, by applying the exclusive NOR (XNOR) operation between each two consecutive bits. In the next step a new block of size $(t-2)$ is generated from previous block of size $(t-1)$ and this process goes on until the generation of block size 1. All these blocks under consideration is taken together to form an equilateral triangle-like shape. After the formation of such a triangular shape, putting together either the MBSs or the LSBs of all the blocks under consideration in either sequence, the target block is formed. The key takes a vital role because only by knowing this key the receiver can understand how the target block is formed from the triangular shape. The encrypted stream of bits is generated by putting together all the target blocks. Then the both changed and unchanged blocks are concatenated and formed bit stream of 512 bits, say S^1 . The Triangular technique is shown in figure 5.2. There are four ways to encrypt in triangular operation, ‘00’ is taking MSB from top to bottom, ‘01’ is taking MSB from bottom to top, ‘10’ is taking LSB from top to bottom and ‘11’ is taking LSB from bottom to top.

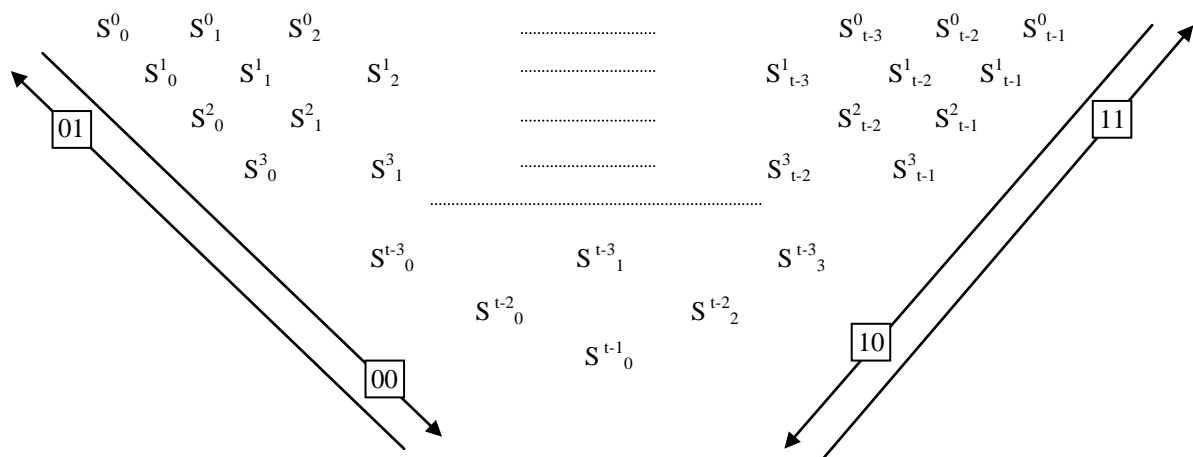


Figure 5.2: Triangle formation

In phase 2 of the technique, Modulo Arithmetic Technique (MAT) is performed on that stream of 512 bits. This is done in 8 rounds. The input stream, S^1 , is broken into a number of blocks, each containing n bits where $n=2^k$, $k=1,2,3,\dots,8$, k denotes the round number. So, $S^1 = B_1B_2B_3\dots B_m$, where $m=512/n$. Starting from the MSB, the blocks are

paired as $(B_1, B_2), (B_3, B_4), (B_5, B_6), \dots, (B_{m-1}, B_m)$. The addition is performed between two blocks of each pair and the content of the second block of each pair is replaced by the result. This will be going on until the content of the last block B_m is replaced by the result. The process is repeated and each time the block size increases till $n=256$. So a new encrypted stream, S^2 is generated after MAT is performed with block size 256.

- Round 1: In this round of encryption, block size is taken as 2, it means $k=1$ and addition is performed between each pair of blocks and second block of each pair is replaced by the result. This round is repeated for a finite number of times and the number of iterations will form a part of the session key as discussed in Section II.
- Round 2: Identical operation is performed as in Round 1 with block size 4 (i.e. $k=2$).

Eight rounds are performed repeatedly with increasing block size to encrypt the stream with varying block size up to 256. So after the completion of Round 8 another encrypted bit stream is generated, say, S^3 .

In phase 3, the intermediate binary stream, S^3 is divided into 8 equal size of blocks and Triangular algorithm is imposed on block no 2, 4, 6, 8 (i.e. even blocks) and rest of the blocks are (odd blocks) remain unchanged. After which all blocks are concatenated together to produce final output stream, S^{en} .

During decryption, the reverse operation is performed. In phase 1, triangular algorithm is performed on block no 2, 4, 6, 8 (i.e even blocks) and odd blocks are remain unchanged and then in the phase 2, modulo subtraction, is performed instead of performing modulo addition where block size starts from 256 and end with 2 ($n=2^k, k=8, 7, 6, \dots, 3, 2, 1$). In phase 3, Triangular algorithm is performed on block no 1, 3, 5, 7 (i.e. odd blocks) and even blocks are remain unchanged. Formation of result is quite different than that of encryption technique. If selection is 00 or 11 during encryption, it is same for decryption technique but if it is 01 or 10, interchange is done between them.

5.2.1 The Modulo Addition

An alternative method for modulo addition has been proposed here to make the calculations more simple. The need for computation of decimal equivalents of the blocks is avoided here since it will be got large decimal integer values for large binary blocks. In the proposed method the carry out of the MSB has been discarded after the addition of two blocks of each pair. For example, if add 1101 and 1000 got 10101. In terms of decimal values, $13+8=21$. Since the modulus of addition is 16 (2^4) in this case, the result of addition should be 5 ($21-16=5$). Discarding the carry from 10101 is equivalent to subtracting 10000 (i.e. 16 in decimal). So the result will be 0101, which is equivalent to 5 in decimal. The same is applicable for any block size.

5.3 Example

Although the proposed scheme is applicable for a 512-bit input stream but here 16 bit input stream has been considered for the convenience, to make the process simple for understanding. Section 5.3.1 discuss the encryption scheme and section 5.3.2 discuss the decryption scheme

5.3.1 The Encryption Process

Consider a stream of 16 bits stream, say $S = 1101001100011011$.

In first phase, the input stream is divided into four blocks consisting of 4 bits each. The Triangular technique is performed on odd blocks (i.e. Block 1 and Block 3) and even blocks (i.e. Block 2 and Block 4) are remaining unchanged.

Consider that the selection of key for block 1 is 00 and Block 3 is 11. Then output from block 1 is 1100 and from block 3 is 0101. Block 2 and block 4 are remaining unchanged. So after Phase 1 output, $S^1 = 1100110001010010$. This output is the input for phase 2. Figure 5.3 shows the example of phase 1.

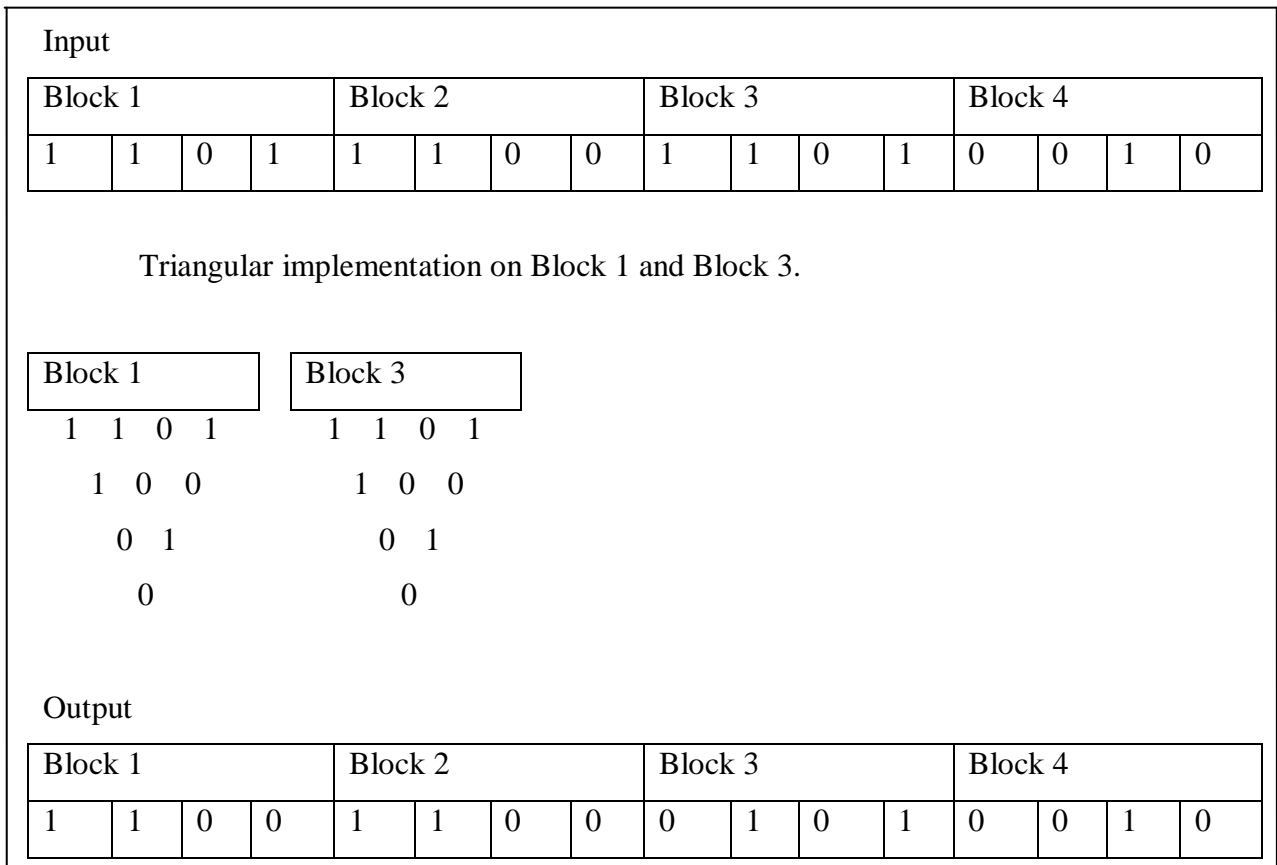
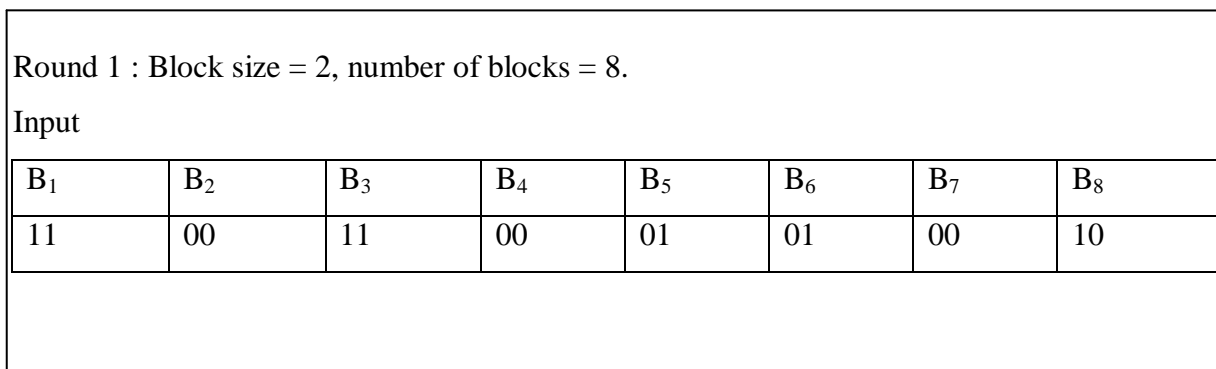


Figure 5.3: Encryption example of phase 1 in TMAT

In phase 2, MAT is performed for block size 2, 4 and 8 (as input is taken 16 bits, so maximum block size is to be 8). So total number of rounds is 3. Each round is performed only once to make the process simple for understanding. Figure 5.4 shows the details of this phase.



(B1, B2) Modulo Addition, B2 is replaced by result. Same operation is performed for (B3, B4), (B5, B6) and (B7, B8).

Output

B ₁	B ₂	B ₃	B ₄	B ₅	B ₆	B ₇	B ₈
11	11	11	11	01	10	00	10

Round 2 : Block size = 4, number of blocks = 4.

Input

B ₁	B ₂	B ₃	B ₄
1111	1111	0110	0010

Output

B ₁	B ₂	B ₃	B ₄
1111	1110	0110	1000

Round 3 : Block size = 8, number of blocks = 2.

Input

B ₁	B ₂
11111110	01101000

Output

B ₁	B ₂
11111110	01100110

Figure 5.4: Encryption example of phase 2 in TMAT

So, on applying phase 2, generated output, $S^2 = 111111001100110$, which is input for phase 3.

In phase 3 the triangular technique is again performed on even blocks, i.e. block 2 and block 4 where odd blocks (blocks 1 and blocks 2) are remain unchanged.

Input															
Block 1				Block 2				Block 3				Block 4			
1	1	1	1	1	1	1	0	0	1	1	0	0	1	1	0

Triangular implementation on Block 2 and Block 4.

Block 2				Block 4			
1	1	1	0	0	1	1	0
	1	1	0		0	1	0
		1	0			0	0
			0				1

Figure 5.5: Encryption example of phase 3 in TMAT

Consider that the selection key for block 2 is 01 and block 4 is 10. Then output from block 2 is 1101 and from block 4 is 0010. Block 1 and block 3 are remaining unchanged. So after on completion of phase 3 output, $S^{en} = 1111011101100001$. This is the final encrypted output. Figure 5.5 elaborate the example.

5.3.2 The Decryption Process

The output stream, which was generated during encryption technique, has been considered as input bit stream for decryption process.

Input															
Block 1				Block 2				Block 3				Block 4			
1	1	1	1	0	1	1	1	0	1	1	0	0	0	0	1

Triangular implementation on block 2 and block 4.

Block 2				Block 4			
0	1	1	1	0	0	0	1
	0	1	1		1	1	0
		0	1			1	0
			0				0

Output

Block 1				Block 2				Block 3				Block 4			
1	1	1	1	1	1	1	0	0	1	1	0	0	1	1	0

Figure 5.6: Decryption example of phase 1 in TMAT

In first phase, the input stream is divided into four blocks with 4 bits each. The Triangular technique is performed on even blocks (i.e. block 2 and block 4) and odd blocks (i.e. block 1 and block 3) are remaining unchanged.

As the selection keys were considered during encryption technique, 01 for block 2 and 10 for block 4. Then output from block 2 is 1010 and from block 4 is 0011. Block 1 and block 3 remain unchanged. So after phase 1 output, $S_1^1 = 1111111001100110$, which is the input of phase 2. Figure 5.6 shows the example.

In second phase, MAT is performed but instead of using modulo addition, modulo subtraction is used. Block size is used in reverse order (i.e. 8, 7, 6, ..., 1). Figure 5.7 shows this phase.

Round 1 : Block size = 8, number of blocks = 2							
Input							
B ₁				B ₂			
11111110				01100110			
Output							
B ₁				B ₂			
11111110				01101000			
Round 2 : Block size = 4, number of blocks = 4							
Input							
B ₁		B ₂		B ₃		B ₄	
1111		1110		0110		1000	
Output							
B ₁		B ₂		B ₃		B ₄	
1111		1111		0110		0010	
Round 3 : Block size = 2, number of blocks = 8							
Input							
B ₁	B ₂	B ₃	B ₄	B ₅	B ₆	B ₇	B ₈
11	11	11	11	01	10	00	10
Output							
B ₁	B ₂	B ₃	B ₄	B ₅	B ₆	B ₇	B ₈
11	00	11	00	01	01	00	10

Figure 5.7: Decryption example of phase 2 in TMAP

So on completion of phase 2 output, $S_1^2 = 1100110001010010$, which is the input of phase 3.

In this phase 3, the Triangular technique is applied on odd blocks (i.e. block 1 and block 3) and even blocks (i.e. block 2 and block 4) are remaining unchanged. Figure 5.8 shows this example.

Input															
Block 1				Block 2				Block 3				Block 4			
1	1	0	0	1	1	0	0	0	1	0	1	0	0	1	0
Block 1				Block 3											
1	1	0	0	0	1	0	1								
1	0	1		0	0	0									
0	0			1	1										
1				1											
Output															
Block 1				Block 2				Block 3				Block 4			
1	1	0	1	1	1	0	0	1	1	0	1	0	0	1	0

Figure 5.8: Decryption example of phase 3 in TMAT

As the keys were considered during encryption technique, 00 for block 1 and 11 for block 3. Then output from block 1 is 1101 and from block 3 is 1101. Block 2 and block 4 remain unchanged. So on completion of phase 3 output, $S_1^{de} = 1101110011010010$. The decrypted bit stream: $S_1^{de} = 1101110011010010$. So $S_1^{de} = S^{en}$.

5.4 Implementation and Key Generation

The proposed technique has been implemented in IEEE VHDL using 8-bit block size. The block-size can be increased just by increasing the size of bit_vector type variables, signals and ports from 7 downto 0 to n-1 downto 0 where 'n' is the block size. The modular design approach is taken while coding this cipher. Figure 5.9 shows the top-level design of TMAT and figure 5.10 gives the top level RTL design. Section 5.4.1 gives the key generation process.

```

library IEEE, STD;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_std.all;
use work.rajdeep.all;
use std.textio.all;
use std.standard.all;
use IEEE.std_logic_textio.all;

entity TMat_Final_VHDL is
  Port ( IN_DATA : in BIT_VECTOR (7 downto 0);
        OUT_DATA : out BIT_VECTOR (7 downto 0);
        EN_DN : inout BIT;
        OPTION_DATA : in BIT_VECTOR(2 downto 0));
end TMat_Final_VHDL;

```

Figure 5.9: Top level design of TMat

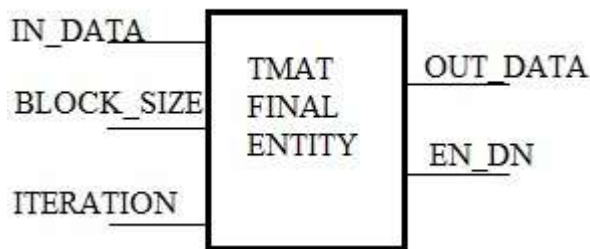


Figure 5.10: Top level RTL design of TMat

The main features of the implementation are given below:-

- Triangular Encryption and Decryption using all options.
- Coded using behavioural model.
- This program is implemented for text file input and output and also for RTL design for FPGA-chip.

- All the four options are available for encryption/decryption, 00→1st option, 01→2nd option, 10→3rd option and 11→4th option.
- This top-level module have four ports, in_data, out_data, option_data and EN_DN.
- EN_DN = 0, means encryption is being done, EN_DN = 1, means decryption is being done.
- The chip entity, in_data is the input bit stream; out_data is the output bit stream.
- option_data selects encryption to be performed or decryption to be performed.
- option_data also selects which of the four types of encryption/decryption is to be performed.
- option_data is of 3-bit, The LSB selects the encryption or decryption is to be performed.
- First two bit of option_data selects the encryption/decryption type from the four alternatives.
- EN_DN will tell the receiver side that encryption or decryption is being done.
- There three types of TEXT files used in this implementation, “in.txt” for Source block (SB), “out.txt” for Target Block (TB) and “option.txt” for dual purpose of selecting option and encryption/decryption choice.

The rest of the coding is done by defining the package which contains functions and procedures. The functions and procedures which are used to realize TMAT are noted below:-

- Function TMAT_ENDN:- This is the main function which performs encryption/decryption using all options.
- Function TMAT_Encryption_ALL:- This is the function which performs encryption using all options.
- Function TMAT_Decryption_ALL:- This is the function which performs decryption using all options.
- Function TMAT_Encryption_00, Function TMAT_Encryption_01, Function TMAT_Encryption_10, and Function TMAT_Encryption_11:- These are the four encryption functions according to each of the options.

- Function TMAT_Decryption_00, Function TMAT_Decryption_01, Function TMAT_Decryption_10, and Function TMAT_Decryption_11:- These are the decryption function according to each of the options.
- TMAT_ADD:- This function performs the modulo addition of two successive blocks and store the result in second block.
- Procedure TMAT_Formation:- This is the common VHDL procedure which forms the triangle according to the Source Block (SB), before performing encryption/decryption.
- The function in IEEE VHDL has many input parameters but only one output parameter and procedure in IEEE VHDL has many input and or output parameters.

Therefore, by using the modular design approach and behavioral approach this proposed cipher has been successfully realized in IEEE VHDL.

5.4.1 Key Generation

In the proposed technique, eight blocks (block 1, block 2, block 3, block 4, block 5, block 6, block 7 and block 8) have been considered for Triangular and eight rounds (for block size 2, 4, 8, 16, 32, 64, 128 and 256) have been considered for MAT. In case of the Triangular technique as 2 bits are required for selection from each block, so for eight blocks, 16 bits are required from 128 bits key. So 16 bits are selected from LSB of 128 bits key for eight rounds of the Triangular technique. Then 16 bits are equally divided into eight blocks (2 bits in each). From MSB of that 8 blocks are used for block 1 to block 8. Each round of MAT is repeated for a finite number of times and the number of iterations is a part of the 112 bits from MSB of 128 bits key. Then those 112 bits are equally divided and formed 8 blocks (14 bits in each). Each block of those 8 blocks are used as number of iterations of a round (from Round 1 to Round 8). In case of decryption technique, same key is used in the same way. Example in section 5.4.1.1 illustrates the key generation process.

5.4.1.1 Example of Key Generation

The key generation procedure has been illustrated in this section. As proposed scheme consists of three phase and same procedure is used for phase 1 and phase 3, the example will be shown in two steps. Table 5.1 shows target block selection using selection key and table 5.2 shows target block selection for the example. Consider a particular session, where 128 bits input key stream is: 00000000010010100000001010101000000100001011011010101101100011001010110110011011101111011101110111011000010110011101100001101100100.

So, 16 bits from LSB are used for the Triangular algorithm and remaining 112 bits are used for MAT. Section 5.4.1.1.1 illustrates the key generation for phase 1 (Triangle) and phase 3 (Triangle), section 5.4.1.1.2 illustrates key generation for phase 2 (MAT).

Table 5.1: Target block selection using selection key

Sl. No.	Selection Key (2 bits)	Target Block Selection
1.	00	Taking all the MSBs starting from the source block till the last block generated
2.	01	Taking all the MSBs starting from the last block generated till the source block
3.	10	Taking all the LSBs starting from the source block till the last block generated
4.	11	Taking all the LSBs starting from the last block generated till the source block

5.4.1.1.1 Key Generation for Phase 1 and Phase 3

16 bits key string is: 1100100101111000.

Table 5.2: Key distribution for the Triangular Technique.

Block No.	Phase	Target Block Selection		
		Selection key (2 bits)	Used for Encryption	Used for Decryption
1	1	11	11	11
2	3	00	00	00
3	1	10	10	01
4	3	01	01	10
5	1	01	01	10
6	3	11	11	11
7	1	10	10	01
8	3	00	00	00

5.4.1.1.2 Key Generation for Phase 2

In this phase of key generation, 112 bits key is used in MAT for block sizes 2, 4, 8, 16, 32, 64, 128, and 256 bits, respectively.

Table 5.3: Key generation for MAT

Round No.	Block Size	No. of Iteration	
		Binary	Decimal
1	2	00000000010010	18
2	4	10000000101010	8234
3	8	10000001000010	8258
4	16	11011010101101	13997
5	32	10001100101011	9003
6	64	01100110111011	6587
7	128	11101110111011	15291
8	256	00001011001110	718

Table 5.3 shows the key distribution of phase 2. 112 bits key is: 00000000010010100000001010101000001000010110110101011011000110010101101100 1101110111110111011101100001011001110.

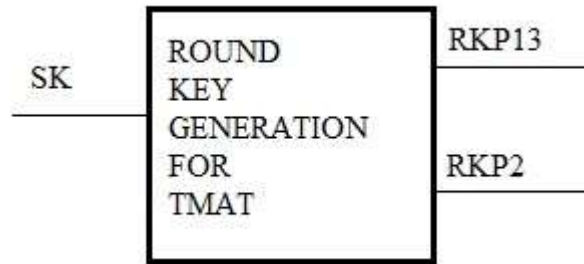


Figure 5.11: Top level RTL for round key generation for TMAT

Figure 5.11 shows the top level RTL for round key generation for TMAT, in this entity there are three ports as follows:-

- SK:- It's the input port where user gives 128 bit session key.
- RKP13:- It's the output port where LSB 16-bits are taken as round key for phase 1 and phase 2 of TMAT.
- RKP2:- It's the output port which is the rest 112-bits round key for phase 2 of TMAT.

Therefore it have been successfully implemented TMAT and key generation through VHDL implementation for FPGA.

5.5 Analysis

TMAT gives four different ways to encrypt, as TMAT consist of two techniques, the triangular and modulo addition. Figure 5.2 shows the formation of triangle and following are the four ways by which encryption can be done:-

- Option 00: Taking all the MSBs starting from the source block till the last block is generated.
- Option 01: Taking all the MSBs starting from the last block generated till the source block.
- Option 10: Taking all the LSBs starting from the source block till the last block generated.

- Option 11: Taking all the LSBs starting from the last block generated till the source block.

TMAT has four ways of decryption also given in the following points:-

- Option 00: Taking all the MSBs starting from the source block till the last block is generated.
- Option 01: Taking all the LSBs starting from the source block till the last block generated.
- Option 10: Taking all the MSBs starting from the last block generated till the source block.
- Option 11: Taking all the LSBs starting from the last block generated till the source block.

Therefore in single implementation, have four ways of encryption and decryption. Since the formation of triangle consist of matrix implementation so the algorithmic complexity of TMAT is $O(n^2)$.

5.6 Results and Simulations

This section gives the results found on various parameters. The main results that are described here are RTL based results, frequency distribution graph, Chi-Square test for non-homogeneity, time complexity analysis and the avalanche ratio. These are all described in respective sub sections. Section 5.6.1 discuss results of RTL/Hardware implementation, section 5.6.2 discuss the results of frequency distribution graph, section 5.6.3 discuss the results of Chi-Square test for non-homogeneity of source files and encrypted files, section 5.6.4 discuss the results of time complexity and section 5.6.5 discuss the results of avalanche ratio test.

5.6.1 RTL Simulation Based Result

In this section results obtained on after implementing the proposed technique in VHDL. This code has been simulated and synthesized in Xilinx 8.1i. The main objective is to

find an efficient FPGA-based cryptographic technique for implementation in embedded systems.

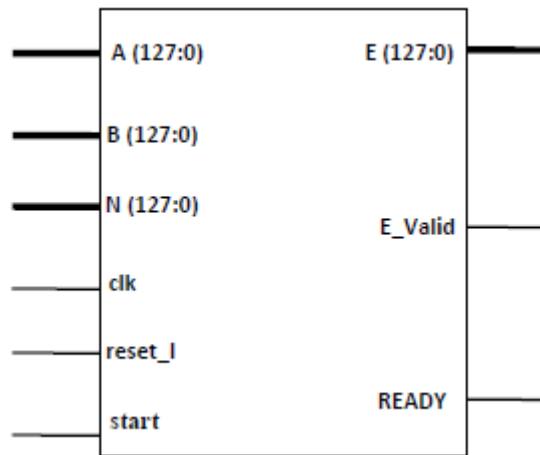


Figure 5.12: RTL diagram of RSA

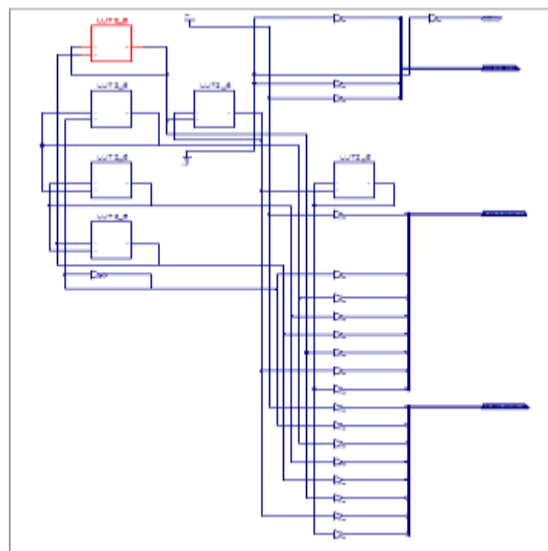


Figure 5.13: Spartan 3E RTL diagram of TPRT

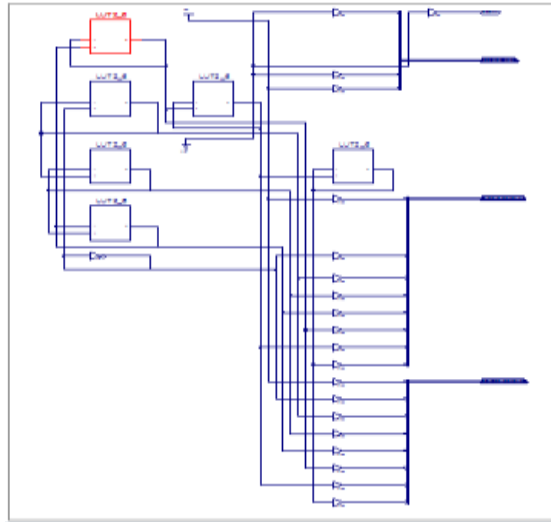


Figure 5.14: Spartan 3E RTL diagram of TMAT

Figure 5.12 gives the RTL diagram of RSA; figure 5.13 shows the Spartan 3E RTL diagram of previous technique TPRT and figure 5.14 shows the Spartan 3E RTL diagram of proposed technique TMAT. If closely observing figure 5.14, it can be seen that six Look-Up Tables (LUTs) are used here. So this implantation is synthesizable and can be burn into the Spartan 3E FPGA-chip. On synthesis of the design, the design translation, design mapping, placement of I/Os and routing has also been done successfully. RTL diagram is created after successfully implementation of the technique in VHDL. Since there is not much difference in the encryption step and decryption step so the two RTL diagrams comes to be almost same and which is at par with the theoretical description of techniques, TPRT and TMAT.

Table 5.4: HDL synthesis report (Netlist generation of RSA, TPRT and TMAT)

Sr No.	Netlist Components	Number		
		RSA	TPRT	TMAT
1	ROMs/RAMs	430	10	14
2	Adders/Subtractions	3	0	2
3	Registers	420	20	30
4	Latches	80	0	0
5	Multiplexers	120	0	0

Table 5.5: HDL synthesis report (Timing summary of RSA, TPRT and TMAT)

Sr No.	Timing Constraint	Values		
		RSA	TPRT	TMAT
1	Speed Grade	-5	-5	-5
2	Minimum period (ns)	9.895	5.66	7.95
3	Maximum Frequency (MHZ)	101.06	101.06	101.06
4	Minimum input arrival time before clock (ns)	6.697	4.33	5.55
5	Maximum output required time after clock (ns)	4.31	3.33	4.25

Table 5.4 gives the netlist generation of RSA, TPRT and TMAT. The number of ROMs/RAMs has increased in this proposed technique, TMAT, adder/sub tractors are also there for TMAT but it was nil for TPRT, number of registers used has also increased in TMAT where as number of latches and multiplexers are still nil in TMAT. So it can be said here that the number of resources used has increased for this proposed technique, TMAT, than TPRT but keeping the technique, TMAT, cryptographically strong than TPRT. It can be also said that comparing with RSA, TMAT uses few less resources but in the same time it is giving comparable results.

Table 5.5 gives the timing constraint of TMAT. It can be seen that the timing parameters have subsequently increased in this proposed technique, TMAT than TPRT. This result is also at par with the theoretical foundation of the technique and same result also got in previous time-complexity analysis. It can be also said here that comparing with RSA, TMAT is much faster in timing summary but in the same time it is giving comparable results.

5.6.2 The Frequency Distribution Graph

This section compares the frequencies of ASCII characters found in plaintext/source file and ciphertext/encrypted file through frequency distribution graph.

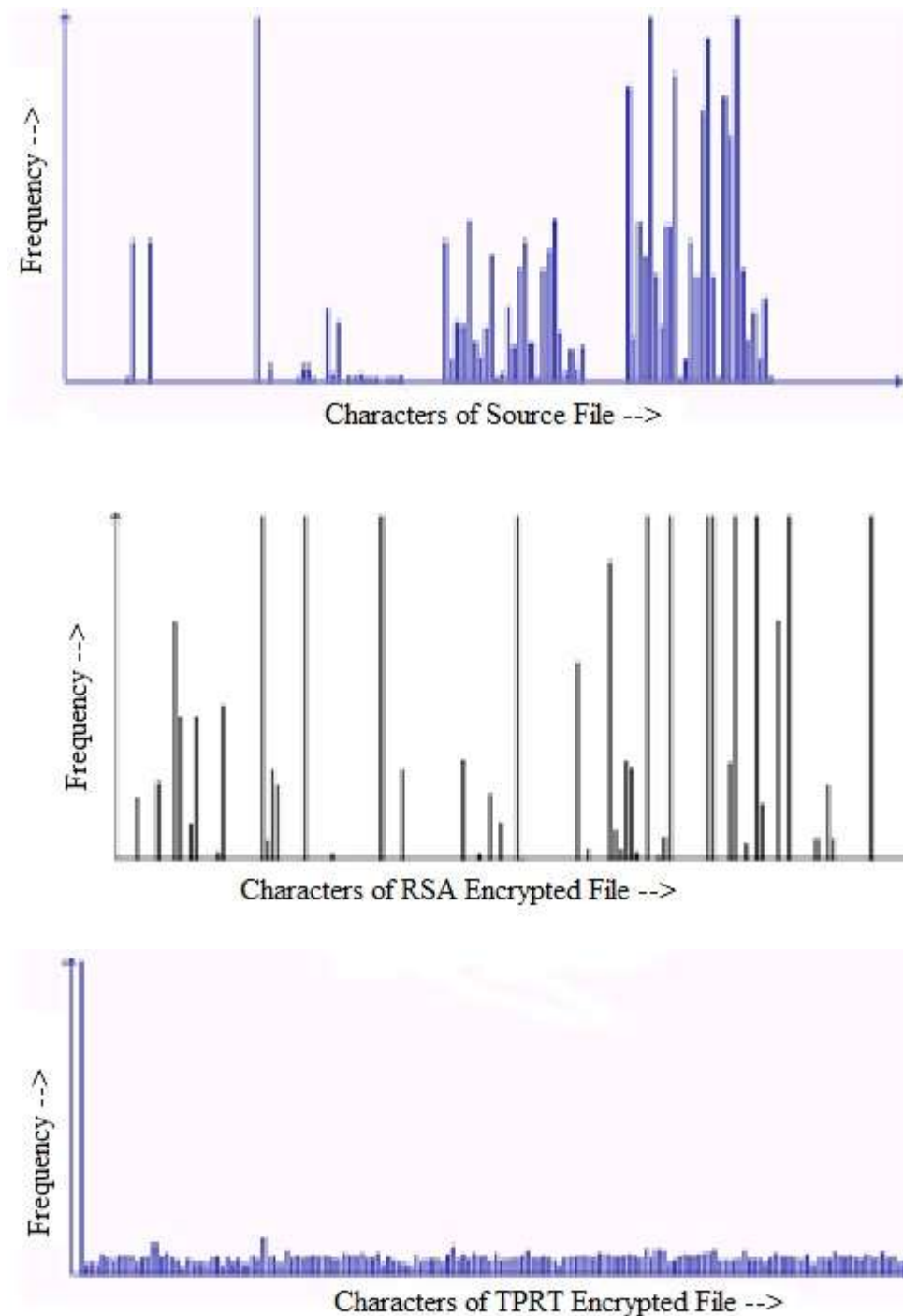


Figure 5.15: The frequency distribution graph of source, RSA encrypted and TPRT encrypted files

Figure 5.15 gives the frequency distribution graphs of source file, RSA encrypted file and TPRT encrypted file and figure 5.16 shows the frequency distribution graph of the proposed technique, TMAT. Frequency distribution graph is one of the important cryptographic properties, the more uniform distribution of the characters in the encrypted file

the harder for cryptanalysis. Although ten different types of source files are encrypted but for analysis one such source file is considered and shown in figure 5.15. The source file considered is of text file type, in text file type the characters are present in the range of 0-127 ASCII value. As already seen in figure 5.15 the characters in the source file is concentrated in a particular region, immediate below the frequency distribution graph of RSA encrypted file is given, and finally the frequency distribution graph of TPRT encrypted file is given. Figure 5.16 gives the frequency distribution graph of TMAP encrypted file.



Figure 5.16: Frequency distribution graph of TMAP encrypted file

This result illustrates the frequency of the proposed technique is well distributed than that of RSA; this infers the statistical cryptanalysis may be difficult. It is also seen that the frequency distribution graph is well comparable with that of the previous technique, TPRT. Thus it can be inferred that TMAP showing a good and comparable result in terms of frequency distribution. The frequency distribution of TPRT and TMAP encrypted files are almost same, though got good and comparable result but there is not any subsequent improvement in frequency distribution graph of the proposed technique, TMAP than that of previous technique, TPRT.

5.6.3 The Non-Homogeneity Test

This section compares the extent of non-homogeneity between plaintext/source file and ciphertext/encrypted file through Pearsonian Chi-Square test.

Table 5.6: Chi-Square values and degree of freedom of TMAT, RSA and TPRT

Source File	File Size (Bytes)	Chi-Square Value			Degree of Freedom		
		TMAT	RSA	TPRT	TMAT	RSA	TPRT
license.txt	17,632	201530	30148	191382	255	64	255
cs405(ei).doc	25,422	286025	185351	253470	255	66	255
acread9.txt	35,121	440184	169424	410735	255	73	255
deutsch.txt	47,829	555220	334371	505121	255	77	255
genesis.txt	49,600	659045	396128	638592	255	75	255
pod.exe	69,981	905416	761842	896405	255	76	255
mspaint.exe	136,463	1297256	1053183	1203665	255	88	255
cmd.exe	152,028	1759014	545752	1692655	255	73	255
d3dim.dll	193,189	4630652	307565	4250652	255	10	255
clbcattq.dll	403,901	4167801	327510	3922143	255	11	255

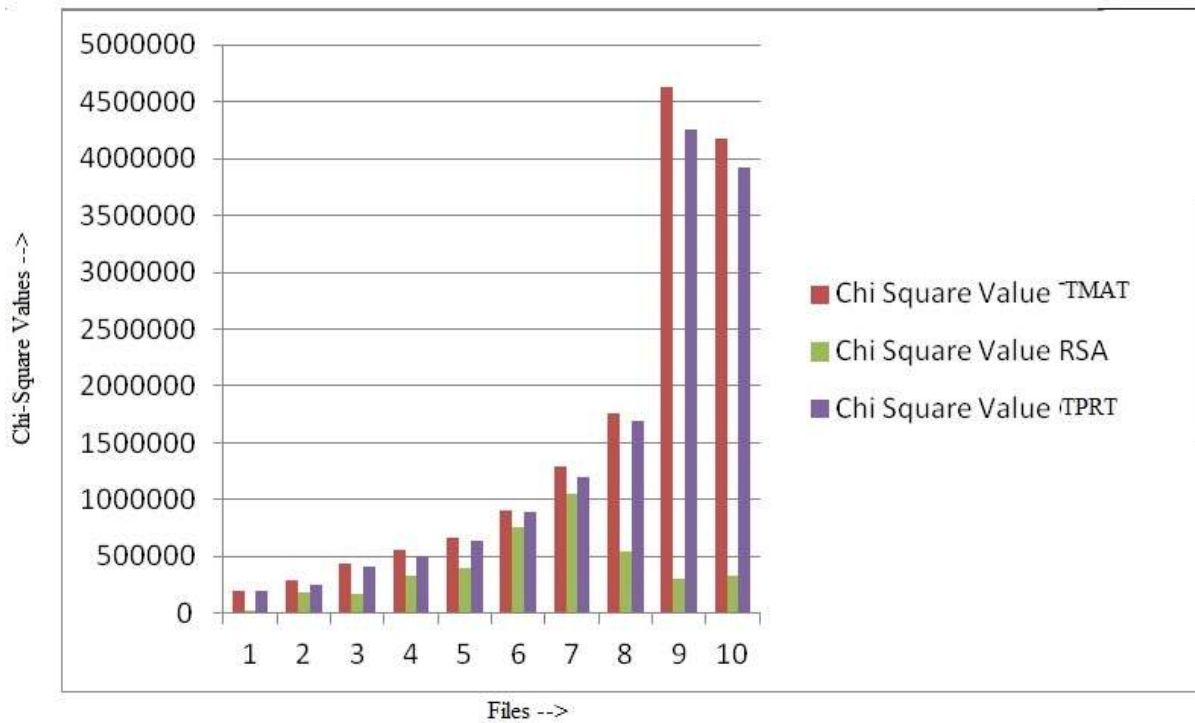


Figure 5.17: Graphical representation Chi-Square value of TMAT, RSA and TPRT

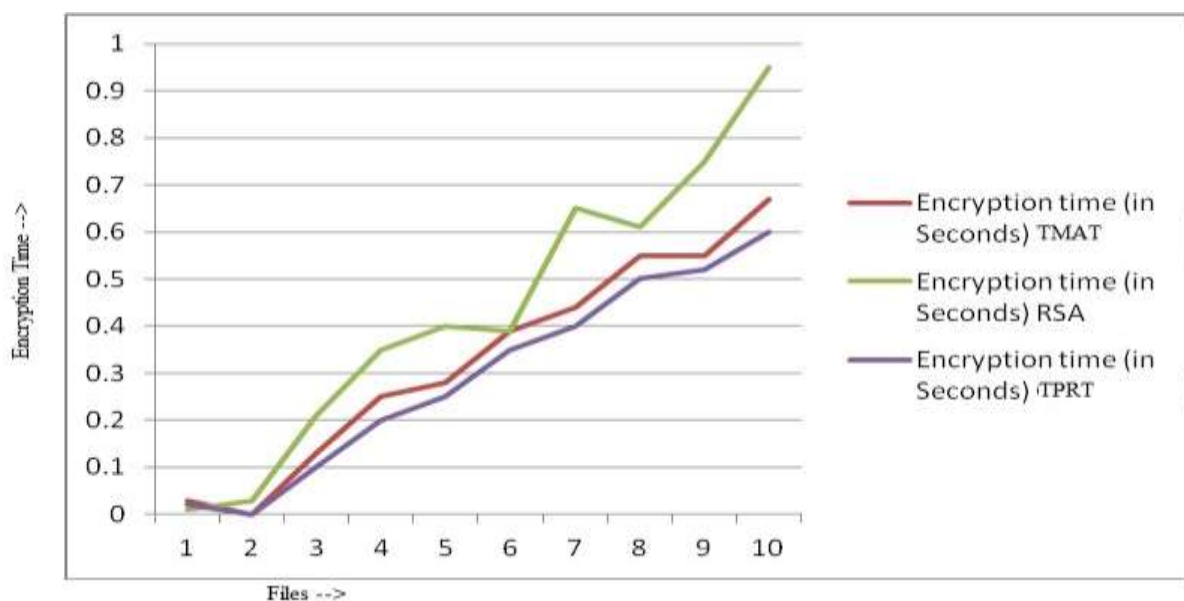
Table 5.6 gives the Chi-Square values of the proposed technique (TMAT), TPRT and that of RSA. Figure 5.17 illustrates the same graphically. The Chi-Square test is another cryptographic parameter in which get the measure of non-homogeneity/heterogeneity between the source file/plaintext and the encrypted file/ciphertext. The Pearsonian Chi-

Square value is considered here. If observing table 5.6, the minimum Chi-Square value of TMAT is 201530, RSA is 30148 and TPRT is 191382. The maximum Chi-Square value of TMAT is 4167801, RSA is 327510 and TPRT is 3922143. Here ten different types of files are encrypted and Chi-Square values are tabulated, text file (TXT), Microsoft word documents (DOC), executable files (EXE) and DLL files are used here for results and analysis.

So observing the above table and figure it has been seen that Chi-Square values of the proposed technique is quite higher than RSA and also the degree of freedom comes to be at a value of 255 in TMAT which says a well distribution of characters present in the TMAT encrypted files than that of source file. It is also been observed that the Chi-Square value of TMAT is quite higher than that of TPRT. Thus it can be seen that this proposed technique, TMAT, produces more heterogeneous files than that of the previous technique, TPRT and RSA. Therefore the Chi-Square values are at par with that of frequency distribution graph result illustrated in section 5.6.2. So, this is the one improvement that got in this chapter.

5.6.4 The Time Complexity Analysis

The main purpose of this chapter is to develop an efficient and fast RTL design so; time complexity analysis is one of the major factors in developing the technique.



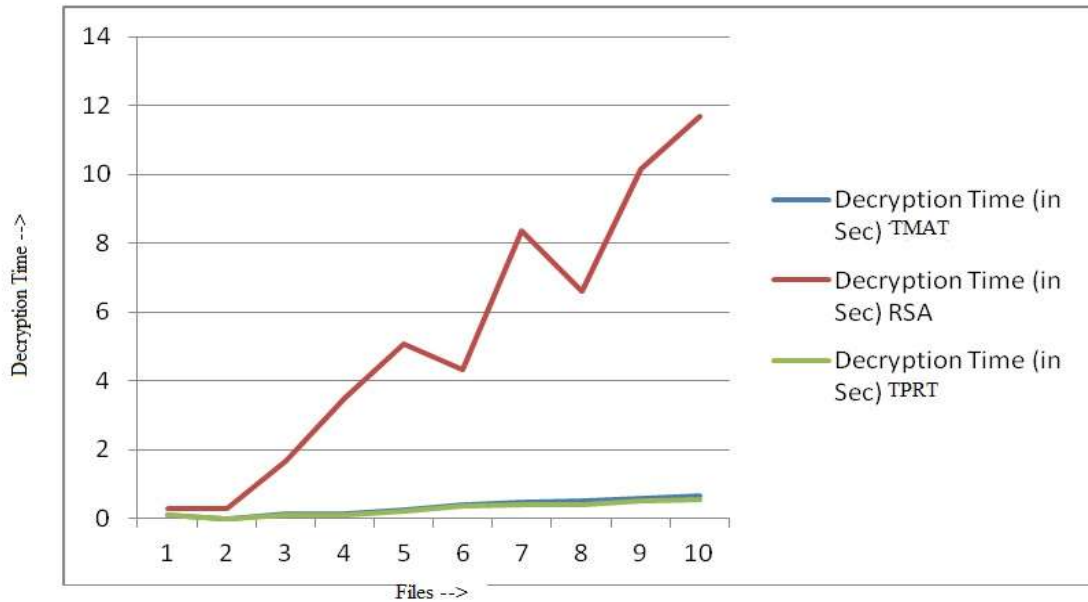


Figure 5.18: Pictorial representation of time complexity analysis of TMAT, RSA and TPRT

Table 5.7: The time complexity analysis of TMAT, RSA and TPRT

Source File	File Size (Bytes)	Encryption time (in Seconds)			Decryption time (in seconds)		
		TMAT	RSA	TPRT	TMAT	RSA	TPRT
license.txt	17,632	0.03	0.01	0.02	0.11	0.28	0.10
cs405(ei).doc	25,422	0.00	0.03	0.00	0.00	0.30	0.00
acread9.txt	35,121	0.13	0.21	0.10	0.13	1.67	0.10
deutsch.txt	47,829	0.25	0.35	0.20	0.15	3.51	0.11
genesis.txt	49,600	0.28	0.40	0.25	0.25	5.06	0.20
pod.exe	69,981	0.39	0.39	0.35	0.39	4.34	0.35
mspaint.exe	136,463	0.44	0.65	0.40	0.48	8.37	0.40
cmd.exe	152,028	0.55	0.61	0.50	0.52	6.59	0.42
d3dim.dll	193,189	0.55	0.75	0.52	0.60	10.15	0.50
clbcataq.dll	403,901	0.67	0.95	0.60	0.65	11.70	0.55

Table 5.7 shows the encryption time and decryption time of the proposed technique, TMAT, previous technique, TPRT, and that of RSA. Figure 5.18 represents the same graphically. The time complexity analysis here given in the unit of file size (in KB) v/s time (in Seconds). If observing the encryption time then it is seen that average encryption time of TMAT is 0.329 seconds, TPRT is 0.294 seconds and RSA is 0.435 seconds. The average decryption time of TMAT is 0.328 seconds, TPRT is 0.273 seconds and RSA is 5.197

seconds. Considering the above results it can be seen that the time complexity of this proposed technique, TMAT is quite better than RSA. The average encryption time of TMAT is less than that of RSA and the average decryption time of TMAT is quite less than that of RSA. So it can be said that the time complexity of the proposed technique is quite less than that of RSA. But it is obvious from the table and graphs that the time complexity of the proposed technique, TMAT is quite higher than that of the previous technique, TPRT. This fact is true because this technique involves extra steps of generating key from the system time and another extra step is there to XOR this time stamp key to the plain text. But, in overall this technique is quite comparable. Therefore it can be concluded that in time complexity analysis though it is better than RSA but there is no significant improvement over TPRT due to algorithmic complexity.

5.6.5 The Avalanche Ratio Test

More the avalanche ratio more difficult to analyses for known plain text – known cipher text pair. The avalanche ratio here obtained by encrypting few bits of source file and then obtaining the percentage of difference between encrypted files of actual source file and modified source files.

Table 5.8: The avalanche ratio of RSA, TPRT and TMAT

Source File	File Size (Bytes)	Avalanche Ratio (in Percentage)		
		RSA	TPRT	TMAT
license.txt	17,632	58.0	77.7	80.8
cs405(ei).doc	25,422	60.0	80.0	85.5
acread9.txt	35,121	75.0	88.8	90.0
deutsch.txt	47,829	78.9	89.0	91.5
genesis.txt	49,600	80.9	87.0	94.7
pod.exe	69,981	58.0	77.0	80.0
mspaint.exe	136,463	58.9	76.0	80.0
cmd.exe	152,028	67.0	77.0	80.0
d3dim.dll	193,189	67.9	82.9	85.0
clbcataq.dll	403,901	68.0	88.5	90.5

Table 5.8 clearly illustrates the result of avalanche ratio, which is found a satisfactory result for the proposed technique, TMAT. If observing clearly then the minimum avalanche ratio of TMAT is 80.0%, TPRT is 77.0% and RSA is 58.0%. The maximum avalanche ratio of TMAT is 94.7%, TPRT is 89.0% and RSA is 80.9%. Here it can be said that at least 80.0% of ciphertext alters and almost 94.7% of ciphertext alters when alter one-bit/byte of source file and apply TMAT. The same analysis can be drawn for TPRT and RSA. Thus TMAT gives far better result than TPRT and RSA. This is another achievement of this chapter.

5.7 Discussions

The technique given here is easily implemented in high-level language and in VHDL. This technique is very easy and it's implemented in FPGA-based systems, the goal of fast execution and strong cryptanalysis requirements are also obtained here. In Chi-Square value analysis that got a much better result, that means this technique, TMAT produces more heterogeneous ciphertext than that of RSA and TPRT. Also in the avalanche ratio analysis this technique, TMAT, giving better result than that of RSA and TPRT. This means that plaintext and keys of this technique is much more related with ciphertext.

So, it can be inferred that improvement impressing of the proposed technique is there in two factors, Chi-Square test and Avalanche ratio test. In hardware implementation and results also got satisfactory results. It uses much less resources than that of RSA but same time giving cryptographically strong results. So, get low power and low area objective with this technique, TMAT. It also giving much faster execution result than that of RSA when comparing the timing simulation results. Moreover this technique can be fabricated in chip to be used in embedded systems. The main goal of the author(s) is to develop an efficient FPGA-based crypto hardware and this chapter is another milestone towards this.

Chapter 6

Recursively Oriented Block Addition and Substitution Technique (ROBAST)

6.1 Introduction

The proposed technique in this chapter termed as, Recursively Oriented Block Addition and Substitution Technique or ROBAST, is a secret-key cryptosystem. In this technique, after decomposing the source stream of bits into a finite number of blocks of finite length, the positions of the bits of each of the blocks is re-oriented using a generating function. For a particular length of block, the block itself is regenerated after a finite number of such iterations. Any of the intermediate blocks during this cycle is considered to be the encrypted block. To decrypt the encrypted block from the ciphertext, the same process is to be followed but the generating function may have to be applied different number of times.

To achieve the security of a satisfactory level, it is proposed that different blocks or blocks should be of different sizes. Accordingly, for different blocks, number of iterations during the encryption and the number of iterations during the decryption also not necessarily should be fixed. This information in a proposed fixed format, described later in this chapter, constitutes the secret key for the system, which is to be transmitted by the sender to the receiver, either with the message or in an isolated manner. The technique does not cause any storage overhead. It provides a large key space, so that the chance of breaking the ciphertext is almost nullified by any technique of cryptanalysis. The implementation on practical scenario is well proven with positive outcome.

TMAT described in Chapter 5 is capable of producing encrypted stream more heterogeneous than TPRT described in Chapter 4. ROBAST described in this chapter is also generator stream which is more heterogeneous than TPRT and TMAT. The HDL synthesis of ROBAST in both netlist and timing summary giving much better result than TPRT and TMAT. Time complexity analysis taking encryption time and decryption time of ROBAST is also giving much better result than TPRT and TMAT. Avalanche test of ROBAST is also better than TPRT and TMAT, that means altering a single/few bits in plaintext/source file or in session key, ROBAST alters many bits in ciphertext/destination file than that of TPRT and TMAT.

Section 6.2 discussed the algorithm of TMAT with a block level diagram, section 6.3 gives a detailed example of encryption and decryption process, section 6.4 discussed the implementation issues with key generation, section 6.5 gives a brief analysis, section 6.6 discussed the results obtained based on implementation and a brief discussions are given in section 6.7.

6.2 The Algorithm of ROBAST

ROBAST is a bit level cipher, the source stream is broken into some finite number of blocks with a fixed block size and then the scheme is applied into these blocks.

The total message can be considered as blocks of bits with different block size like 8, 16, 32, 64, 128 and 256 bits. Figure 6.1 gives the block/flow diagram of the proposed technique, ROBAST. This is a symmetric block cipher so a block of n -bits is considered for encryption. The rules to be followed for generating a cycle are as follows:

- Consider any source stream of a finite number (where $N=2^n$, $n=3$ to 8) and divide it into two equal parts.
- Make the source stream into paired form so that a pair can be used for the operation.
- Make the modulo- 2^n addition between the first and second pair, second and third pair, third and fourth pair of the source stream, to get the first intermediate block.
- The same process is repeated recursively between second and first, third and second, fourth and third of the source stream, to get the next intermediate block.
- The above points are mainly substitute technique and then permutation technique has been performed by orientation of bits based on the session key. Therefore, these resultant blocks of stream can be considered as cipher text.

This process is repeated until the source stream is generated. After a finite number of iterations source stream is regenerated. So, decryption is basically the iteration of the same process. Thus any of the intermediate blocks can be considered as a cipher text, since it is a symmetric block cipher so the same number of iterations that are used during encryption process is required for decryption. This technique gives much better result in terms of Chi-Square value and hardware implementations.

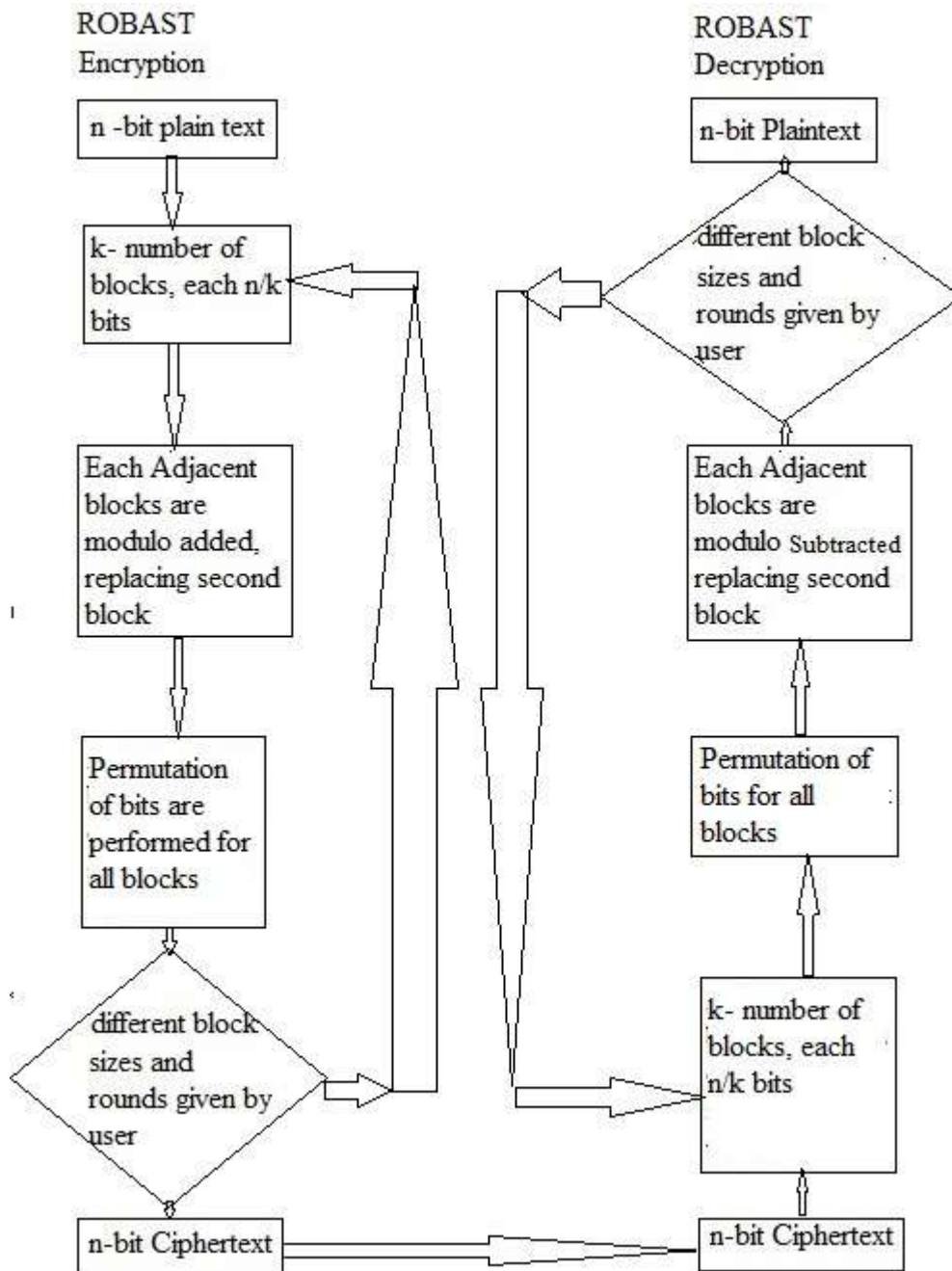


Figure 6.1: Block diagram of ROBAST

In this technique the modulo addition with substitution and permutation is given but to enhance the security further other arithmetic operations can also implemented in this technique. Figure 6.1 gives the block diagram of ROBAST.

6.3 Example

Consider the block $S = 10010011$ of size 8 bits. The Flow diagram to show how positions of the bits of s and the different intermediate blocks can be reoriented with the key values to complete the cycle is shown in figure 6.1 gives the flow/block diagram of ROBAST and table 6.1 illustrates this example in details. In this diagram, each arrow indicates positional orientation of a bit during iteration. Therefore the final cipher text is $S' = 01001001$.

Table 6.1: An encryption example of ROBAST

Source Stream(S) →	10010011			
Blocks of 2-bits	10	01	00	11
Forward 2-bit Modulo 2^2 addition	10	11	11	10
Backward 2-bit Modulo 2^2 addition	10	00	01	10
Orientation of Bits	01	00	10	01
Final Ciphertext (S')	01001001			

Table 6.1 illustrates an encryption example of ROBAST. Let consider the above source stream, S , is divided into blocks of 2-bits each. So, get four blocks, $B_1=10$, $B_2=01$, $B_3=00$ and $B_4=11$. Then forward modulo-4 addition is performed, $B_2=B_1+B_2$, $B_3=B_2+B_3$ and $B_4=B_3+B_4$. Now, get the result as $B_1=10$, $B_2=11$, $B_3=11$ and $B_4=10$, which is shown in the third row of table 6.1. After that backward modulo-4 addition is performed, $B_3=B_3+B_4$, $B_2=B_2+B_3$ and $B_1=B_1+B_2$. Now, get the result as $B_1=10$, $B_2=00$, $B_3=01$ and $B_4=10$, which is shown in the fourth row of table 6.1. Then orientation of the bits are performed that is 1st bit is oriented with 2nd bit of every 2-bit blocks and the result is shown in the fifth row of table 6.1. Finally all the bits are concatenated to get the ciphertext S' .

Table 6.2 illustrates a decryption example of ROBAST. In this decryption example the modulo- 2^n subtraction is performed instead of addition and the steps used in encryption are just reversed for the decryption. Since it is a symmetric cipher so decryption will be the just reverse of encryption.

Let consider the above ciphertext, S' , is divided into blocks of 2-bits each. So, get four blocks, $B_1=01$, $B_2=00$, $B_3=10$ and $B_4=01$. First, orientation of bits is performed, that is

1st bit is oriented with 2nd bit of every 2-bit blocks. The result is given in third row of table 6.2.

Table 6.2: Example of decryption in ROBAST

Source Stream(S')→	01001001			
Blocks of 2-bits	01	00	10	01
Orientation of Bits	10	00	01	10
Backward 2-bit Modulo 2 ² subtraction	10	11	11	10
Forward 2-bit Modulo 2 ² subtraction	10	01	00	11
Final Plaintext (S'')	10010011			

During decoding the backward modulo-4 subtraction is performed, $B_3=B_4-B_3$, $B_2=B_3-B_2$ and $B_1=B_2-B_1$. Now, get the result as $B_1=10$, $B_2=11$, $B_3=11$ and $B_4=10$, which is shown in the fourth row of table 6.2. After that forward modulo-4 subtraction is performed, $B_2=B_2-B_1$, $B_3=B_3-B_2$ and $B_4=B_4-B_3$. The result obtained as $B_1=10$, $B_2=01$, $B_3=00$ and $B_4=11$, which is shown in the fifth row of table 6.2. Finally all the bits are concatenated to get the final plaintext S'', which is the same as the original plaintext S.

6.4 Implementation and Key Generation

The technique executes modulo addition between two blocks, the first iteration performs in forward basis and then backward operation is performed. Next, final permutation is done to get the final cipher text.

This technique has been implemented in C and then feasibility study has been performed. Finally, FPGA based implementation has been done in VHDL. In both implementation, the technique takes input from file as a source stream and encryption is performed. The cipher text generated is finally written in another file. The data blocks (8, 16, 32, 64, 128 and 256-bits) from the input file have been stored in array. Then encryption is performed and also stored in array. The reading and writing of data from and in file is based on 8-bit ASCII codes. Xilinx ISE 8.1i software has been used for writing codes in VHDL.

```

library std;
library ieee;
use ieee.std_logic_arith.all;
use work.pack.all;
use std.textio.all;
use ieee.std_logic_TEXTIO.all;
entity ROBAST_VHDL is
Port ( input_bits : in BIT_VECTOR (16 downto 1);
      output_bits : out
      BIT_VECTOR (16 downto 1); key_bits : in
      BIT_VECTOR (8 downto 1);
      EN_DN : in BIT);
end ROBAST_VHDL;
architecture Behavioral of ROBAST_VHDL is
begin
process(EN_DN)
variable varin_bits,varout_bits: bit_vector(16 downto
1);
begin
if (EN_DN='1')then varin_bits:=input_bits;
AA:
ROBAST_Encryption(varin_bits,key_bits,varout_bits)
;
output_bits<=varout_bits;
else
BB:
ROBAST_Decryption(varin_bits,key_bits,varout_bits)
;
output_bits<=varout_bits;
end if;
end process;
end Behavioral;

```

Figure 6.2: ROBAST entity and its function

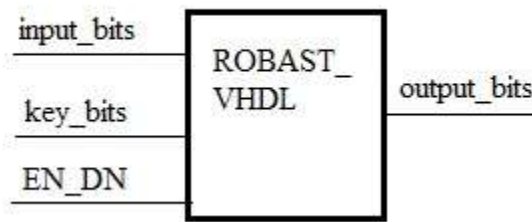


Figure 6.3: Top level RTL design of ROBAST

Figure 6.2 gives the implementation of ROBAST entity and its function. The encryption/decryption entity input bit vector (16-bit), output bit vector (16-bit), key bit vector (8-bit) and EN_DN signal. If EN_DN = 1 then encryption is performed else decryption is performed. During encryption the input bit vector of 16-bits is the plaintext and output 16-bit vector is the ciphertext where as EN_DN value is '1'. During decryption the input bit vector of 16-bits is the ciphertext and the output 16-bit vector is the plaintext where as EN_DN value is '0'. Figure 6.3 shows the top RTL diagram of ROBAST.

When EN_DN = 1, the 'ROBAST_Encryption' function is called with the parameters, 'varin_bits' which is the plaintext, 'varout_bits' which is the ciphertext, both of these are of 16-bits and third parameter is the 'key_bits' which is the session key of the encryption of 8-bits. When EN_DN = 0, the 'ROBAST_Decryption' function is called with the parameters, 'varin_bits' which is the ciphertext, 'varout_bits' which is the plaintext, both of these are of 16-bits and third parameter is the 'key_bits' which is the session key of the decryption of 8-bits. This code is written in VHDL using behavioral model of coding. The 'ROBAST_VHDL' entity in this coding has three ports, 'input_bits' of IN type of bit vector of 16-bits, 'output_bits' of OUT type of bit vector of 16-bits, 'key_bits' of IN type of bit vector of 16-bits and 'EN_DN' bit of IN type. 'Behavioral' is the architecture of the entity 'ROBAST_VHDL', this architecture contains a process which is called on the signal 'EN_DN' that is whenever there is a signal in 'EN_DN' this process is called. This process contains two functions, 'ROBAST_Encryption' and 'ROBAST_Decryption'. These two functions are called according to the value of signal bit 'EN_DN' which is already discussed. The implementation here is both functional and files type. These means that the code can be implemented in Xilinx FPGA and the simulation takes the input from a text file and the output is written into another text file. There are various libraries are used, library 'std' and library 'ieee', it is important to note that library 'ieee.std_logic_TEXTIO.all' is used for the

implementation of text file reading and writing. Figure 6.2 gives the main ROBAST entity coded in VHDL.

Section 6.4.1 deals with the key generation process, section 6.4.2 illustrates an example and section 6.4.3 gives the concept of modulo addition.

6.4.1 The Key Generation Process of ROBAST

In this section key generation process has been illustrated, the session key is 128-bits for generalized ROBAST implementation.

Table 6.3: Representation of number of iterations in each round by bits, the key generation for ROBAST

Round	Block Size	Number of Iterations	
		Decimal	Binary
8.	256	50021	1100001101100101
7.	128	49870	1100001011001110
6.	64	48950	1011111100110110
5.	32	44443	1010110110011011
4.	16	46250	1011010010101010
3.	8	4321	0001000011100001
2.	4	690	0000001010110010
1.	2	72	0000000001001000

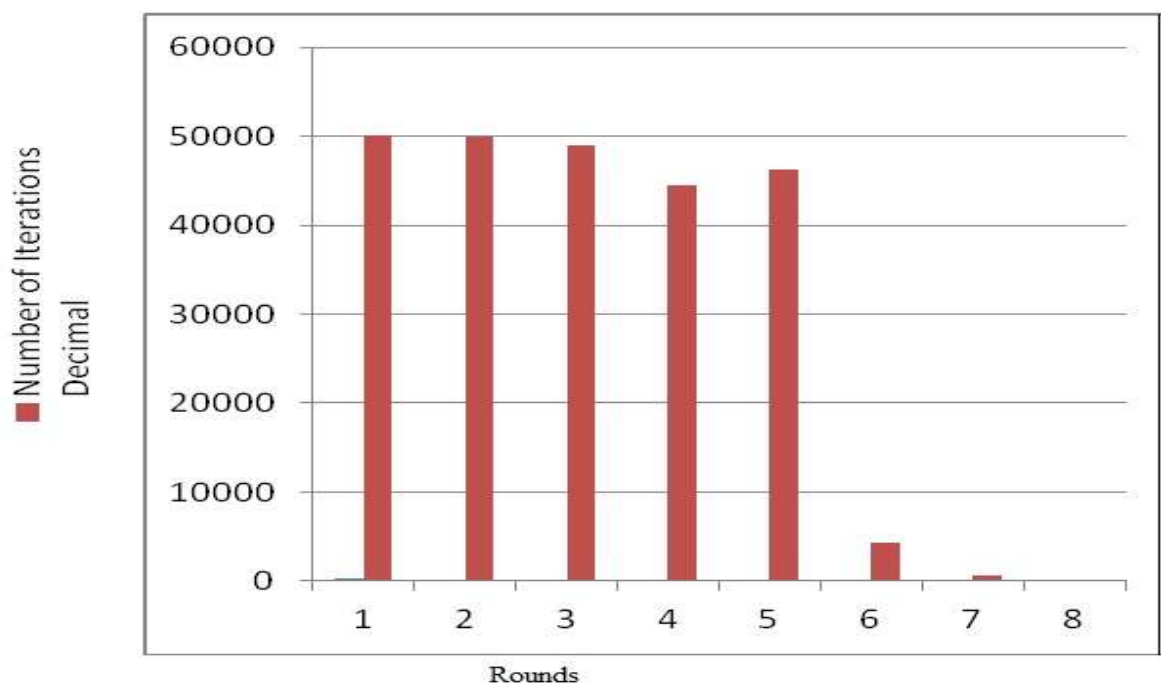


Figure 6.4: Graphical representation of key generation of ROBAST

The key generation process depends on block size, iteration of each block and final permutation performed. Thus, in the proposed scheme, eight rounds have been considered, each for 2, 4, 8, 16, 32, 64, 128, and 256 block size. As mentioned in each round is repeated for a finite number of times and the number of iterations will form a part of the encryption-key. Although the key may be formed in many ways, for the sake of brevity it is proposed to represent the number of iterations in each round by a 16-bit binary string. The binary strings are then concatenated to form a 128-bit key for a particular key. Table 6.3 gives the key generation process and the same is shown graphically in figure 6.4. For the block size of 2-bits are considering 72 rounds, for block size of 4-bits are considering 690 rounds and so on and finally for block size of 256-bits 50021 rounds have been considered for encryption. Since the technique is symmetric block cipher so for decryption same number of rounds will be required. These numbers of rounds have been considered in binary value, for each block size the number of rounds is considered in 16-bits of binary value. So there is eight block sizes and their corresponding eight 16-bits rounds, the key is formed by concatenating all the 16-bits binary values. Therefore, the size of the session key proposed here is $16 \times 8 = 128$ -bits, which is now a day's considered the secure key length.

An example of key generation is illustrated in section 6.4.2. Section 6.4.3 describes the modulo addition used in ROBAST, which is an important operation in the technique and should be taken into account while forming the session key.

6.4.2 An Example of Key Generation

Consider a particular session where the source file is encrypted using iterations for block sizes 2, 4, 8, 16, 32, 64, 128, and 256 bits, respectively. Table 6.3 shows the corresponding binary value for the number of iterations in each round. If considering the block size of 256-bits then the binary value of round is '1100001101100101', for block size of 128-bit the binary value of round is '1100001011001110' and so on finally for block size of 2-bits the binary value of round is '0000000001001000'. These eight 16-bits binary strings are concatenated together to form the 128-bit binary string, which is given below.

- 110000110110010111000010110011101011111100110110101011011001101110110
10010101010000100001110000100000010101100100000000001001000

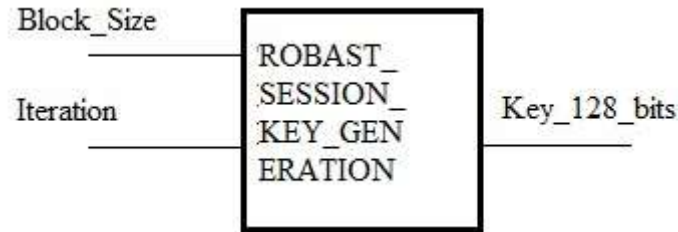


Figure 6.5: Session key generation of ROBAST

This 128-bit binary string will be the encryption-key for this particular session. During decryption, the same key is taken to iterate each round of modulo-subtraction for the specified number of times and reverse permutation. Figure 6.5 shows the top level RTL diagram of session key generation of ROBAST.

6.4.3 Modulo Addition Used in ROBAST

An alternative method for modulo addition is proposed here to make the calculations simple. The need for computation of decimal equivalents of the blocks is avoided here since it will get large decimal integer values for large binary blocks. The method proposed here is just to discard the carry out of the MSB after the addition to get the result. For example, if add 1101 and 1001 it get 10110. In terms of decimal values, $13+9=22$. Since the modulus of addition is 16 (2^4) in this case, the result of addition should be 6 ($22-16=6$). Discarding the carry from 10110 is equivalent to subtracting 10000 (i.e. 16 in decimal). So the result will be 0110, which is equivalent to 6 in decimal. The same is applicable to any block size.

6.5 Analysis

Analyzing all the results presented in section 6.6, following are the points obtained on the proposed technique, ROBAST:

- The algorithmic complexity of ROBAST is $O(n^2)$.
- ROBAST encrypts block of fixed sizes of 2^n , where $n = \{ 3, 4, 5, \dots \}$, that is 8, 16, 32, 64, 128 and 256.
- ROBAST is a substitution cipher, where the modulo addition between two consecutive blocks replaces the second block.

- ROBAST is a recursive cipher, the encryption block size starts with 256-bits and goes up-to 8-bit block size.
- Decryption is same as encryption where round keys are given in reverse manner in decryption than that of encryption.

6.6 Results and Simulations

Any cryptographic technique is to be accepted, a satisfactory results are very much required. This technique has been tested for feasibility both in terms of algorithmic parameters and cryptographic parameters. These are all described in respective sub sections. Section 6.6.1 discuss results of RTL/Hardware implementation, section 6.6.2 discuss the results of frequency distribution graph, section 6.6.3 discuss the results of Chi-Square test for non-homogeneity of source files and encrypted files, section 6.6.4 discuss the results of time complexity and section 6.6.5 discuss the results of avalanche ratio test.

6.6.1 RTL Simulation Based Result

In this section gives some of the results found after implementing the proposed technique in VHDL. This code has been simulated and synthesized in Xilinx 8.1i. The main objective is to find an efficient FPGA-based cryptographic technique for implementation in embedded systems.

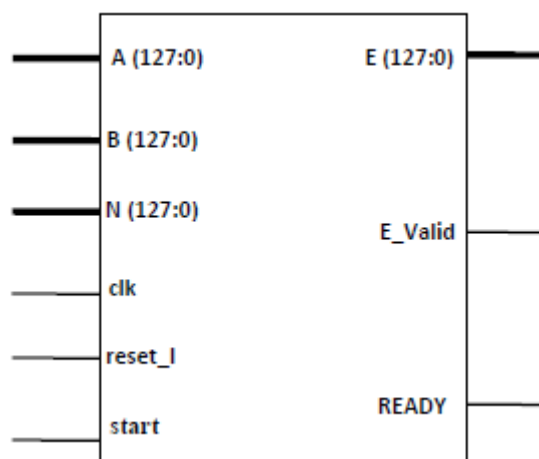


Figure 6.6: RTL diagram of RSA

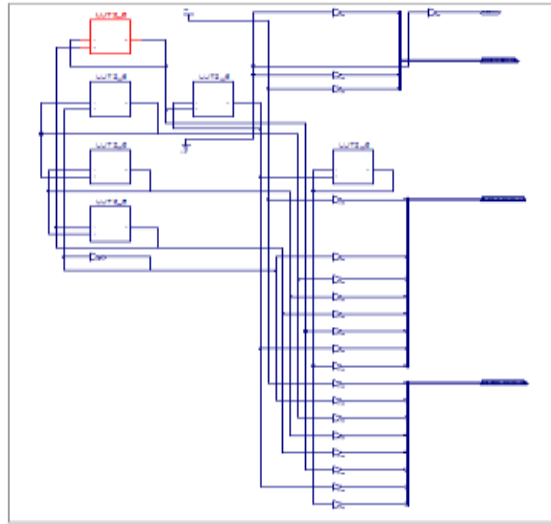


Figure 6.7: Spartan 3E RTL diagram of TPRT

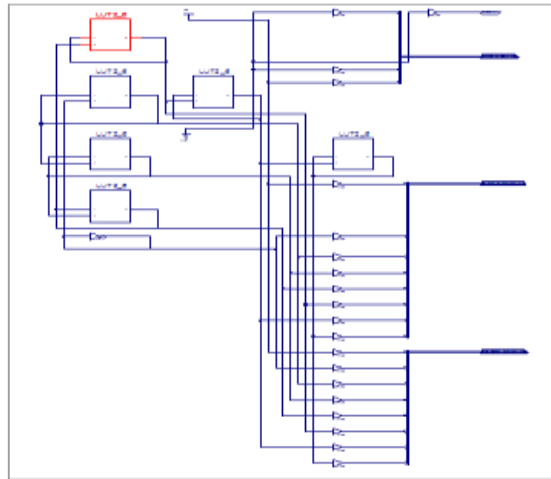


Figure 6.8: Spartan 3E RTL diagram of TMAP

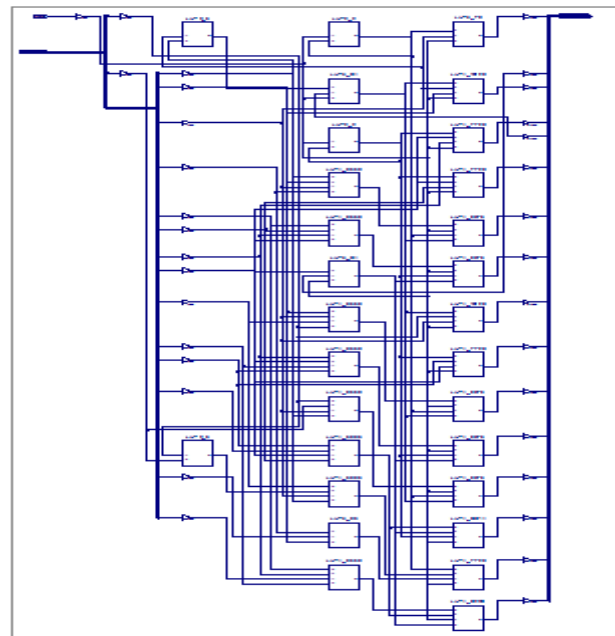


Figure 6.9: Spartan 3E schematic of ROBAST

Figure 6.6 shows the RTL schematic of RSA, figure 6.7 shows the RTL schematic of TPRT, figure 6.8 shows the RTL schematic of TMAT and figure 6.9 shows the RTL schematic of ROBAST. If observe the figures given above that it can be seen that a lot of Look-Up-Tables are required, almost thirty, to realize this proposed technique, ROBAST, in Spartan 3E FPGA. Similarly by seeing RTL schematic it can be said that a lot of registers are required, almost fifty, to realize this proposed technique, ROBAST. So, this technique uses the resources efficiently, the netlist study and speed grade study is discussed in later paragraphs.

Table 6.4 gives the netlist generation of proposed technique, ROBAST, previous techniques, TPRT, TMAT and RSA. RAMs/ROMs used in ROBAST is quite more than that of TPRT and TMAT but still less than RSA. Since modulo addition is the backbone of ROBAST, so, ROBAST uses quite large number of adder/subtraction than that of TPRT, TMAT and RSA. Registers used here, ROBAST, is also larger than that of TPRT and TMAT but it is still less than that of RSA. This technique, ROBAST, also uses a quite number of latches and multiplexers than TPRT and TMAT. So, in overall it can be said that ROBAST uses the resources available in FPGA chip.

Table 6.4: HDL synthesis report (Netlist generation of RSA, TPRT, TMAT and ROBAST)

Sr No.	Netlist Components	Number			
		RSA	TPRT	TMAT	ROBAST
1	ROMs/RAMs	430	10	14	25
2	Adders/Subtractions	3	0	2	20
3	Registers	420	20	30	50
4	Latches	80	0	0	10
5	Multiplexers	120	0	0	10

Table 6.5 gives the timing summary of proposed technique, ROBAST, previous techniques, TPRT and TMAT and RSA. The minimum period for ROBAST is less than TPRT, TMAT and RSA. The maximum frequency is same for all the implementation because all the implementation is based of Spartan 3E FPGA with a speed grade of -5. Minimum input arrival time of ROBAST is of medium value than TPRT, TMAT and RSA.

Table 6.5: HDL synthesis report (Timing summary of RSA, TPRT, TMAT and ROBAST)

Sr No.	Timing Constraint	Values			
		RSA	TPRT	TMAT	ROBAST
1	Speed Grade	-5	-5	-5	-5
2	Minimum period (ns)	9.895	5.66	7.95	5.55
3	Maximum Frequency (MHZ)	101.06	101.06	101.06	101.06
4	Minimum input arrival time before clock (ns)	6.697	4.33	5.55	5.55
5	Maximum output required time after clock (ns)	4.31	3.33	4.25	4.44

The maximum output required time is higher for proposed technique, ROBAST, than that of TPRT, TMAT and RSA. So, in overall it can be said that ROBAST is giving satisfactory result in hardware implementation.

6.6.2 The Frequency Distribution Graph

The frequency distribution is the distribution of the all 256 ASCII characters in the respective files. This is also a cryptographic parameter which measures the degree of cryptanalysis. Figure 6.10 illustrates the source file, RSA encrypted file and TPRT encrypted file frequency distribution results found after implementation of respective algorithms/techniques. Figure 6.11 illustrates the frequency distribution of TMAT and ROBAST encrypted file. The frequency distribution graph of all the proposed techniques, ROBAST, TPRT and TMAT are giving the optimal result. All the frequencies are evenly distributed over 256 region for all the technique except that of RSA where the frequency distribution is not evenly distributed and somewhat resembles frequency distribution of a text file. Though ten files have been encrypted but for this result the file 'genesis.txt' of size 48.44 KB is considered.

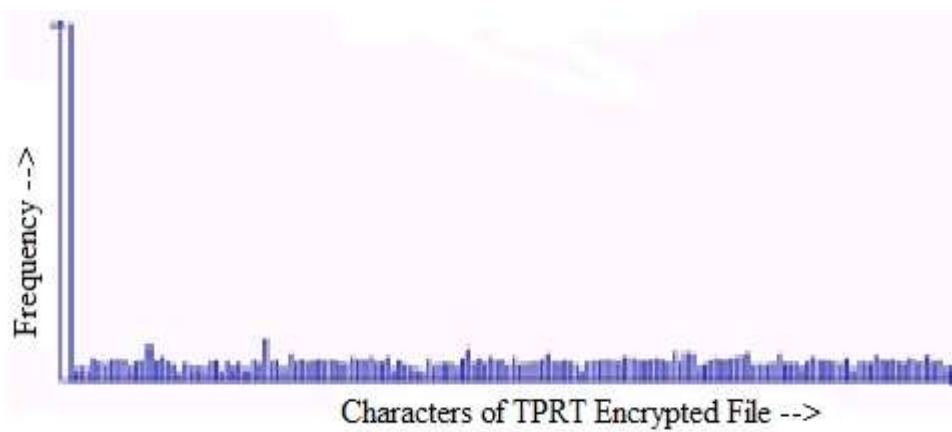
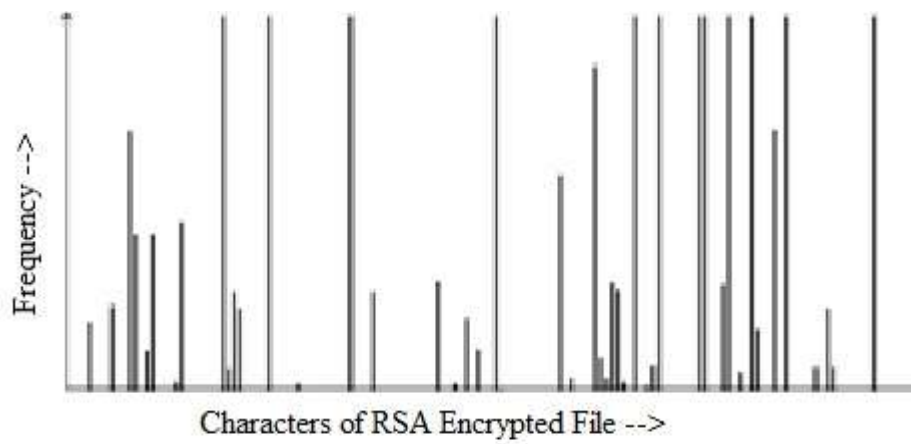
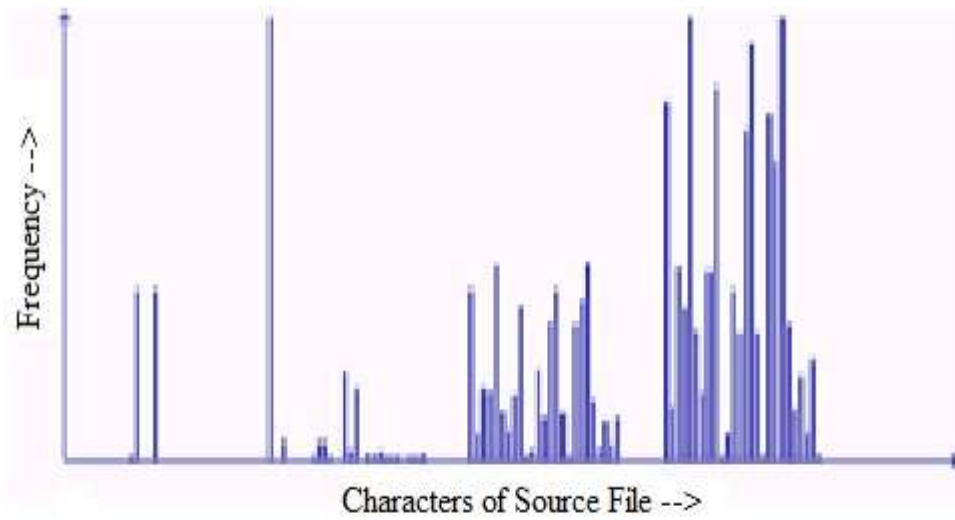


Figure 6.10: Frequency distribution graph of source, RSA encrypted and TPRT encrypted files

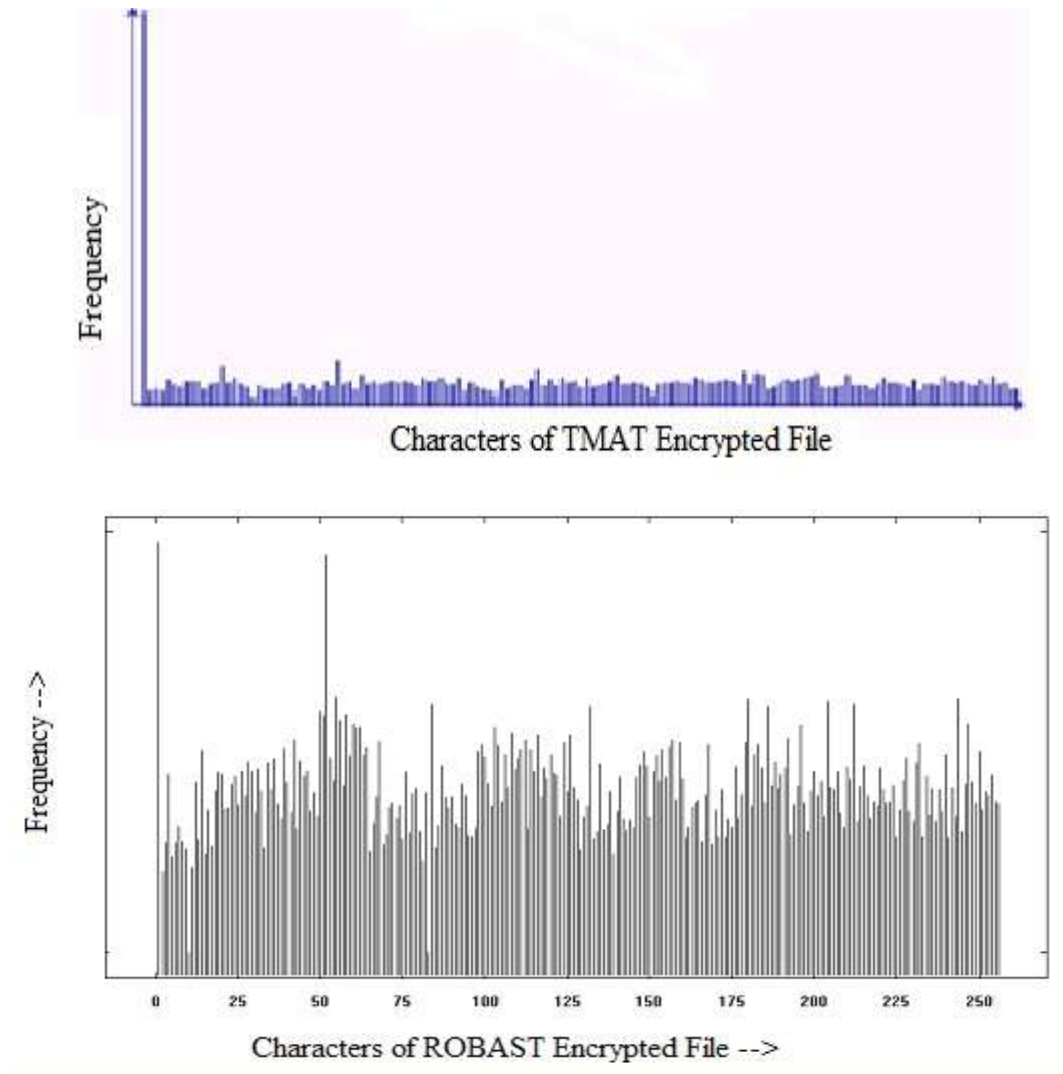


Figure 6.11: Frequency distribution graph of TMAT and ROBAST encrypted files

6.6.3 The Non-Homogeneity Test

Test for non homogeneity has been done using Chi-Square value and degree of freedom; this is one of the important cryptographic parameters. Chi-Square value is the statistical value between source file and encrypted files, which gives the difference. Degree of freedom in the character distribution of the above said files. Table 6.6 gives the Chi-Square value and figure 6.12 illustrates the same graphically.

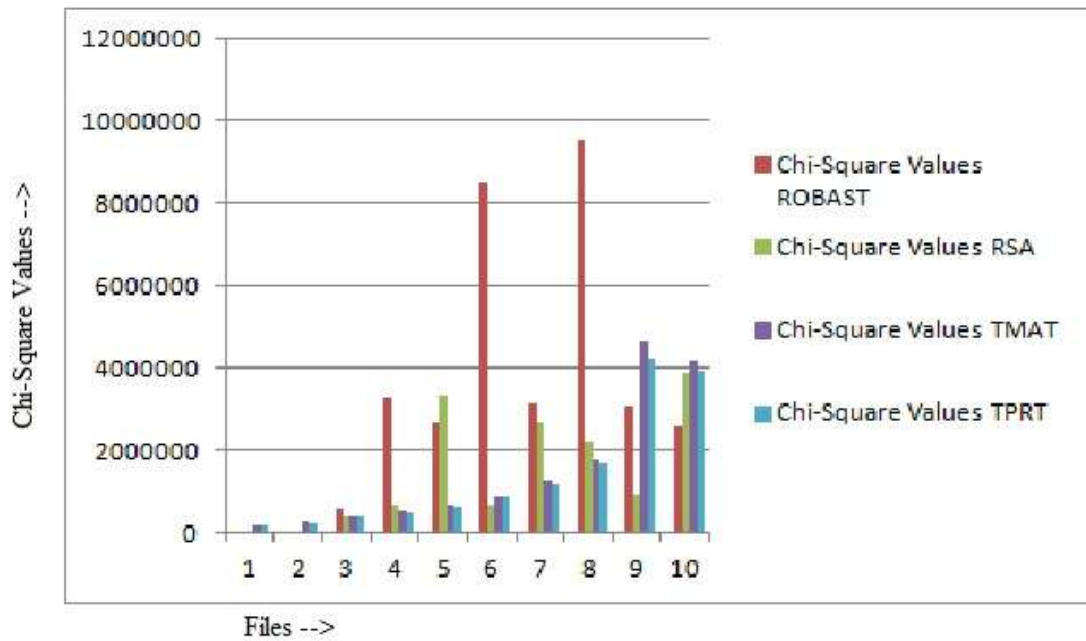


Figure 6.12: Pictorial representation of Chi-Square values of ROAST, RSA, TPRT and TMAT

Table 6.6: Chi-Square values of ROAST, RSA, TPRT and TMAT

Source File	File Size (Bytes)	Chi-Square Values			
		ROAST	RSA	TMAT	TPRT
license.txt	17,632	6472	5668	201530	191382
cs405(ei).doc	25,422	4407	2654	286025	253470
acread9.txt	35,121	560357	447984	440184	410735
deutsch.txt	47,829	3307374	685963	555220	505121
genesis.txt	49,600	2679799	3318506	659045	638592
pod.exe	69,981	8495675	694410	905416	896405
mspaint.exe	136,463	3131296	2667664	1297256	1203665
cmd.exe	152,028	9559993	2216429	1759014	1692655
d3dim.dll	193,189	3102369	906300	4630652	4250652
clbcattq.dll	403,901	2590855	3896171	4167801	3922143

Table 6.7: Degree of freedom of ROBAST, RSA, TPRT and TMAT

Source File	File Size (Bytes)	Degree of Freedom			
		ROBAST	RSA	TMAT	TPRT
license.txt	17,632	253	253	255	255
cs405(ei).doc	25,422	253	253	255	255
acread9.txt	35,121	253	253	255	255
deutsch.txt	47,829	253	253	255	255
genesis.txt	49,600	253	253	255	255
pod.exe	69,981	253	253	255	255
mspaint.exe	136,463	254	254	255	255
cmd.exe	152,028	253	253	255	255
d3dim.dll	193,189	253	253	255	255
clbcatq.dll	403,901	253	253	255	255

From the table it is seen that the average Chi-Square value of ROBAST is 33,43,860, RSA is 14,84,175, TMAT is 14,90,214 and TPRT is 13,96,482. Therefore it can be said that ROBAST is giving the optimal solution for non-homogeneity test but in degree of freedom TMAT and TPRT are giving better result than ROBAST. Figure 6.12 giving the Chi-Square values graphically, X-axis is the ten files and Y-axis is the corresponding Chi-Square values, here bar graph is selected for this result.

6.6.4 The Time Complexity Analysis

Time complexity is based on encryption time and decryption time. Encryption time is the time required to encrypt a source file and decryption time is the time to decrypt the cipher text file to get the original file. Table 6.8 gives the encryption time complexities and figure 6.13 illustrates the same. Table 6.9 gives the decryption time complexities and figure 6.14 illustrates the same. This test is in terms of efficient algorithmic parameter.

Table 6.8: Comparison of encryption time of ROBAST, RSA, TMAT and TPRT

Source File	File Size (Bytes)	Encryption Time			
		ROBAST	RSA	TMAT	TPRT
license.txt	17,632	0.00	0.01	0.03	0.02
cs405(ei).doc	25,422	0.01	0.06	0.00	0.00
acread9.txt	35,121	0.02	0.07	0.13	0.10
deutsch.txt	47,829	0.03	0.11	0.25	0.20
genesis.txt	49,600	0.04	0.12	0.28	0.25
pod.exe	69,981	0.04	0.12	0.39	0.35
mspaint.exe	136,463	0.06	0.20	0.44	0.40
cmd.exe	152,028	0.07	0.25	0.55	0.50
d3dim.dll	193,189	0.08	0.28	0.55	0.52
clbcattq.dll	403,901	0.08	0.32	0.67	0.60

Table 6.9: Comparison of decryption time of ROBAST, RSA, TMAT and TPRT

Source File	File Size (Bytes)	Decryption Time			
		ROBAST	RSA	TMAT	TPRT
license.txt	17,632	0.01	0.15	0.11	0.10
cs405(ei).doc	25,422	0.02	0.71	0.00	0.00
acread9.txt	35,121	0.03	1.15	0.13	0.10
deutsch.txt	47,829	0.03	1.36	0.15	0.11
genesis.txt	49,600	0.04	1.61	0.25	0.20
pod.exe	69,981	0.04	1.86	0.39	0.35
mspaint.exe	136,463	0.05	2.71	0.48	0.40
cmd.exe	152,028	0.06	3.34	0.52	0.42
d3dim.dll	193,189	0.07	3.73	0.60	0.50
clbcattq.dll	403,901	0.08	4.25	0.65	0.55

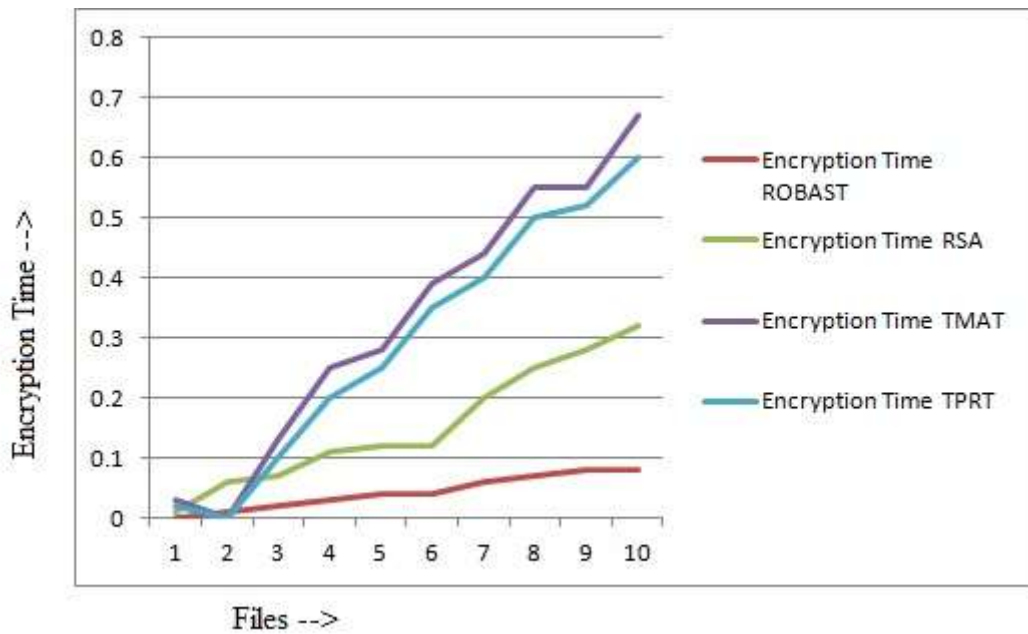


Figure 6.13: Pictorial representation of encryption time of ROBAST, RSA, TMAT and TPRT

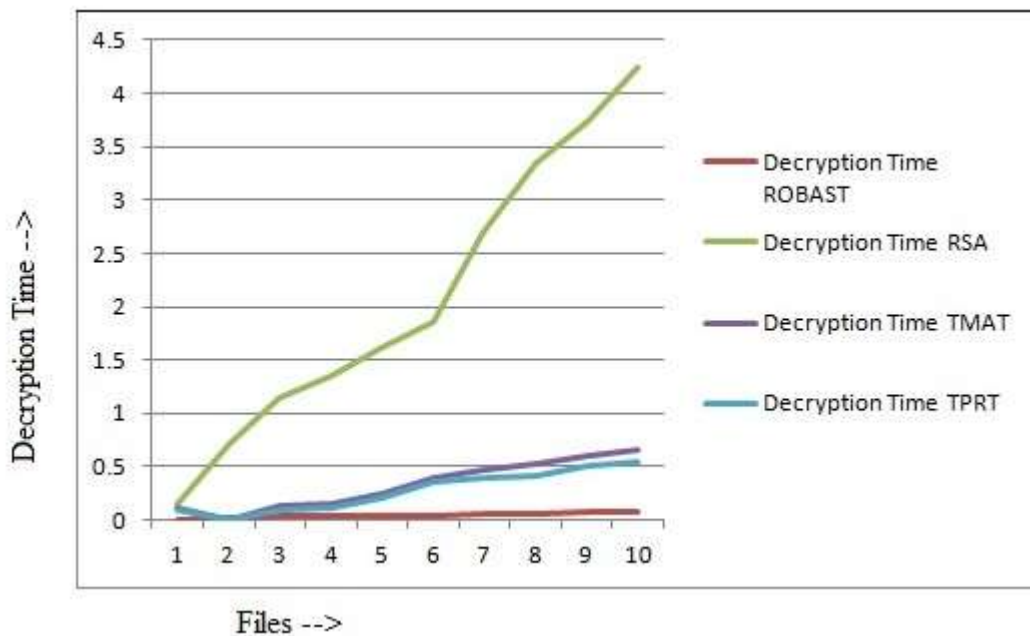


Figure 6.14: Pictorial representation of decryption time of ROBAST, RSA, TMAT and TPRT

Encryption time in table 6.8, the encryption time is given against file size in KB, and considering all the ten files then a total 2.36 MB of file is encrypted. The total time of encryption by ROBAST is 0.43 seconds, RSA is 1.54 seconds, TMAT is 3.29 seconds and TPRT is 2.94 seconds. So, ROBAST is giving the optimal result for encryption time

complexity analysis. Figure 6.13 giving these results graphically, X-axis is the ten files and Y-axis is the corresponding encryption time, here line graph is selected for the result.

Decryption time in table 6.9, the decryption time is given against file size in KB, and considering all the ten files then a total 2.36 MB of file is decrypted. The total time of decryption by ROBAST is 0.43 seconds, RSA is 20.87 seconds, TMAT is 3.28 seconds and TPRT is 2.73 seconds. So, ROBAST is giving the optimal result for decryption time complexity analysis and far better than that of RSA. Figure 6.14 giving these results graphically, X-axis is the ten files and Y-axis is the corresponding decryption time, here line graph is selected for the result.

6.6.5 The Avalanche Ratio Test

The avalanche ratio has been obtained by modifying 2-3 bits/bytes in the encryption key as well as in source files. It's a strong cryptographic parameter and this may be conceptualize with the avalanche occurs in hill area. Table 6.10 gives the avalanche ratio values of ROBAST, RSA, TPRT and TMAT.

Table 6.10: Comparison of avalanche ratio of ROBAST, RSA, TPRT and TMAT encrypted files

Source File	File Size (Bytes)	Avalanche Ratio of ROBAST encrypted files (in %)	Avalanche Ratio of RSA encrypted files (in %)	Avalanche Ratio of TPRT encrypted file (in %)	Avalanche ratio of TMAT encrypted file (in %)
license.txt	17,632	71.90	58.0	77.7	80.8
cs405(ei).doc	25,422	99.69	60.0	80.0	85.5
acread9.txt	35,121	99.93	75.0	88.8	90.0
deutsch.txt	47,829	99.96	78.9	89.0	91.5
genesis.txt	49,600	97.72	80.9	87.0	94.7
pod.exe	69,981	77.00	58.0	77.0	80.0
mspaint.exe	136,463	98.22	58.9	76.0	80.0
cmd.exe	152,028	99.97	67.0	77.0	80.0
d3dim.dll	193,189	99.98	67.9	82.9	85.0
clbcqtq.dll	403,901	75.55	68.0	88.5	90.5

The average avalanche ratio of ROBAST is 91.99%, RSA is 67.26%, TPRT is 82.39% and TMAT is 85.8%. So ROBAST is again giving optimal solution for avalanche ratio analysis that means modifying of plaintext/key will greatly affect ciphertext than other techniques/algorithm, RSA, TPRT and TMAT.

6.7 Discussions

The technique given here is easily implemented in high-level language and in VHDL. This technique is very easy and it's implemented in FPGA-based systems, the goal of fast execution and strong cryptanalysis requirements are also obtained here. Moreover this technique can be fabricated in chip to be used in embedded systems. The main goal is to develop an efficient FPGA-based crypto hardware and this proposed technique is another step towards this. Now in the next two chapters are develop technique which will give much better result in terms of confusion and diffusion. It is also be seen that the avalanche ratio analysis will be boosted in the subsequent chapters.

Chapter 7

Shuffle – Rotational Addition Technique (SRAT)

7.1 Introduction

In this chapter, another secret-key cryptographic system is proposed. The proposed system is a bit-level implementation. It is a block cipher and it follows the principle of substitution and permutation. The techniques of the encryption and the decryption are not the same but one can easily be derived from the other. During encryption modulo addition and butterfly shuffle has been done and during decryption modulo subtraction and butterfly shuffle has been done. The encryption key and decryption keys are same.

During encryption, a butterfly shuffle is applied to the whole source stream, then the source stream is broken down into blocks of fixed size, then the consecutive blocks are modulo added, the result replaces the second block keeping the first block intact, in the next phase the whole block is left circular rotated. Now the blocks are concatenated and again another round of butterfly shuffle is applied. The block size and number of modulo addition to be performed depends on round key K1 and the butterfly shuffle depends upon round key K2, thus the RAT operation is sandwiched between two butterfly shuffles.

During decryption, at first butterfly shuffle is done on source stream, then the source stream is broken down into number of fixed size blocks, the consecutive blocks are modulo subtracted the result replacing the second block keeping the first block intact, then here right circular rotation is performed. Then the blocks are merged and another round of butterfly shuffle is applied.

The system does not cause any storage overhead, the execution time changes almost linearly with the size of the file being encrypted. The result of the Chi-Square test establishes the fact that the source file and the encrypted file are non-homogeneous. The frequency distribution test between the source and the encrypted files shows how the encrypted characters are well distributed. The comparison of this proposed technique with the RSA system on the basis of the Chi-Square values establishes the success of the technique in ensuring the security of highly satisfactory level. Encryption and decryption time analysis has also been done and this technique is giving satisfactory result. Another important cryptographic parameter is avalanche test and this technique is giving much better result. These entire tests have been performed by implementing the techniques in C-programming language. Moreover, this technique has also been implemented in VHDL for FPGA-based systems and the results found there after is also better than previous techniques.

Section 7.2 discussed the algorithm of SRAT with a block level diagram, section 7.3 gives a detailed example of encryption and decryption process, section 7.4 discussed the

implementation issues with key generation, section 7.5 gives a brief analysis, section 7.6 discussed the results obtained based on implementation and a brief discussions are given in section 7.7.

7.2 The Algorithm of SRAT

This section describes the algorithm of SRAT with a block level diagram. The plaintext for Shuffle-RAT is considered as a stream of 512 bit blocks. Figure 7.1 shows the block diagram of Shuffle-RAT. The basic round function is Rotational Addition Technique (RAT), applied on the 512-bit plaintext over 8 rounds where RAT is sandwiched between two Butterfly-Shuffles.

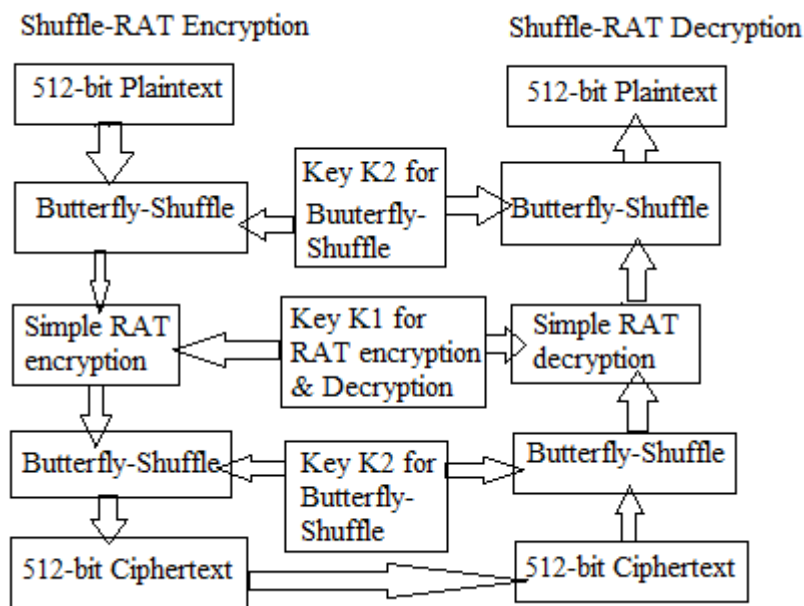


Figure 7.1: Block diagram of Shuffle-RAT

The plaintext is subdivided into smaller blocks in each round of Shuffle-RAT, where the block sizes vary with the powers of 2 in the rounds, i.e., 2^n -bit blocks are considered for round 'n', where 'n = 1, 2, 3 ... 8'. In the 'n-th' round of Shuffle-RAT, the rotational addition adds each block to the adjacent block modulo 2^n , and stores the result in the second block, iteratively over the length of the plaintext, the operation of RAT is in between two Butterfly-Shuffle. In mathematical terms, the round function of Shuffle-RAT is as follows.

One Round of Butterfly-Shuffle (1)

$$B_{i+1} = (B_i + B_{i+1}) \bmod 2^n \quad (2)$$

Another Round of Butterfly-Shuffle (3)

In equation (2), the index 'i' cover all the blocks in each round. Each round of Shuffle-RAT is iterated for some number of times defined by 'keys', where the round-keys are of size 16 bits each. Thus, the total key-size of Shuffle-RAT is $8 \times 16 = 128$ bits. Decryption for Shuffle-RAT is just the opposite of encryption, where one has to use modular subtraction instead of addition and the round-keys are considered in the reverse order.

A close study of Shuffle-RAT reveals a few areas for improving the design even further. The degree of randomness may be increased for better non-homogeneity and security than the previous scheme. In terms of improving the algorithm, it was observe that RAT, the previous proposed technique, has a strong property of 'confusion', like all good block ciphers, but lacks good 'diffusion'. Thus, I propose the diffusion of Shuffle-RAT with the technique of butterfly shuffle to produce a new cipher – Shuffle-RAT, with high confusion and diffusion. This Butterfly-Shuffle produces high 'diffusion' which is performed twice in this technique, before and after each round of RAT, and RAT here provides good 'confusion'. Therefore this proposed cipher, Shuffle-RAT (SRAT), provides high confusion and diffusion properties.

The algorithm of the Shuffle-RAT technique is based on RAT [131], and can be summarized as follows:

- Step 1: The 512 bit message is divided into a number of blocks; each containing $N = 2^n$ bits, where N is any one of 2, 4, 8, 16, 32, 64, 128, 256.
- Step 2: Each round key of 16 bits, produced similar to that in RAT, is divided into two parts each of 8 bits. Suppose that they are named as key1 and key2.
- Step 3: First, Butterfly-Shuffle of bits/blocks is performed, which is based on key2.
- Step 4: Two adjacent blocks are added and result is stored in the 2^{nd} block where the modulus of addition is 2^n as in the case of RAT. This RAT operation is performed and which is based on Key1. Thus,

original RAT is performed on the blocks of size N for (key1) times of iterations.

- Step 5: The blocks are shuffled within the message to create proper diffusion. This is done by a simple butterfly shuffle, shuffling pairs of adjacent blocks, and the shuffling is done just once. Thus another round of Butterfly-Shuffle is performed and which is based on key2.
- Step 6: Finally, all the blocks are concatenated to form 512-bit ciphertext.

Thus, Shuffle-RAT incorporate diffusion in the structure of RAT using the butterfly shuffle, sandwiched between two regular rounds of RAT, which already provides sufficient amount of confusion as in the original design. The decryption is same as the above step, but in the middle phase, that is, in RAT, operation the modulo subtraction is performed instead of modulo addition. Since it is a symmetric block so repeated modulo addition would form the original bit stream, but this number of iterations would increase in exponential term with the increase of block size. Thus, for decryption modulo subtraction is proposed.

Table 7.1: Number of iteration to regenerate source stream using modulo-addition

Block size	No. of iterations
2	4
4	16
8	256
16	65536

Table 7.1 gives number of iterations against block sizes in SRAT. If observe table 7.1, four numbers of iterations is required for 2-bit block size, 16 numbers of iterations is required for 4-bit block size and 65536 numbers of iterations are required to get back the original source stream for 16-bit block size using modulo-addition. Thus for 512-bit block size the number of iterations will be huge (in millions) to get back the original source stream using modulo-addition. Figure 7.2 illustrates the same graphically.

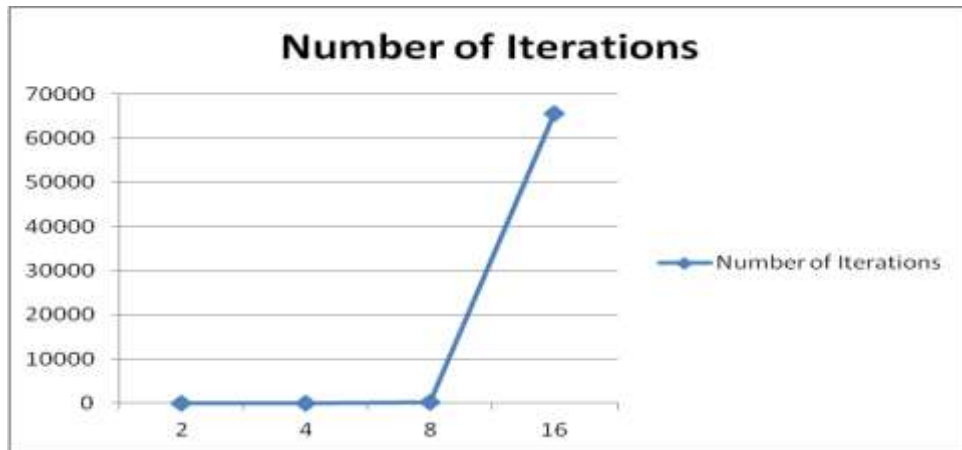


Figure 7.2: Graphical representation of number of iterations to obtain source stream using modulo-addition

It is seen from figure 7.2 that the number of iteration to get back the original source stream varies exponentially with the block size, therefore modulo-subtraction is proposed for decryption.

7.3 Example

An example of SRAT has been given. Consider the plaintext as 1011110111000111. Table 7.2 gives the encryption process, here plaintext of 16-bit is considered. At first the 16-bit plaintext is divided into eight 2-bit block size, and then a butterfly shuffle is performed.

Table 7.2: Encryption process of SRAT

Plaintext		1011110111000111
Round 1 (Block size = 2 bits)	Butterfly-Shuffle	11 11 01 10 11 11 00 01
	RAT	11 10 11 01 00 11 11 00
	Butterfly-Shuffle	10 11 01 11 00 00 11 11
Next Input		1011011100001111
Round 2 (Block size = 4 bits)	Butterfly-Shuffle	0111 1011 1111 0000
	RAT	0111 0010 1111 1111
	Butterfly-Shuffle	0010 0111 1111 1111
Final Ciphertext		0010011111111111

Table 7.2 gives the encryption process, here plaintext of 16-bit is considered. At first the 16-bit plaintext is divided into eight 2-bit block size, and then a butterfly shuffle is performed.

Let consider the plaintext as $P = '1011110111000111'$. The eight blocks will be, $B1=10, B2=11, B3=11, B4=01, B5=11, B6=00, B7=01$ and $B8=11$. In Butterfly-Shuffle in the left part ($B1, B2, B3$ and $B4$) is, $B3$ will be replaced by $B4, B2$ will be replaced by $B3, B1$ will replace by $B2$ and $B4$ will replace by $B1$. The Butterfly-Shuffle for right part ($B5, B6, B7$ and $B8$) is, $B6$ will be replaced by $B5, B7$ will be replaced by $B6, B8$ will be replaced by $B7$ and $B5$ will be replaced by $B8$. This type of replacing is known as Butterfly-shuffle.

Table 7.3: Decryption process of SRAT

Ciphertext		0010011111111111
Round 1 (Block size = 4 bits)	Butterfly-Shuffle	0111 0010 1111 1111
	RAT	0111 1011 1111 0000
	Butterfly-Shuffle	1011 0111 0000 1111
Next Input		1011011100001111
Round 2 (Block size = 2 bits)	Butterfly-Shuffle	11 01 11 10 11 00 00 11
	RAT	11 10 11 01 00 11 11 00
	Butterfly-Shuffle	10 11 11 01 11 00 01 11
Final Plaintext		1011110111000111

Then one round of RAT is performed but here 2-bit modulo addition is done, again another round of Butterfly-Shuffle is done. All the eight blocks will be input to the next round of Shuffle-RAT encryption, in this round, 16-bit sub-stream is divided into four blocks of 4-bits each, and then Butterfly-Shuffle is performed.

Let consider the sub-stream, $S = '1011011100001111'$. The four blocks would be, $B1=1011, B2=0111, B3=0000, B4=1111$. In Butterfly-Shuffle in the left part just $B1$ and $B2$ are swapped. In right part of Butterfly-Shuffle just $B3$ and $B4$ are swapped. This type of replacing is known as Butterfly-shuffle.

Then one round of RAT operation is performed where 4-bit modulo addition is done. Again another round of Butterfly-Shuffle is performed. Finally Concatenation of all four blocks will result in 16-bit ciphertext.

Table 7.3 gives the decryption process of the same ciphertext generated in table 7.3. The process of decryption is same as that of encryption, but with two differences, at first 4-bit block size is considered then 2-bit block size is considered. Second, in RAT operation

7.4 Implementation and Key Generation

The implementation of SRAT is done in IEEE VHDL and synthesized in Xilinx 8.1i. These contains various modules and sub modules. Before laying out the architectural plan for this proposed cipher, SRAT, let take note of all components that it will be required to use in this context:

- **Storage:** The plaintext is stored in a 512 bit, which is a 64-byte register array denoted by ‘regbox’. The key is stored in a 128 bit, which is a 16-byte register array denoted by ‘keymod’. The masks for two rounds are stored in a 10-byte register array denoted by ‘mskbox’.
- **Logic blocks:** This consists of the main controller module denoted by ‘srat_main’, the individual circuits for 8 rounds of Shuffle-RAT (SRAT2 to SRAT256), and the access logic and multiplexing circuit to read and write from the storage.
- **SRAT_n:** Means Shuffle-RAT algorithm with n-bit of plaintext, SRAT2 means Shuffle-RAT with 2-bit plaintext block, SRAT256 means Shuffle-RAT with 256-bit plaintext block.

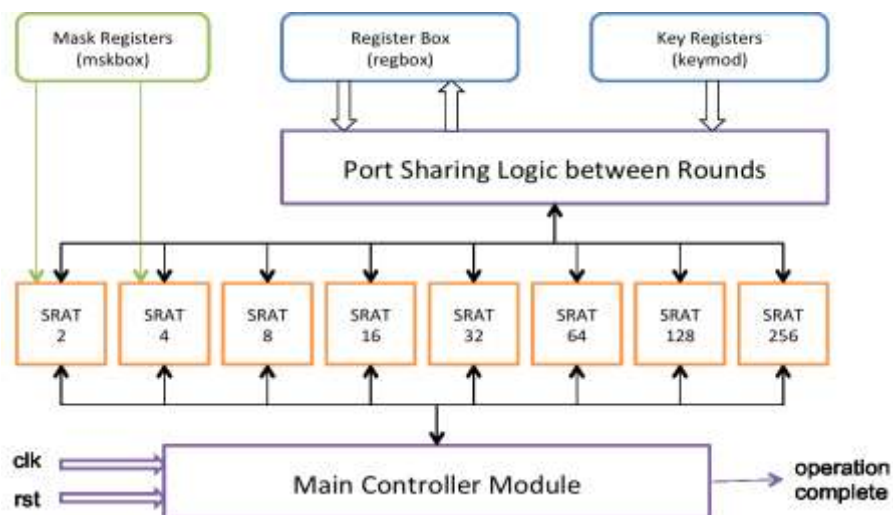


Figure 7.3: Top-level hardware architecture for Shuffle-RAT

Figure 7.3 shows the design of Shuffle-RAT for FPGA simulation. The plaintext, keys are taken from storage registers named ‘regbox’ and ‘keymod’. The complete Shuffle-RAT operations are done in SRAT2 to SRAT256, as described in the algorithm. Clock and reset inputs are fed to the main controller module, which instructs the SRAT modules to operate in a particular sequence, and indicates when all operations are completed successfully.

The registers are byte array (8 bit) for the storage ‘regbox’, whereas SRAT2 and SRAT4 require the access of 2-bit and 4-bit blocks respectively. This is why the masks are stored in ‘mskbox’ to access the required 2-bit or 4-bit blocks from the bytes. Another main point in terms of an efficient design is that the blocks SRAT2 to SRAT256 operate sequentially, and do not overlap in time. Thus, the access ports to the storage modules, that are ‘regbox’, ‘keymod’ and ‘mskbox’, can be shared among the SRAT operations. So, the port sharing logic between rounds of Shuffle-RAT is incorporated.

The main storage for the Shuffle-RAT hardware is the ‘regbox’ array and the ‘keymod’ array. The ‘regbox’ comprises of 8 bit registers made of edge-triggered master-slave flip-flops, with a total of 64 such registers to hold the 512-bit plaintext. To accommodate the read and write accesses to the ‘regbox’, use write-access decoders and read-access decoders, which in turn control 64-to-1 multiplexer units associated to each location of the array. The ‘keymod’ that holds the 128-bit Shuffle-RAT key is also designed in a similar fashion, but with the exception that no intermediate write accesses are required for the registers.

Modulo adder is another important component in this proposed technique, SRAT and also for ROBAST. Carry Look-ahead Adder is designed for this purpose. The carry Look-ahead Adder (CLA) solves the carry delay problem by calculating the carry signals in advance, based on the input signals. It is based on the fact that a carry signal will be generated in two cases: (4) when both bits a_i and b_i are 1 or (5) when one of the two bits is 1 and the carry-in is 1. Thus, one can write,

$$c_{i+1} = a_i \cdot b_i + (a_i \text{ XOR } b_i) \cdot c_i \quad (4)$$

$$s_i = (a_i \text{ XOR } b_i) \text{ XOR } c_i \quad (5)$$

The above two equations can be written in terms of two new signals P_i and G_i , which are shown in figure 7.4.

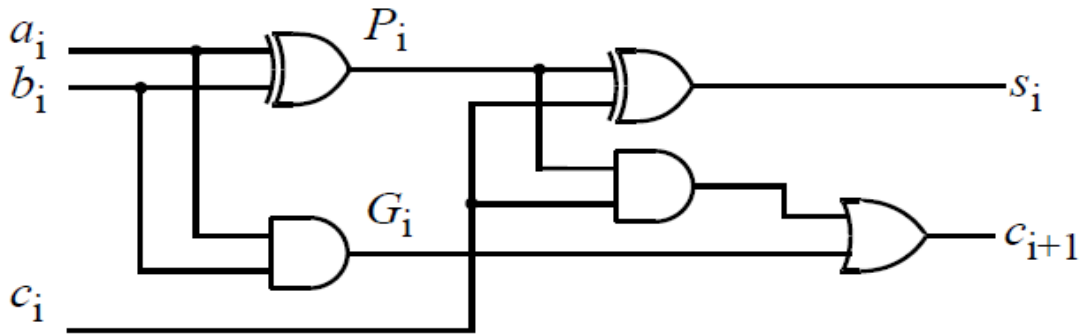


Figure 7.4: Full adder at stage i with P_i and G_i

- $c_{i+1} = G_i + P_i \cdot c_i$ (6)
- $s_i = P_i \text{ XOR } c_i$ (7)

Where

- $G_i = a_i \cdot b_i$ (8)
- $P_i = a_i \text{ XOR } b_i$ (9)

G_i and P_i are called the carry generate and carry propagate terms, respectively. Notice that the generate and propagate terms only depend on the input bits and thus will be valid after one and two gate delay, respectively. If one uses the above expression to calculate the carry signals, one does not need to wait for the carry to ripple through all the previous stages to find its proper value.

Let's apply this to a 4-bit adder to make it clear. Putting $i = 0, 1, 2, 3$ in equation (6) got

- $c_1 = G_0 + P_0 \cdot c_0$ (10)

- $c_2 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot c_0$ (11)

- $c_3 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot c_0$ (12)

- $c_4 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot c_0$ (13)

Figure 7.5 shows that a 4-bit CLA is built using gates to generate the P_i and G_i signals and a logic block to generate the carry out signals according to Equations 10- 13. For modulo-4 CLA only have to discard c_4 . CLAs are usually implemented as 4-bit modules and are used in a hierarchical structure to realize adders that have multiples of 4-bits. It is a simple matter to develop a more versatile 2's complement adder/subtractor based on the adder in figure 7.5.

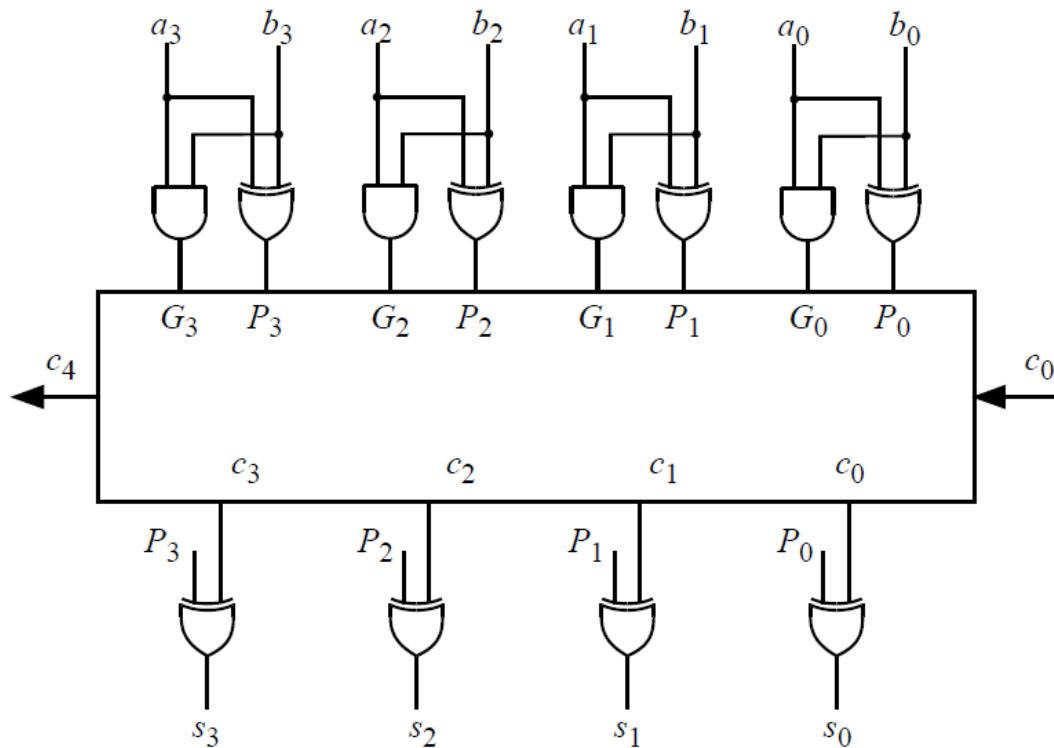


Figure 7.5: 4-bit modulo carry look-ahead adder implementation details

Section 7.4.1 describes the key generation process and section 7.4.2 gives an example of key generation.

7.4.1 Key Generation

In the proposed technique, eight rounds have been considered, each for 2, 4, 8, 16, 32, 64, 128, and 256 block size. Each round is repeated for a finite number of times and the number of iterations will form a part of the encryption-key. Although the key may be formed in many ways, for the sake of brevity it is proposed to represent the number of iterations in

each round by a 16-bit binary string. The binary strings are then concatenated to form a 128-bit key for a particular key.

In the proposed technique, eight rounds have been considered, each for 2, 4, 8, 16, 32, 64, 128, and 256 block size. Each round is repeated for a finite number of times and the number of iterations will form a part of the encryption-key. Although the key may be formed in many ways, for the sake of brevity it is proposed to represent the number of iterations in each round by a 16-bit binary string. The binary strings are then concatenated to form a 128-bit key for a particular key.

Table 7.4: Representation of number of iterations in each round in SRAT

Round	Block Size	Number of Iterations	
		Decimal	Binary
8.	256	50021	1100001101100101
7.	128	49870	1100001011001110
6.	64	48950	1011111100110110
5.	32	44443	1010110110011011
4.	16	46250	1011010010101010
3.	8	4321	0001000011100001
2.	4	690	0000001010110010
1.	2	72	0000000001001000

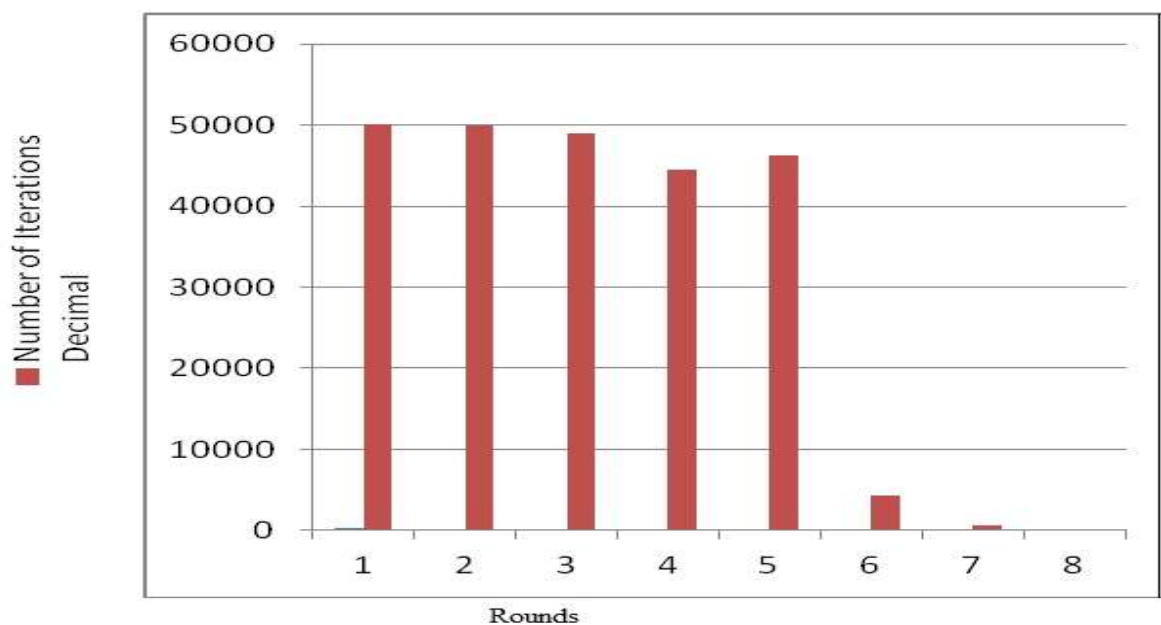


Figure 7.6: Graphical representation of round v/s iteration



Figure 7.7: Session key generation for SRAT

Figure 7.7 shows the top level RTL diagram of session key generation, here two session keys are generated, session key 1 (SK1_RAT) is used for RAT operation that is modulo addition operation, session key 2 (SK2_SHUFFLE) is used for two rounds of butterfly shuffle. The total size of key is 128-bits.

7.4.2 Example of Key Generation

Consider a particular session where the source file is encrypted using iterations for block sizes 2, 4, 8, 16, 32, 64, 128, and 256 bits, respectively. Table 7.4 shows the corresponding binary value for the number of iterations in each round. Figure 7.6 shows the graph for the round v/s iteration. The binary strings are concatenated together to form the 128-bit binary string:

```

110000110110010111000010110011101011111100110110101011011001101110110
10010101010000100001110000100000010101100100000000001001000
  
```

This 128-bit binary string will be the key for encryption for a particular session. During decryption, the same key is taken to iterate each round of modulo-subtraction for the specified number of times.

7.5 Analysis

This technique is derived from Rotational Addition Technique (RAT), the advantages of RAT are:-

- The technique can take little time to encode and decode though the block length is large.

- The encoding string will not generate any overhead bits.
- Selecting the block pairs randomly (rather than consecutive pairs) may increase the security.

RAT also has severe limitations which are listed below:-

- The first 2-bits of the message never get encrypted. So, the first two bits are always in the hand of cryptanalyst.
- The first 4-bits of the message never gets encrypted from round two onwards. The first 8-bits of the message never gets encrypted from round three onwards and so froths from all the rounds.
- In general first 2^k bits never gets encrypted in round 'k' onwards.
- Key size of RAT encryption and RAT decryption is 16-bits per round. Thus the runtime of RAT (with 16-bits of keys at each level) is approximately in order of $8*2^{16}=2^{19}$ RAT cycles. Where one cycle is equivalent to on complete RAT operation over the whole 512-bits. This number makes RAT a bit slower.

The basic ideas of the design of SRAT are as follows:-

- The degree of randomness has also increased here by introducing two butterfly shuffles in between one round of RAT.
- RAT does the job of confusion well but butterfly shuffle introduce the job of diffusion as well.

Thus got much better technique, the algorithmic complexity of SRAT is found to be $O(n^2)$.

7.6 Results and Simulations

In this section, the various results obtained on implementation of the proposed technique, Shuffle-RAT, has been compared with the previous techniques, TPRT, TMAT and ROBAST. This technique is also compared with popular and industrially accepted cipher, RSA. The comparisons are done in two categories, first one is the comparisons based on

FPGA-Based hardware implementation and second one is the comparisons based on software implementation through C-programming language. The software implementation includes Chi-Square values, encryption time and decryption time, avalanche test and frequency distribution. The hardware implementation will mainly deal with the Register Transfer Logic (RTL) parameters and diagrams. Section 7.6.1 discuss results of RTL/Hardware implementation, section 7.6.2 discuss the results of frequency distribution graph, section 7.6.3 discuss the results of Chi-Square test for non-homogeneity of source files and encrypted files, section 7.6.4 discuss the results of time complexity and section 7.6.5 discuss the results of avalanche ratio test.

7.6.1 RTL Simulation Based Result

In this section gives some of the results found after implementing the proposed technique in VHDL. This code has been simulated and synthesized in Xilinx 8.1i. The main objective is to find an efficient FPGA-based cryptographic technique for implementation in embedded systems.

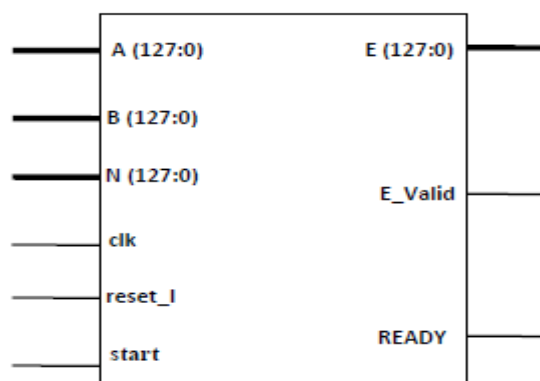


Figure 7.8: RTL diagram of RSA

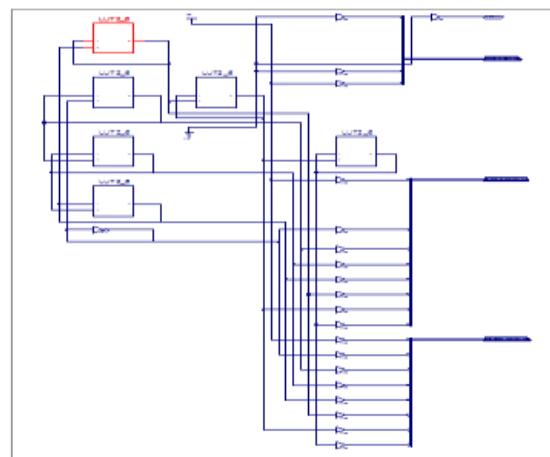


Figure 7.9: Spartan 3E RTL diagram of TPRT

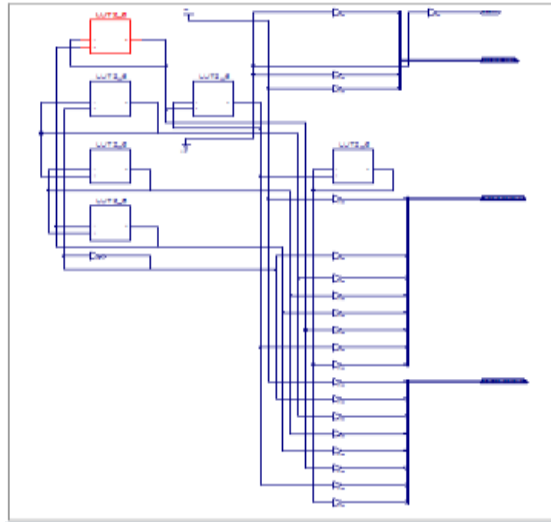


Figure 7.10: Spartan 3E RTL diagram of TMAT

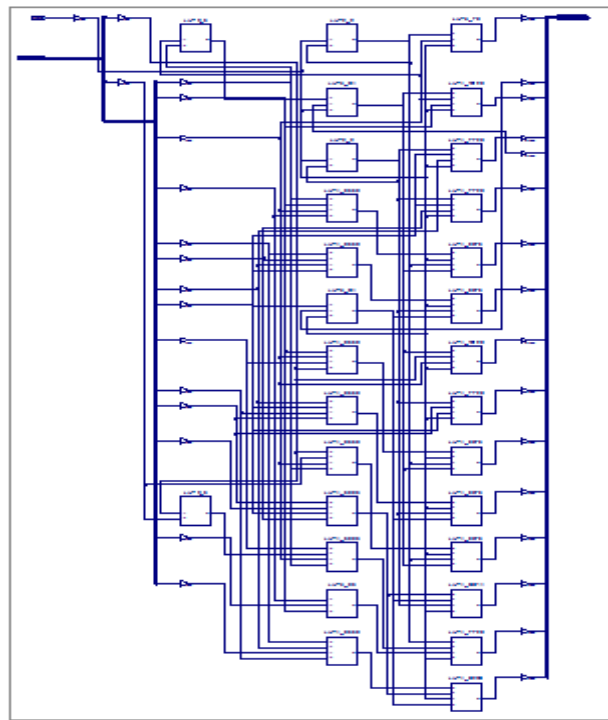


Figure 7.11: Spartan 3E schematic of ROBAST

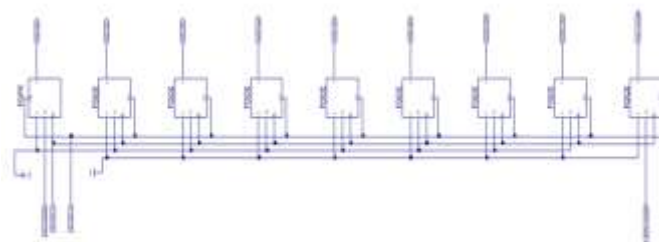


Figure 7.12: Spartan 3E RTL schematic of the main controller module of Shuffle-RAT

Figure 7.8 shows the RTL schematic of RSA, figure 7.9 shows the RTL schematic of TPRT, figure 7.10 shows the RTL schematic of TMAT, figure 7.11 shows the RTL schematic

of ROBAST and figure 7.12 shows RTL schematic of SRAT. If the figures are analyzed given above it can be seen that a few Look-Up-Tables are required. Nine lookup tables are required to realize this proposed technique, SRAT, in Spartan 3E FPGA. So, this technique uses the resources efficiently, the netlist study and speed grade study is discussed in subsequent paragraphs.

Table 7.5: HDL synthesis report (Netlist generation of RSA, TPRT, TMAT, ROBAST and SRAT)

Sr No.	Netlist Components	Number				
		RSA	TPRT	TMAT	ROBAST	SRAT
1	ROMs/RAMs	430	10	14	25	28
2	Adders/Subtractions	3	0	2	20	28
3	Registers	420	20	30	50	641
4	Latches	80	0	0	10	80
5	Multiplexers	120	0	0	10	136

Table 7.5 gives the HDL synthesis of netlist generation, the number of ROMs/RAMs and adder/subtraction used in SRAT in 28 which is the highest than other techniques, number of register of SRAT is 641 which is also highest, the number of latches is 80 and multiplexers is 136, these results are also highest. So, in terms of netlist generation SRAT consuming the maximum resources efficiently.

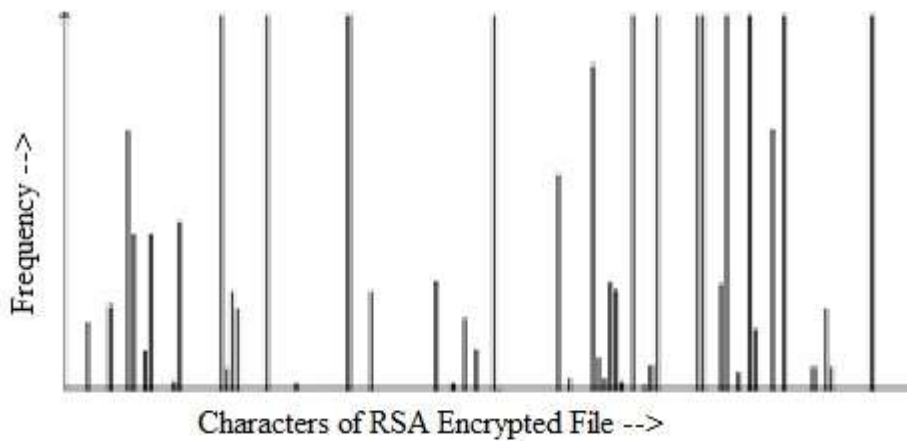
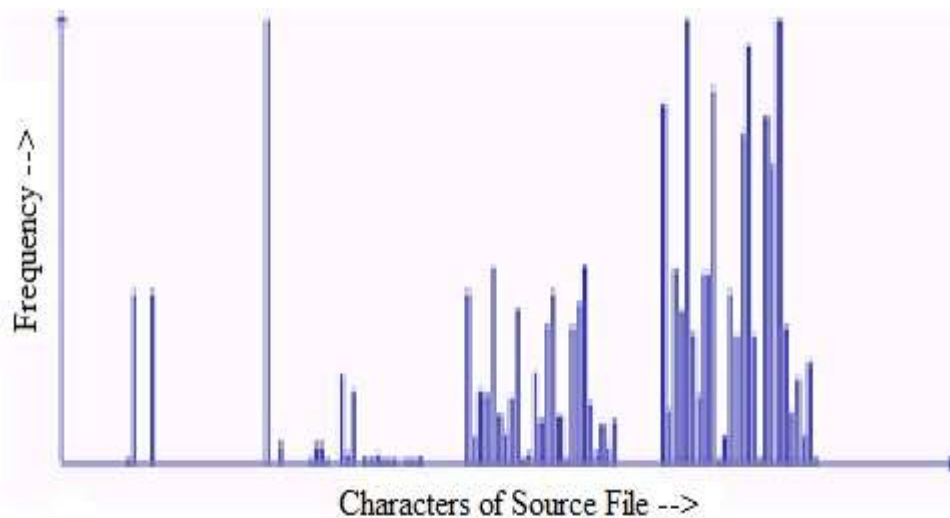
Table 7.6: HDL synthesis report (Timing summary of RSA, TPRT, TMAT, ROBAST and SRAT)

Sr No.	Timing Constraint	Values				
		RSA	TPRT	TMAT	ROBAST	SRAT
1	Speed Grade	-5	-5	-5	-5	-5
2	Minimum period (ns)	9.895	5.66	7.95	5.55	5.50
3	Maximum Frequency (MHZ)	101.06	101.06	101.06	101.06	101.06
4	Minimum input arrival time before clock (ns)	6.697	4.33	5.55	5.55	4.25
5	Maximum output required time after clock (ns)	4.31	3.33	4.25	4.44	3.33

Table 7.6 gives the timing synthesis of the techniques, RAT giving the minimum period of 5.50ns, minimum input arrival time before clock of SRAT is 4.25ns, and Maximum output required time after clock of SRAT is 3.33ns, these shows the optimal result of SRAT than TPRT, TMAT, ROBAST and RSA. Thus SRAT is the best implemented in FPGA-based systems.

7.6.2 The Frequency Distribution Graph

The frequency distribution is the distribution of the all 256 ASCII characters in the respective files. This is also a cryptographic parameter which measures the degree of cryptanalysis.



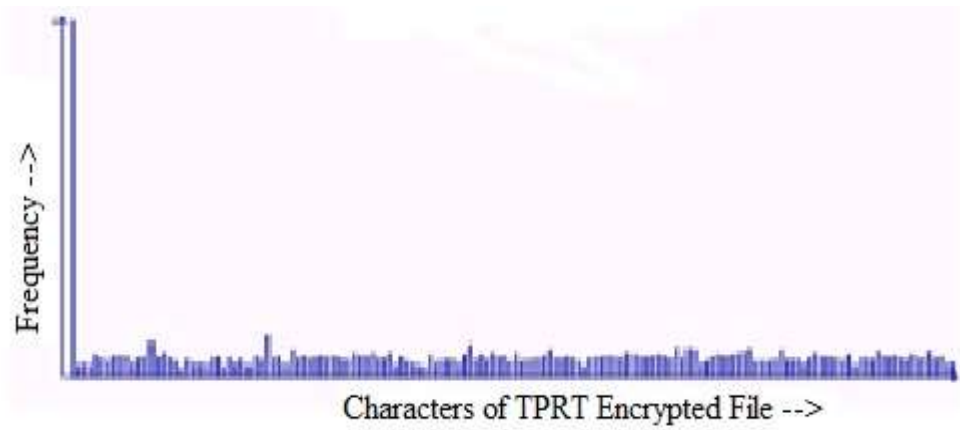


Figure 7.13: Frequency distribution graph of source, RSA encrypted and TPRT encrypted files

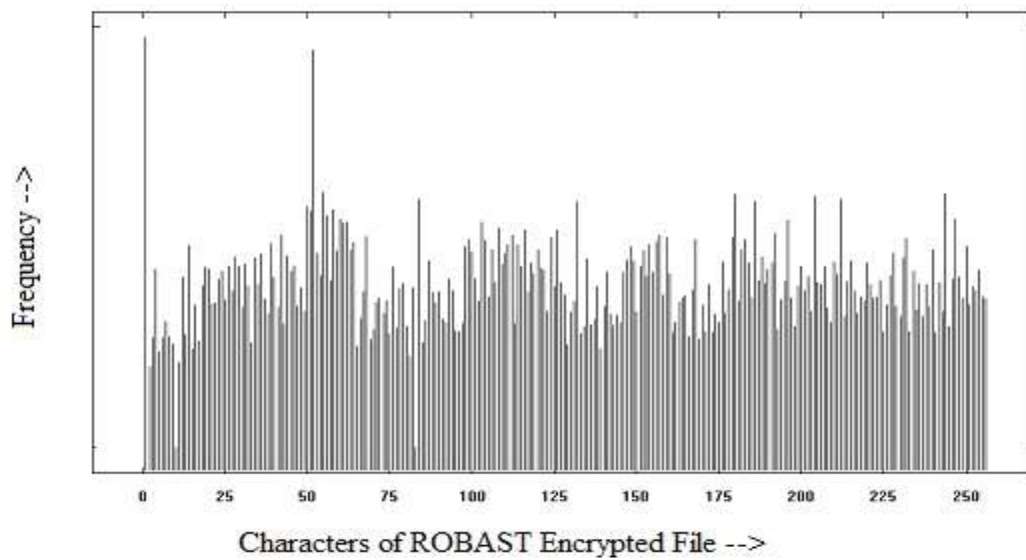


Figure 7.14: Frequency distribution graph of TMat and ROBAST encrypted files

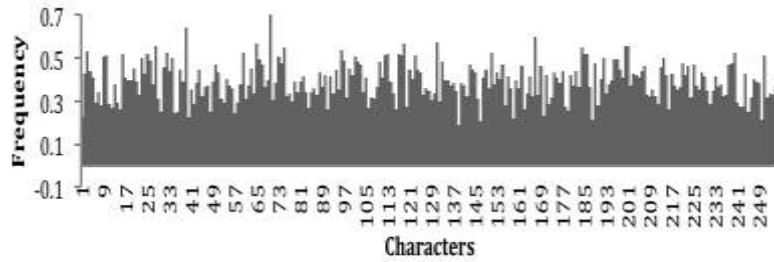


Figure 7.15: Frequency distribution graph of SRAT encrypted files

Figure 7.13 illustrates the source file, RSA encrypted file and TPRT encrypted file frequency distribution results found after implementation of respective algorithms/techniques. Figure 7.14 illustrates the frequency distribution of TMAT and ROBAST encrypted file. Figure 7.15 illustrates the frequency distribution graph of SRAT encrypted file. The frequency distribution graph of all the proposed techniques, SRAT, ROBAST, TPRT and TMAT are giving the optimal result. All the frequencies are evenly distributed over 256 region for all the technique except that of RSA where the frequency distribution is not evenly distributed and somewhat resembles frequency distribution of a text file. Though ten files have been encrypted but for this result the file ‘genesis.txt’ of size 48.44 KB is considered. Therefore there is no substantial improvement in this result for SRAT.

7.6.3 The Non-Homogeneity Test

This section shows the extent of non-homogeneity between source file and encrypted file. To test this Chi-Square is taken as a parameter. Here the observed frequency is source file that is plaintext and the expected frequency is the ciphertext that is encrypted file. Ten files has been taken for this test in increasing order of file size, the size starts from 17,632 bytes (17.22 KB) and goes to 403,901 bytes (394.43 KB). Among these ten files, four text files has been taken, one Microsoft word file has been taken, three executable file has been taken and two dll files has been taken.

These files are then repeatedly encrypted by RSA, TPRT, TMAT, ROBAST and SRAT, then with software tools the Chi-Square value between the source files and encrypted files are noted down in tabular format.

Table 7.7: Comparison of Chi-Square values of ROBAST, RSA, TPRT, TMAT and SRAT

Source File	File Size (Bytes)	Chi-Square Values				
		ROBAST	RSA	TMAT	TPRT	SRAT
license.txt	17,632	6472	5668	201530	191382	201960
cs405(ei).doc	25,422	4407	2654	286025	253470	305590
acread9.txt	35,121	560357	447984	440184	410735	451125
deutsch.txt	47,829	3307374	685963	555220	505121	558330
genesis.txt	49,600	2679799	3318506	659045	638592	683128
pod.exe	69,981	8495675	694410	905416	896405	937565
mspaint.exe	136,463	3131296	2667664	1297256	1203665	1308890
cmd.exe	152,028	9559993	2216429	1759014	1692655	2009956
d3dim.dll	193,189	3102369	906300	4630652	4250652	9900630
clbcataq.dll	403,901	2590855	3896171	4167801	3922143	4525650

Table 7.8: Comparison of degree of freedom of ROBAST, RSA, TPRT, TMAT and SRAT

Source File	File Size (Bytes)	Degree of Freedom				
		ROBAST	RSA	TMAT	TPRT	SRAT
license.txt	17,632	253	253	255	255	253
cs405(ei).doc	25,422	253	253	255	255	254
acread9.txt	35,121	253	253	255	255	255
deutsch.txt	47,829	253	253	255	255	240
genesis.txt	49,600	253	253	255	255	255
pod.exe	69,981	253	253	255	255	255
mspaint.exe	136,463	254	254	255	255	255
cmd.exe	152,028	253	253	255	255	255
d3dim.dll	193,189	253	253	255	255	255
clbcataq.dll	403,901	253	253	255	255	255

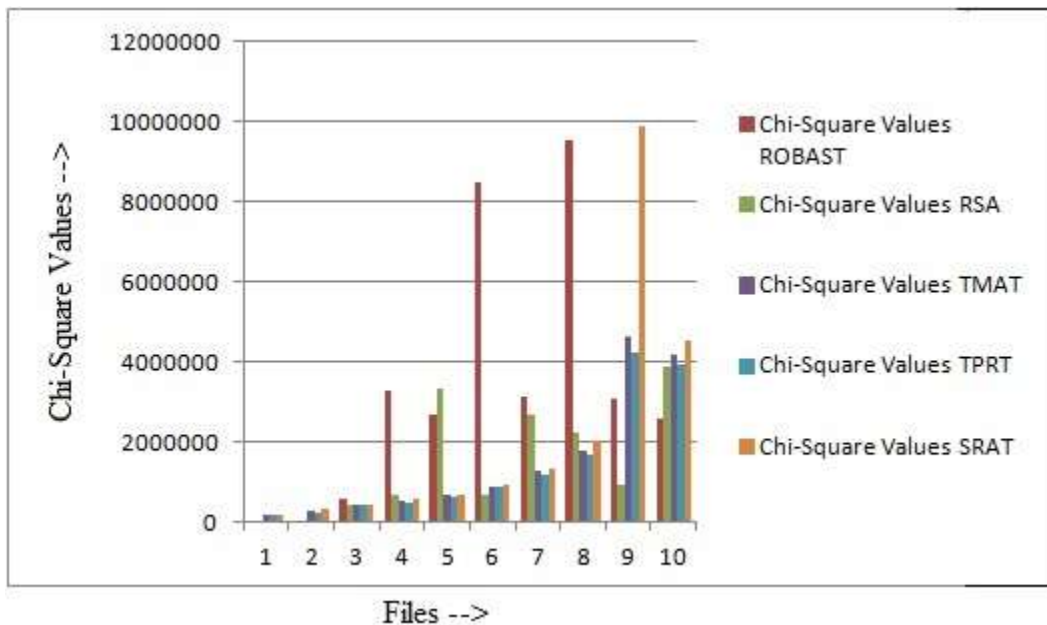


Figure 7.16: Comparison Chi-Square values for ROBAST, RSA, TMAT, TPRT and SRAT

It is seen from the table the average Chi-Square value of SRAT is 2088282, ROBAST is 33,43,860, RSA is 14,84,175, TMAT is 14,90,214 and TPRT is 13,96,482. Therefore it can be said that SRAT is giving the optimal solution for non-homogeneity test but in degree of freedom TMAT and TPRT are giving better result than SRAT. Table 7.7 giving the Chi-Square values, table 7.8 giving the degree of freedom values and figure 7.16 giving the Chi-Square values graphically, X-axis is the ten files and Y-axis is the corresponding Chi-Square values, here bar graph is selected for this result. The degree of freedom result of SRAT is nearly 255. Thus SRAT is giving heterogeneous result than other techniques including RSA.

7.6.4 The Time Complexity Analysis

Time complexity is based on encryption time and decryption time. Encryption time is the time required to encrypt a source file and decryption time is the time to decrypt the cipher text file to get the original file. Ten source files are encrypted and the encryption times are noted, then these encrypted files are decrypted and the decryption time is noted.

Table 7.9: Comparison encryption time of ROBAST, RSA, TMAT, TPRT and SRAT

Source File	File Size (Bytes)	Encryption Time				
		ROBAST	RSA	TMAT	TPRT	SRAT
license.txt	17,632	0.00	0.01	0.03	0.02	0.00
cs405(ei).doc	25,422	0.01	0.06	0.00	0.00	0.01
acread9.txt	35,121	0.02	0.07	0.13	0.10	0.01
deutsch.txt	47,829	0.03	0.11	0.25	0.20	0.01
genesis.txt	49,600	0.04	0.12	0.28	0.25	0.02
pod.exe	69,981	0.04	0.12	0.39	0.35	0.02
mspaint.exe	136,463	0.06	0.20	0.44	0.40	0.03
cmd.exe	152,028	0.07	0.25	0.55	0.50	0.05
d3dim.dll	193,189	0.08	0.28	0.55	0.52	0.05
clbcatq.dll	403,901	0.08	0.32	0.67	0.60	0.05

Table 7.10: Comparison of decryption time of ROBAST, RSA, TMAT, TPRT and SRAT

Source File	File Size (Bytes)	Decryption Time				
		ROBAST	RSA	TMAT	TPRT	SRAT
license.txt	17,632	0.01	0.15	0.11	0.10	0.00
cs405(ei).doc	25,422	0.02	0.71	0.00	0.00	0.01
acread9.txt	35,121	0.03	1.15	0.13	0.10	0.01
deutsch.txt	47,829	0.03	1.36	0.15	0.11	0.01
genesis.txt	49,600	0.04	1.61	0.25	0.20	0.02
pod.exe	69,981	0.04	1.86	0.39	0.35	0.02
mspaint.exe	136,463	0.05	2.71	0.48	0.40	0.02
cmd.exe	152,028	0.06	3.34	0.52	0.42	0.05
d3dim.dll	193,189	0.07	3.73	0.60	0.50	0.05
clbcatq.dll	403,901	0.08	4.25	0.65	0.55	0.05

Table 7.9 gives the encryption times of all the techniques including RSA, table 7.10 gives the decryption times of all the techniques including RSA. Figure 7.17 shows the encryption time graphically and figure 7.18 shows the decryption time graphically.

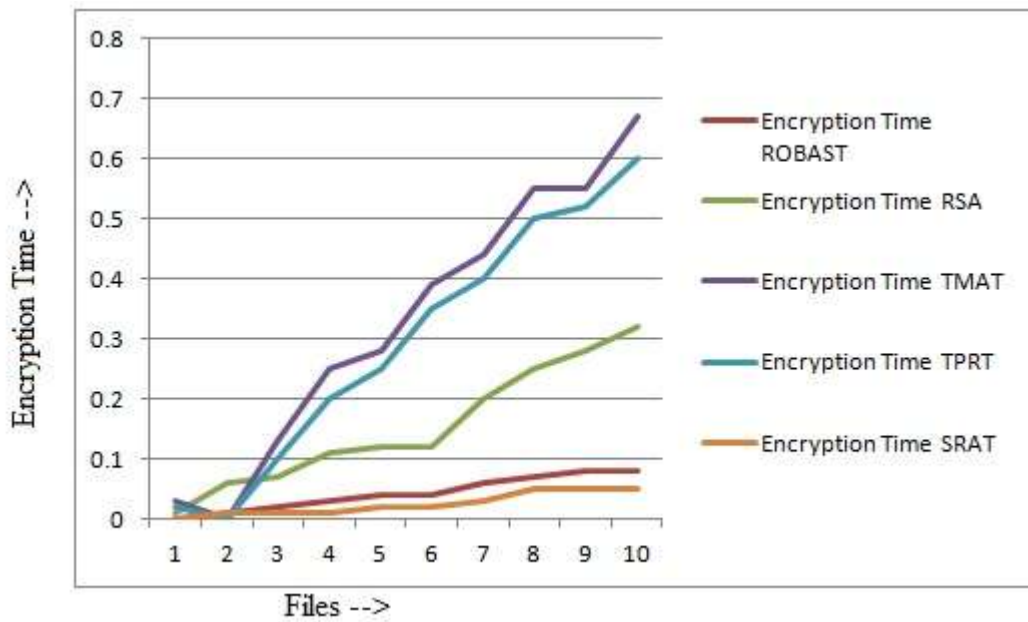


Figure 7.17: Pictorial representation of encryption time against file size

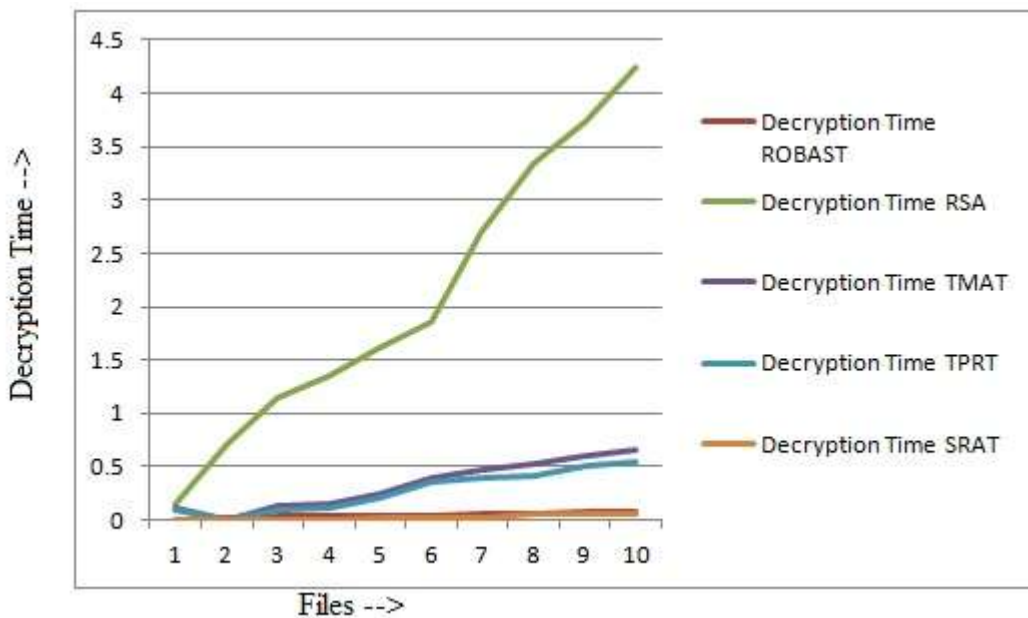


Figure 7.18: Pictorial representation of decryption time against file size

The cumulative encryption time of ROBAST is 0.43 seconds, RSA is 3.54 seconds, TMAT is 3.29 seconds, TPRT 2.94 seconds and SRAT is 0.25 seconds. The cumulative decryption time of ROBAST is 0.43 seconds, RSA 20.87 seconds, TMAT is 3.28 seconds, TPRT is 2.63 seconds and SRAT is 0.24 seconds. Therefore SRAT is giving the best result in terms of encryption and decryption time.

7.6.5 The Avalanche Ratio Test

The avalanche ratio is the ratio between the modified results to the original result. The avalanche ratio is obtained by modifying 2-3 bits/bytes in the encryption key as well as in source files.

Table 7.11: Comparison of avalanche ratio of ROBAST, RSA, TPRT, TMat and SRAT encrypted files

Source File	File Size (Bytes)	Avalanche Ratio of ROBAST encrypted files (in %)	Avalanche Ratio of RSA encrypted files (in %)	Avalanche Ratio of TPRT encrypted file (in %)	Avalanche ratio of TMat encrypted file (in %)	Avalanche ratio of SRAT encrypted file (in %)
license.txt	17,632	71.90	58.0	77.7	80.8	91.5
cs405(ei).doc	25,422	99.69	60.0	80.0	85.5	90.5
acread9.txt	35,121	99.93	75.0	88.8	90.0	98.0
deutsch.txt	47,829	99.96	78.9	89.0	91.5	99.5
genesis.txt	49,600	97.72	80.9	87.0	94.7	99.9
pod.exe	69,981	77.00	58.0	77.0	80.0	99.9
mspaint.exe	136,463	98.22	58.9	76.0	80.0	98.0
cmd.exe	152,028	99.97	67.0	77.0	80.0	97.0
d3dim.dll	193,189	99.98	67.9	82.9	85.0	97.5
clbcattq.dll	403,901	75.55	68.0	88.5	90.5	99.0

Table 7.11 gives the avalanche ratio of all the techniques, thus it can be seen that SRAT is giving the best result in terms of avalanche ratio test. So, it can be said that modifying few bits or bytes in source file of session key will effect most bits or bytes of encrypted file through SRAT.

7.7 Discussions

In this chapter, efficient iterated block cipher Shuffle-RAT has been proposed based on an existing design of Rotational Addition Technique (RAT) with a novel inclusion of butterfly-shuffle in the process. Detailed analysis of the new cipher based on relevant cryptographic properties have been studied, and comparison with existing well-known

designs, including the original RAT has also been done. Efficient hardware architecture for Shuffle-RAT implementation has also been done on FPGA, and tested for the feasibility of the design using VHDL description, simulated using Xilinx ISE. The natural step for future work would be to exploit the advantages of Shuffle-RAT through its practical implementation and synthesis on FPGA or ASIC platforms. Shuffle-RAT is also compared to all the other proposed techniques in the next chapter.

Chapter 8
Triple Sagacious Vanquish (TSV)

8.1 Introduction

The Triple SV is a block cipher that uses secret key encryption. This algorithm takes a fixed-length string of plaintext bits and transforms it through a series of complicated operations into another cipher-text bit string of the same length. The proposed block size is 256 bits. The key comprises 112 bits. Figure 8.1 summarizes the overall structure of Triple SV.

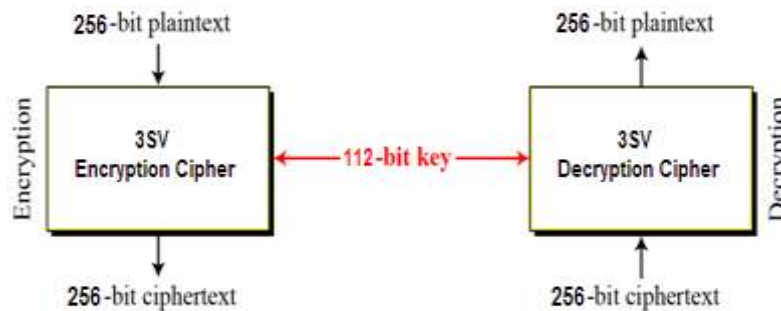


Figure 8.1: Overview of the TSV

The TSV consist of complex operation of encryption and decryption and the key generation produce a key of 112 bits. This proposed technique is also symmetric in nature because the operation required for encryption is same required for decryption with the same key for encryption and decryption. Modes of operation for this proposed technique is Cipher Block Chaining (CBC), which is used for encryption and decryption. This technique is successfully implemented in software module using C programming and also in hardware module using VHDL. Apart from other parameters this proposed technique exhibits a good avalanche effect. The CBC modes of operation converts a block cipher to a stream cipher and stream cipher has a good avalanche effect that is why this proposed technique shows a good avalanche effect. So, through this proposed technique got a stream cipher design using block cipher through CBC. Figure 8.1 shows the block diagram of Triple SV (TSV).

Section 8.2 discussed the algorithm of TSV with a block level diagram, section 8.3 gives a detailed example of encryption and decryption process, section 8.4 discussed the implementation issues with key generation, section 8.5 gives a brief analysis, section 8.6 discussed the results obtained based on implementation and a brief discussions are given in section 8.7.

8.2 The Algorithm of TSV

TSV takes 256-bits plaintext as input and then inverse function is applied. The inverse function is a function which takes a block of bits as input then gives out the complement of these bits. Then seven rounds of encryption is performed, 2-bits block encryption, 4-bits block encryption, 8-bits block encryption, 16-bits block encryption, 32-bits block encryption, 64-bits block encryption and 128-bits block encryption. The details have been discussed in later sub-section. Again inverse function is applied so that it cancels out the first round of inverse function and finally 256-bits of ciphertext obtained. The proposed mode of operation is CBC which gives a high avalanche as well as better non-homogeneity is obtained.

Section 8.2.1 gives a brief discussions on modes of operation, section 8.2.2 gives the encryption process and section 8.2.3 gives the decryption process.

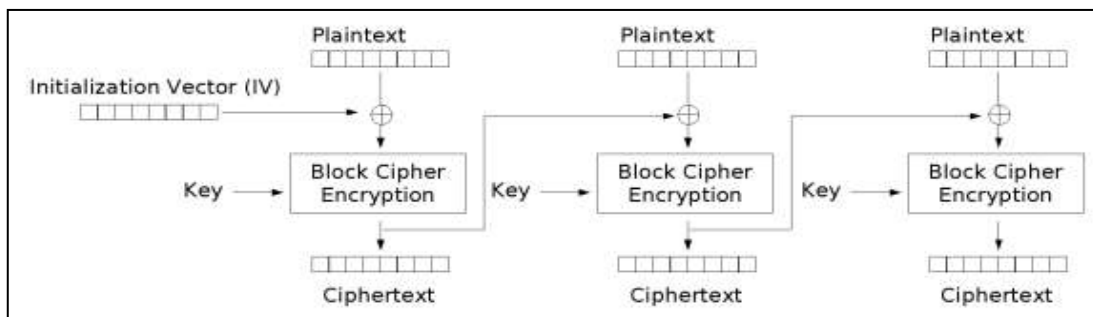


Figure 8.2: The Cipher Block Chaining (CBC) mode for encryption in TSV

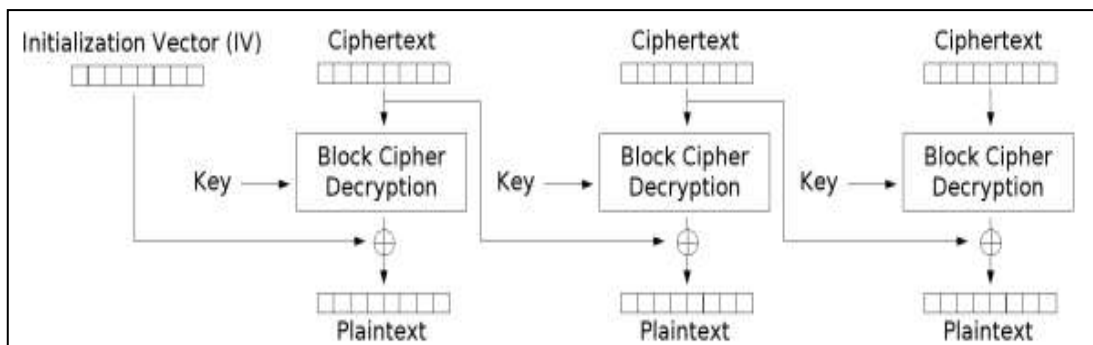


Figure 8.3: The Cipher Block Chaining (CBC) mode for decryption in TSV

8.2.1 Modes of Operation

Like other block ciphers, TSV must be used in one of the several modes of operation, like Electronic codebook (ECB), Cipher-block chaining (CBC), Propagating cipher-block

chaining (PCBC), Cipher feedback (CFB), and Output feedback (OFB). TSV has been designed in CBC mode.

In the CBC mode, each block of plaintext is XORed with the previous cipher-text block before being encrypted. This way, each cipher-text block is dependent on all plaintext blocks processed up to that point. Also, to make each message unique, an initialization vector must be used in the first block. A one-bit change in a plaintext affects all following cipher-text blocks. A plaintext can be recovered from just two adjacent blocks of cipher-text. As a consequence, decryption can be parallelized, and a one-bit change to the cipher-text causes complete corruption of the corresponding block of plaintext, and inverts the corresponding bit in the following block of plaintext. Figure 8.2 and figure 8.3 represent the encryption and the decryption process of CBC mode. CBC mode of operation also converts a block cipher to a stream cipher with high avalanche effect and which is found in the result after implementation of TSV in CBC.

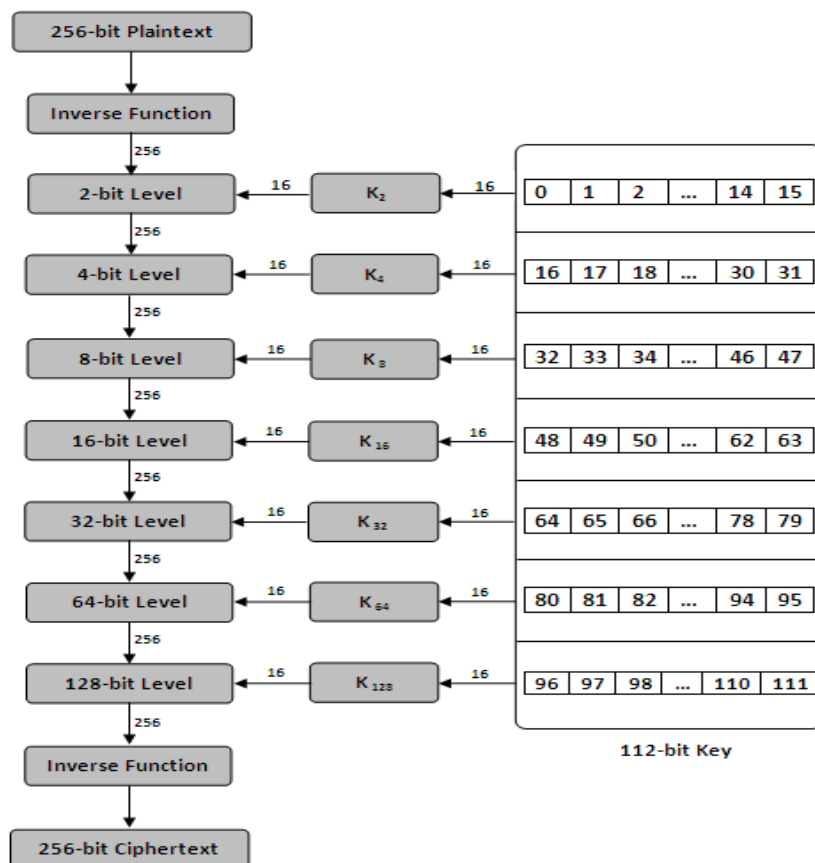


Figure 8.4: TSV encryption overview

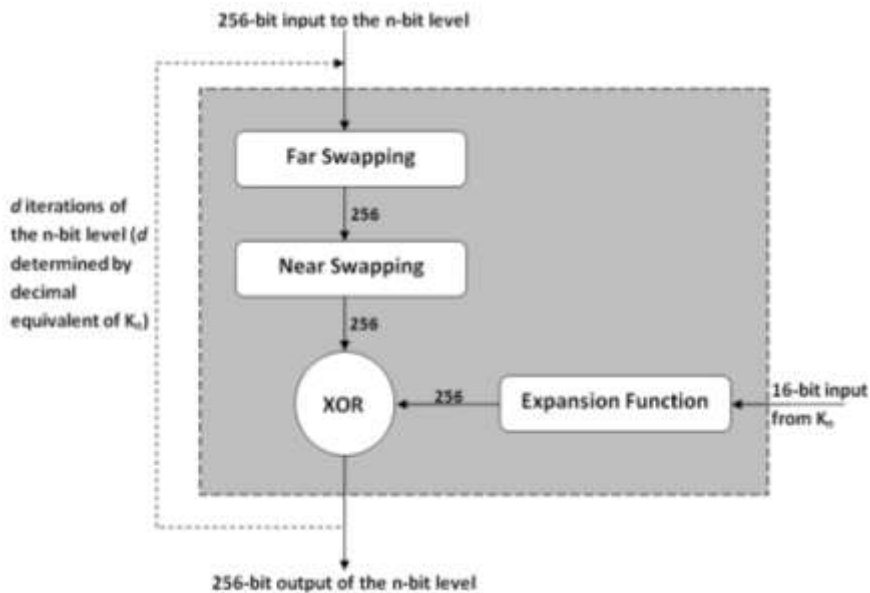


Figure 8.5: n-BIT level structure (encryption) for TSV

8.2.2 Encryption

There are basically seven similar levels of processing. In addition there is also an initial and final inversion operation. The seven similar levels of processing have identical structure but differ in the number of consecutive n bits out of the input 256 bit to each level, which are coupled together and treated as a single entity while being processed inside each level. The values that n take in the 7 distinct levels are 2, 4, 8, 16, 32, 64, 128, (that is $2^{\text{(level number)}}$), respectively. Hence the seven levels of processing are named as 2-bit level, 4-bit level, 8-bit level, 16-bit level, 32-bit level, 64-bit level, and 128-bit level, respectively. This technique's overall structure (for encryption) is shown in figure 8.4.

- n-Bit Level Structure

Figure 8.5 shows the entire construct of the n -bit level. Each level basically comprises three major functions, namely, Far Swapping, Near Swapping and Expansion Function, and a XOR Function. The 256-bit input to the level first undergoes an n -bit far swap. The 256-bit output of the n -bit far swap is then introduced to an n -bit near swap, which again generates a 256-bit output. In the far swap the 0^{th} block is swapped with $(k-1)^{\text{th}}$ block, 1^{st} block is swapped with the $(k-2)^{\text{th}}$ block and so on for k -blocks of input bits and n -bit far swap means the block size is of n -bits. In the near swap the 0^{th} block is swapped with 1^{st} block, 2^{nd} block

is swapped with 3^{rd} block and so on for k-blocks of input bits and n-bit near swap means the block size is of n-bits.

After far swap and near swap of n-bit blocks, all the blocks are combined to get 256-bit stream which is then XORed with the output of the expansion function and the resultant is passed to the next n-bit round structure. Figure 8.6 shows the n-bit far swap where as figure 8.7 show the n-bit near swap.

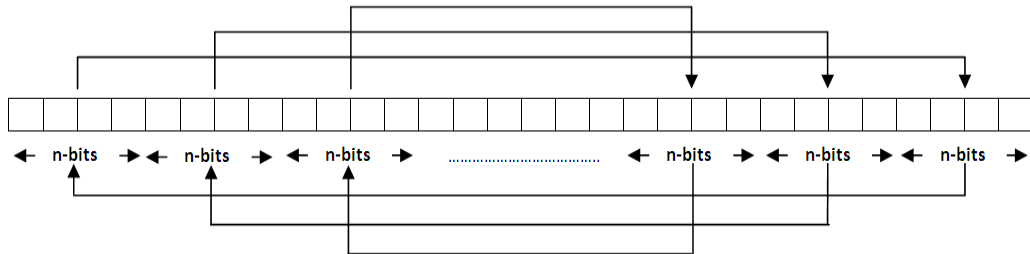


Figure 8.6: n-bit far swap function for TSV

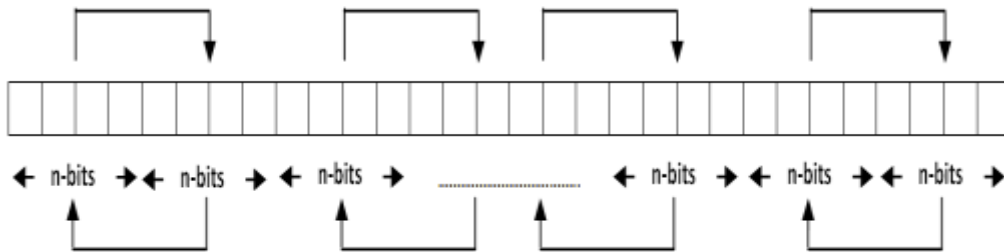


Figure 8.7: n-bit near swap function for TSV

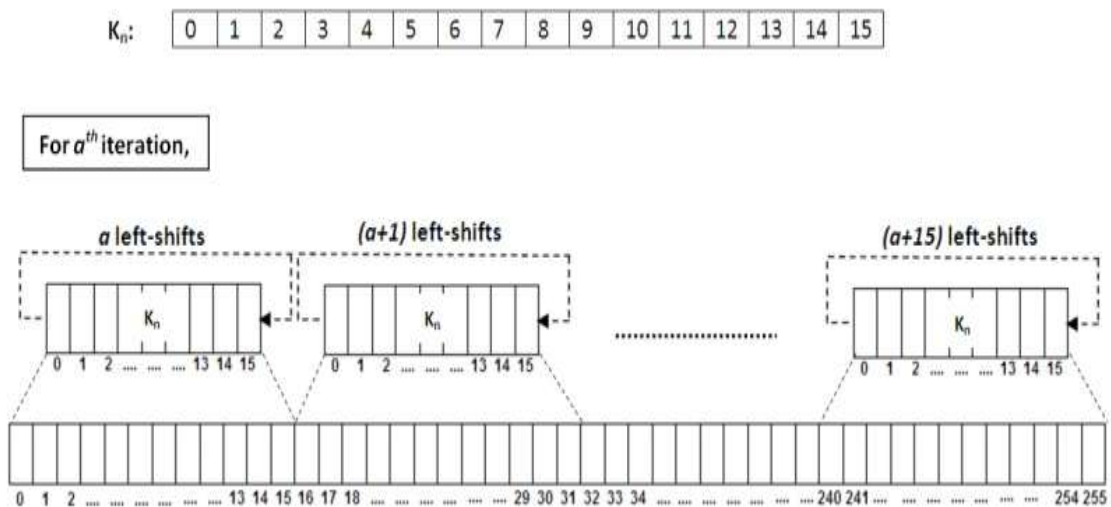


Figure 8.8: Expansion function for encryption of TSV

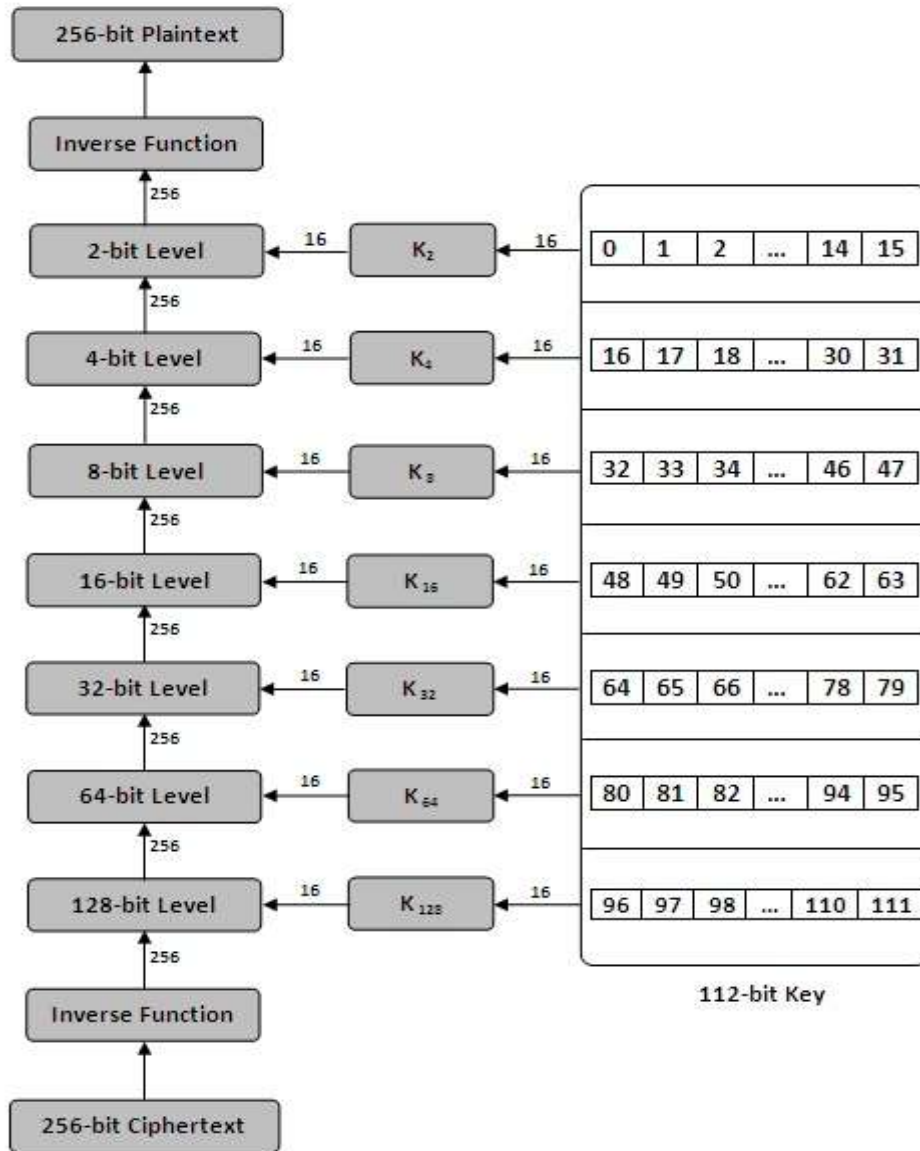


Figure 8.9: TSV decryption process

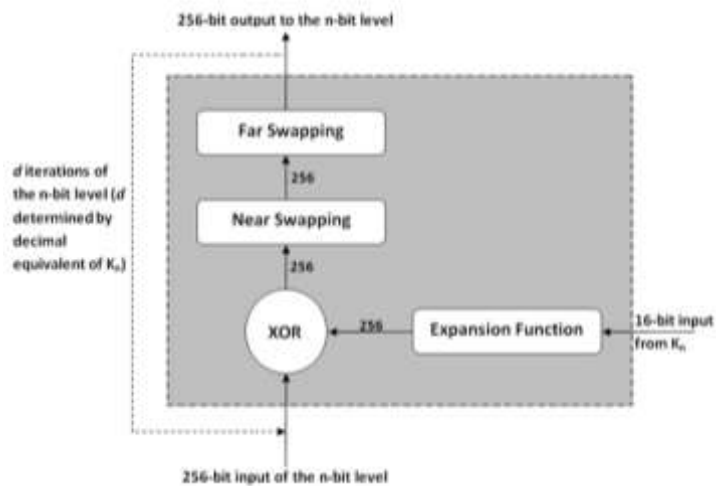


Figure 8.10: n-bit level structure (decryption) of TSV

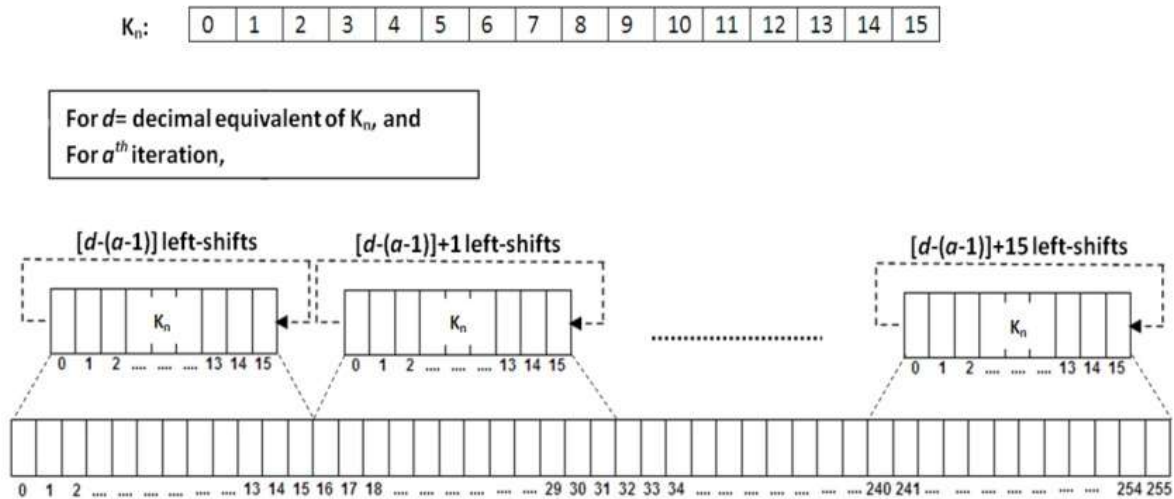


Figure 8.11: Expansion function for decryption of TSV

Meanwhile, the 16-bit string, K_n enters the level and gets expanded into a 256-bit string with which the 256-bit output of the near swap gets XORed to produce a 256-bit intermediate. This intermediate is again fed into the same level to carry out the procedure all over again. This iterative operation of the level continues for d times, where d is a positive integer, the value of which is determined by the decimal equivalent of the string K_n .

After complete iterations of ‘ d ’ times, the 256-bit output is the output of that level and is carried to the next n -bit level for similar series of operations.

- **n-Bit Far Swap Function:** The n -bit far swap function has been diagrammatically depicted in figure 8.6. The n -bit far swap function is a simple function. Firstly, the 256-bit of the incoming string are grouped into distinct n -bit groups, where n is 2, 4, 8, 16, 32, 64, or 128, depending on the level at which are operating. The groups are formed by starting from the first bit and grouping together the first n consecutive bits, then the next n consecutive bit, and so on. These distinct n -bit groups behave as individual entities at that particular level.
- For n -bit far swapping, the first n -bit group gets swapped (interchanged) the last (farthest) n -bit group. The second n -bit group gets swapped with the penultimate n -bit group, and so on.
- **n-Bit Near Swap Function:** The n -bit near swap function is quite similar to n -bit far swapping function, with a subtle change in swapping pattern, as

demonstrated in figure 8.7. For n-bit near swapping, the first n-bit group gets swapped (interchanged) the second (nearest) n-bit group. The third n-bit group gets swapped with the fourth n-bit group, and likewise the penultimate n-bit group is swapped with the ultimate n-bit group.

- Expansion Function (for encryption): The Expansion Function, in comparison to the earlier functions is a little more complex. For a certain n-bit level, the Expansion Function transforms the 16-bit string, K_n into a 256-bit string which is used as an input to the XOR Function. Figure 8.8 summarizes the expansion function for encryption.
- Inverse function: Inverse function is the random permutation of the 256-bit plaintext and it is completely defined by the implementation. The initial inverse function inverts the bits and performs the random permutation. The final inverse function which get the 256-bit output from 128-bit level, it again invert the bits and performed random permutation. The final inverse function is designed in such a way that it cancel out the effect of initial inverse function.

The function takes the 16-bit K_n string as input. Next, it determines the number of iteration of the particular level. Then K_n is given a left-rotations and the modifies string makes the first 16 bits of the expanded string. Another left-rotation is given to the first 16-bits to produce the next 16-bits, and so on.

Sixteen such modifications of the 16-bit string finally produce the 256-bit string for that particular level and that particular iteration. Thus, a 256-bit output is generated by the expansion function, which then gets XORed with the 256-bit output of the n-bit near-swapping of that particular iteration of the level.

8.2.3 Decryption

The decryption algorithm is just the reverse of the encryption algorithm. In case of decryption, the 256-bit cipher text fed to the cipher first undergoes 128-bit level, then 64-bit level and so on till 2-bit level. Figure 8.9 shows the decryption algorithm.

Even the order of operations inside each level is reversed with the expansion function operating first, then the n-bit near swap and then the n-bit far swap, as depicted in figure 8.10. All the individual function retains exactly the same functionality as in case of encryption. The

only function that gets a little modified in case of decryption is the Expansion Function. Figure 8.11 clearly explains the functioning of the expansion function in case of decryption.

8.3 Example

The proposed technique is defined on 256-bit plaintext and 112-bit key with 7-rounds. To understand it in better way here 2-bit level round/structure encryption is illustrated. A 16-bit plain text and a key of 8-bit are taken as an example.

Table 8.1: TSV encryption using 2-bit level with 16-bit plaintext and 8-bit Key

Step No.	Caption	Bit Sequence
1	Input 16-bit Plaintext	1011010010101001
2	Input 8-bit Key	11000110
3	Formation of 2-bit block of Plaintext	10 11 01 00 10 10 10 01
4	Far Swap	01 10 10 10 00 01 11 10
5	Near Swap	10 01 10 10 01 00 10 11
6	Key Expansion	11000110 10001101
7	XOR operation and Final Output	0101110011000110

A simple TSV encryption is shown in table 8.1. The technique, TSV, has seven round structure, 2-bit level, 4-bit level, 8-bit level, 16-bit level, 32-bit level, 64-bit level and 128-bit level. In this example 16-bit plaintext, 8-bit key and 2-bit level structure encryption is illustrated. All the steps are shown in Table 8.1.

- Step 1: Take 16-bit plaintext as input.
- Step 2: Take 8-bit key as input.
- Step 3: Since, it is a 2-bit stage, the 16-bit plaintext is now broken into eight blocks of 2-bit each.
- Step 4: Now far swap function is performed. Let consider the above eight blocks as B1, B2, B3,, B8. In far swap, B1 is swapped with B8, B2 is swapped with B7, B3 is swapped with B6 and finally B4 is swapped with B5.

- Step 5: Now near swap function is performed. Let consider the above eight blocks as B1, B2, B3,, B8. In near swap, B1 is swapped with B2, B3 is swapped with B4, B5 is swapped with B6 and finally B7 is swapped with B8.
- Step 6: In this technique key expansion is proposed to form round keys. To perform XOR operation of round key with 16-bit plaintext have to expand the input 8-bit key. Expansion function works as: first 8-bit of round key is same as 8-bit input key, then 8-bit input key is left rotated by 1-bit and which form the next 8-bit of round key, thus got 16-bit round key.
- Step 7: In this step the 16-bit round key (output from Step no. 6) is XORed with 16-bit output from near swap operation (Step no. 5). Thus got the final encrypted stream.

In actual implementation, the output from 2-bit level is passed to 4-bit level and so on. Decryption is just the opposite of encryption just illustrated. When this algorithm is implemented with CBC mode of operation it get poly-alphabetic cipher with good avalanche effect and better non-homogeneity.

8.4 Implementation and Key Generation

Proposed technique, TSV, has swapping, round key generation and Cipher Block Chaining (CBC) as the main important module for hardware implementation. Let first describe the hardware implementation of CBC. Section 8.4.1 gives the implementation details of Cipher Block Chaining (CBC) mode and section 8.4.2 illustrates the round key generation.

8.4.1 Cipher Block Chaining (CBC) Mode

In Cipher Block Chaining (CBC) mode, the output of one block cipher is fed into the other block cipher along with the next block message. CBC mode converts the block cipher into stream cipher. The algorithm below describes the mode and a pictorial description is provided in figure 8.12 and figure 8.13 respectively.


```

Algorithm CBC_Encryption
K (P)
1: Partition P into P1, P2, . . . , Pm
2: C1 = EK(P1 ⊕ IV);
3: for i = 2 to m
4: Ci = EK(Pi ⊕ Ci-1)
5: end for
6: return C1, C2, . . . , Cm

```

Figure 8.12: Top level algorithm for CBC encryption of TSV

```

Algorithm CBC_Decryption
K (C)
1: Partition C into C1, C2, . . . , Cm
2: P1 = E-1
      K (C1) ⊕ IV
3: for i = 2 to m
4: Pi = E-1
      K (Ci) ⊕ Ci-1
5: end for
6: return P1, P2, . . . , Pm

```

Figure 8.13: Top level algorithm for CBC decryption of TSV

Figure 8.12 depicts the top level algorithm for CBC encryption and figure 8.13 depicts the top level algorithm for CBC decryption, these algorithms has been implemented in both C- programming for software implementation and VHDL implementation for FPGA-based systems. CBC takes as input m message blocks and an initialization vector (IV). During encryption, the output of the ith block depends on the previous i-1 blocks. So, CBC encryption is inherently sequential. The output of each block depends on all the previous blocks and thus provides more security than ECB. The sequential design does not allow a fully pipelined implementation for this mode.

8.4.2 Round Key Generation

Round Key Generation is another important module of TSV, the round key is a function of session key and number of iterations of each round.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use work.tsv_package.all;

entity key_gen is
    port (key: in STD_LOGIC_VECTOR(111 downto 0);
          round: in round_type;
          DATAOUT: out STD_LOGIC_VECTOR(255 downto 0));
end entity key_gen;

architecture top_tsv_RTL of key_gen is
    begin
        process (key, round) is
            begin
                DATAOUT <= ROUNDKEY_GEN(key, round);
            end process;
        end architecture top_tsv_RTL;
```

Figure 8.14: Top level VHDL module for round key generation of TSV



Figure 8.15: Top level entity of round key generation of TSV

Figure 8.14 shows the top level VHDL module for round key generation for TSV. Here 'key_gen' is an entity for round key generation. As stated before the round keys of each round is generated from the user given key, the length of each round key is 256-bits and user key is 112-bit. Figure 8.15 shows the top level entity of round key generation of TSV.

In port logic, 'key' is an array of 112-bit (111-0), which takes the 112-bit user encryption/decryption key. Then this key and round number is passed through process 'ROUNDKEY_GEN', this process generates the 256-bit round key for each round and it is stored in 'DATAOUT' array of 256-bit (255-0).

Key length is one of the two most important security factors of any encryption algorithm—the other one being the design of the algorithm itself. The effective key length of Triple SV is 112 bits, giving 2^{112} possible combinations. The 112-bit key is completely user defined and is provided by the user in the form of numbers of iteration that each of the n-bit levels would have while the encryption or decryption process progresses. The 112 bits of the key have been logically divided into seven 16-bit binary sequences, each of which relates to a particular n-bit level. The association is elucidated below.

- Bit number 1 to 16 form string K2, and is associated with 2-bit level.
- Bit number 17 to 32 form string K4, and is associated with 4-bit level.
- Bit number 33 to 48 form string K8, and is associated with 8-bit level.
- Bit number 49 to 64 form string K16, and is associated with 16-bit level.
- Bit number 65 to 80 form string K32, and is associated with 32-bit level.
- Bit number 81 to 96 form string K64, and is associated with 64-bit level.
- Bit number 97 to 112 form string K128, and is associated with 128-bit level.

Therefore, TSV is successfully implemented in VHDL with CBC modes of operation; round key of 128-bits is also generated by taking 112-bits input as session key. TSV is also implemented in C-programming to find the testing parameters and to compare it with RSA and previously proposed techniques. TSV is giving a much better result in avalanche ratio test and non-homogeneity test using Chi-Square values.

8.5 Analysis

TSV is implemented in both hardware and software modules. Some of the characteristics are:

- TSV encryption and decryption is done in CBC mode so it converts a block cipher to stream cipher.
- The avalanche ratio test reveals a much better result for TSV, which means if alter a few bits/bytes in session key or in plaintext than it effects or alters 99.9% bits/bytes of ciphertext.
- TSV also involves generation of 128-bits round keys from 112-bits of session key, during decryption the round keys are applied in reverse manner as that was applied during encryption.
- TSV gives much better result in non-homogeneity test using Chi-Square values that means that the ciphertext differs in large manner from plaintext.
- Algorithmic complexity of TSV is found to be $O(n^2)$.
- TSV is symmetric block cipher which means same key is used for encryption and decryption.
- TSV is also a non Feistel block cipher, which is commonly used for design of symmetric block cipher.
- In hardware implementation perspective TSV uses much less resources than that of RSA where giving better results in testing parameters.
- TSV can be used in key distribution techniques using Key Distribution Centres (KDC).

8.6 Results and Simulations

In this section some of the results of TSV are discussed and the various comparisons made with the earlier proposed technique and also with RSA. Section 8.6.1 discuss results of RTL/Hardware implementation, section 8.6.2 discuss the results of frequency distribution graph, section 8.6.3 discuss the results of Chi-Square test for non-homogeneity of source files and encrypted files, section 8.6.4 discuss the results of time complexity and section 8.6.5 discuss the results of avalanche ratio test.

8.6.1 RTL Simulation Based Result

In this section gives some of the results found after implementing the proposed technique in VHDL. This code has been simulated and synthesized in Xilinx 8.1i. The main objective is to find an efficient FPGA-based cryptographic technique for implementation in embedded systems.

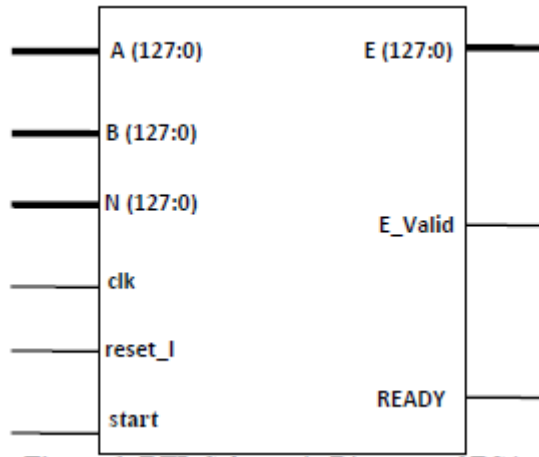


Figure 8.16: RTL diagram of RSA

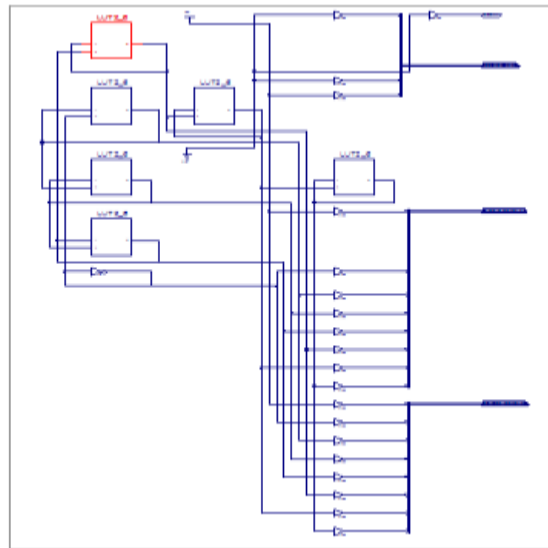


Figure 8.17: Spartan 3E RTL diagram of TPRT

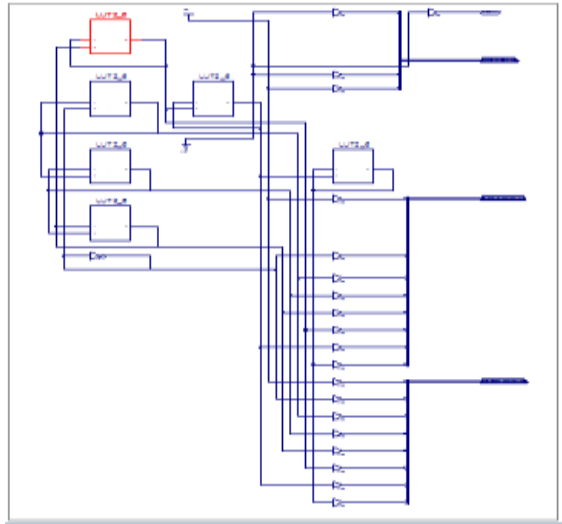


Figure 8.18: Spartan 3E RTL diagram of TMAT

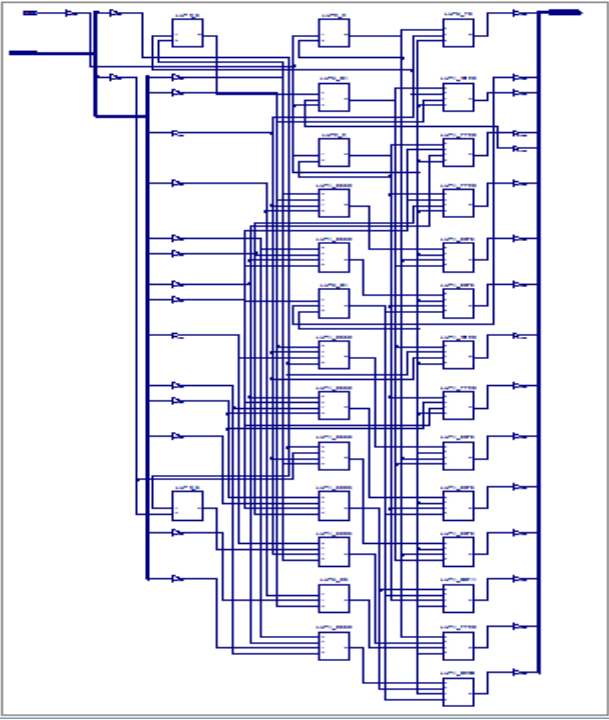


Figure 8.19: Spartan 3E schematic of ROBAST

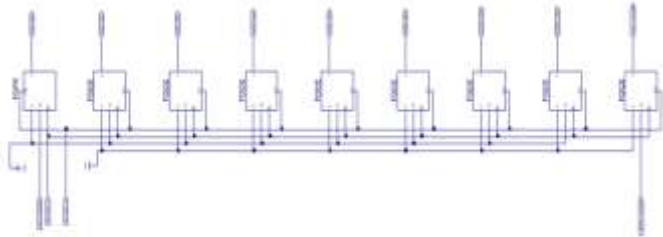


Figure 8.20: Spartan 3E RTL schematic of the main controller module of Shuffle-RAT

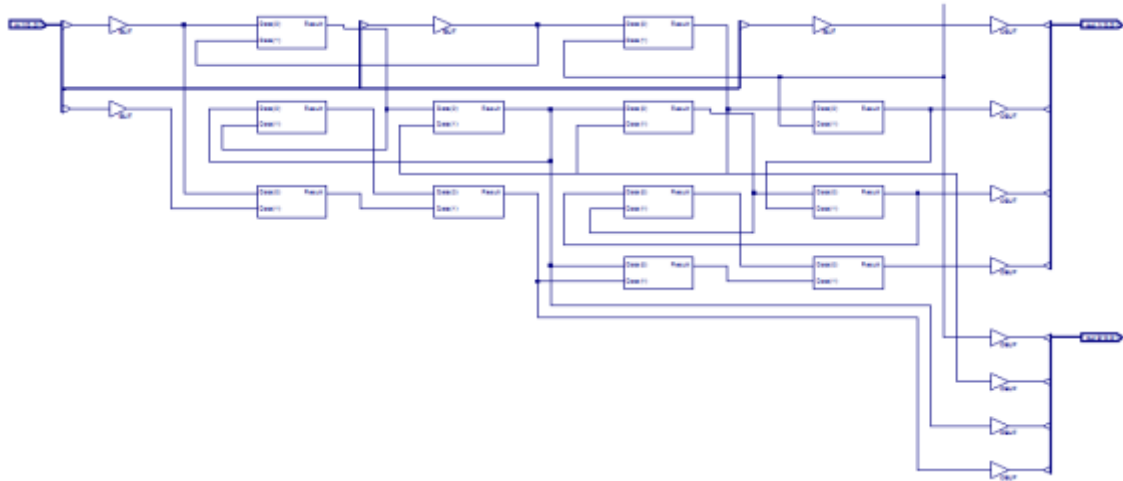


Figure 8.21: Spartan 3E RTL diagram of TSV

The design of TSV is done using VHDL and implemented in Xilinx Spartan-3E XC3S100E-5VQ100 (package: VQ100, speed grade: -5) FPGA using the ISE 8.1i design tool. Figure 8.16 shows the RTL of RSA, figure 8.17 shows the RTL of TPRT, figure 8.18 shows RTL of TMAT, figure 8.19 shows RTL of ROBAST, figure 8.20 shows RTL of SRAT and figure 8.21 shows the RTL diagram of TSV. Here 64-bit implementation timing diagram is illustrated, plaintext is of 64-bit and user encryption/decryption key is of 56-bit, the output 64-bit ciphertext is got after 450ns.

Table 8.2: HDL synthesis report (Netlist generation of RSA, TPRT, TMAT, ROBAST, SRAT and TSV)

Sr No.	Netlist Components	Number					
		RSA	TPRT	TMAT	ROBAST	SRAT	TSV
1	ROMs/RAMs	430	10	14	25	28	12
2	Adders/Subtractions	3	0	2	20	28	0
3	Registers	420	20	30	50	641	10
4	Latches	80	0	0	10	80	0
5	Multiplexers	120	0	0	10	136	0

Table 8.2 illustrates the hardware implementation analysis of TSV and its comparisons with other techniques/algorithms, namely, RSA, TPRT, TMAT, ROBAST and SRAT. This proposed technique, TSV, uses no adder/subtractions, latches and multiplexers. TSV uses 22 memory units (ROM/RAM) and 10 registers which are quite less than that of

other techniques/algorithms. Observing the above table it is seen that RSA consumes maximum of resources, then comes ROBAT followed by SRAT. TPRT, TMAT consumes the minimum resources.

Table 8.3: HDL synthesis report (Timing summary of RSA, TPRT, TMAT, ROBAST, SRAT and TSV)

Sr No.	Timing Constraint	Values					
		RSA	TPRT	TMAT	ROBAST	SRAT	TSV
1	Speed Grade	-5	-5	-5	-5	-5	-5
2	Minimum period (ns)	9.895	5.66	7.95	5.55	5.50	10.22
3	Maximum Frequency (MHZ)	101.06	101.06	101.06	101.06	101.06	101.06
4	Minimum input arrival time before clock (ns)	6.697	4.33	5.55	5.55	4.25	6.66
5	Maximum output required time after clock (ns)	4.31	3.33	4.25	4.44	3.33	5.55

Table 8.3 illustrates the entire timing summary obtained after HDL synthesis. The speed grade and maximum frequency is same as all the techniques/algorithms have been implemented in Xilinx Spartan-3E XC3S100E-5VQ100 (package: VQ100, speed grade: -5). TPRT and ROBAST gives optimal solution in terms of minimum period, minimum input arrival time and maximum output time. Though TSV doesn't give optimal results in hardware implementation but it gives best result in avalanche effect.

8.6.2 The Frequency Distribution Graph

The frequency distribution is the distribution of the all 256 ASCII characters in the respective files. This is also a cryptographic parameter which measures the degree of cryptanalysis.

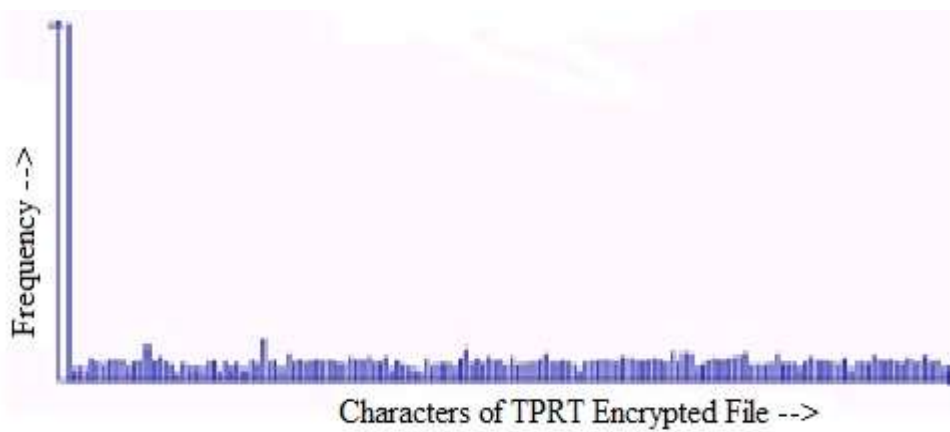
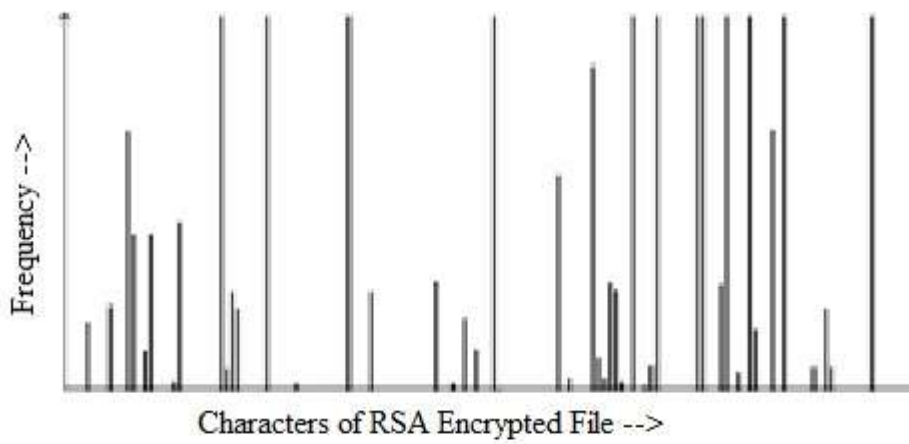
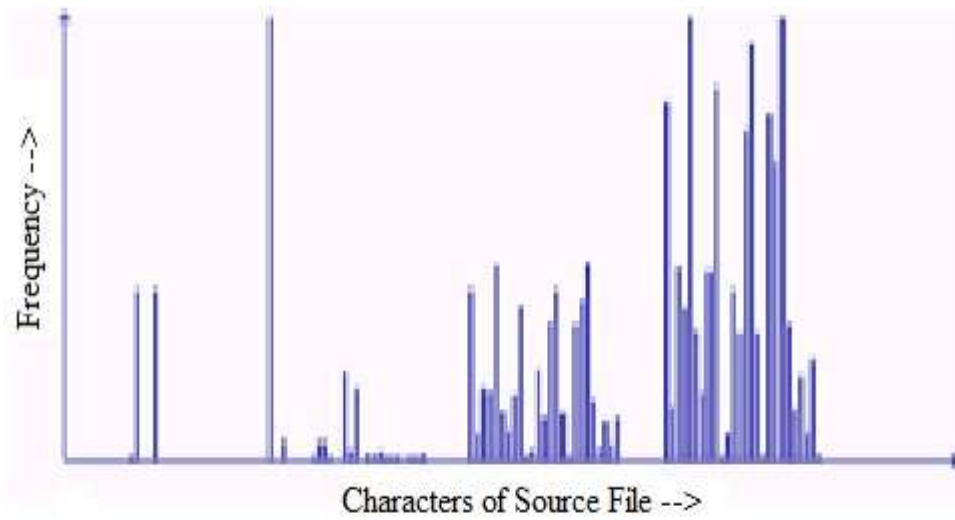


Figure 8.22: Frequency distribution graph of source, RSA encrypted and TPRT encrypted files

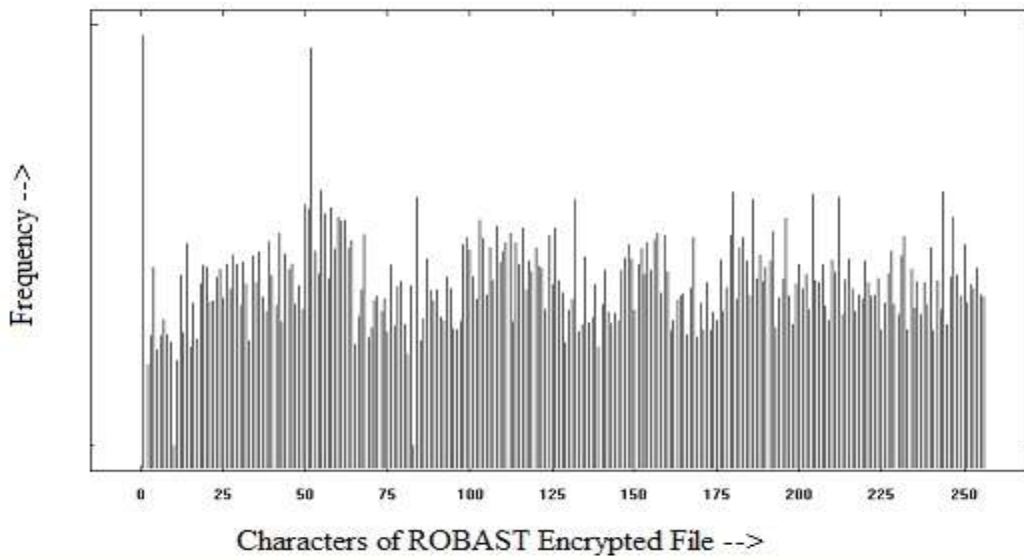


Figure 8.23: Frequency distribution graph of TMAT and ROBAST encrypted files

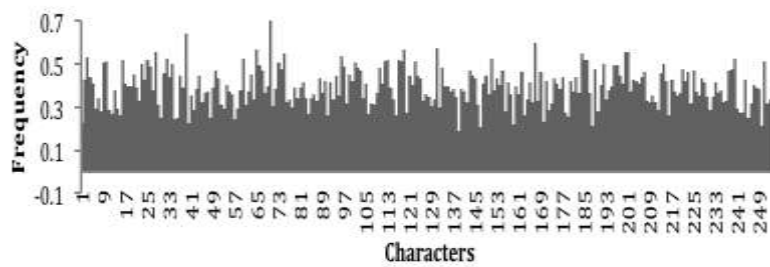


Figure 8.24: Frequency distribution graph of SRAT encrypted files

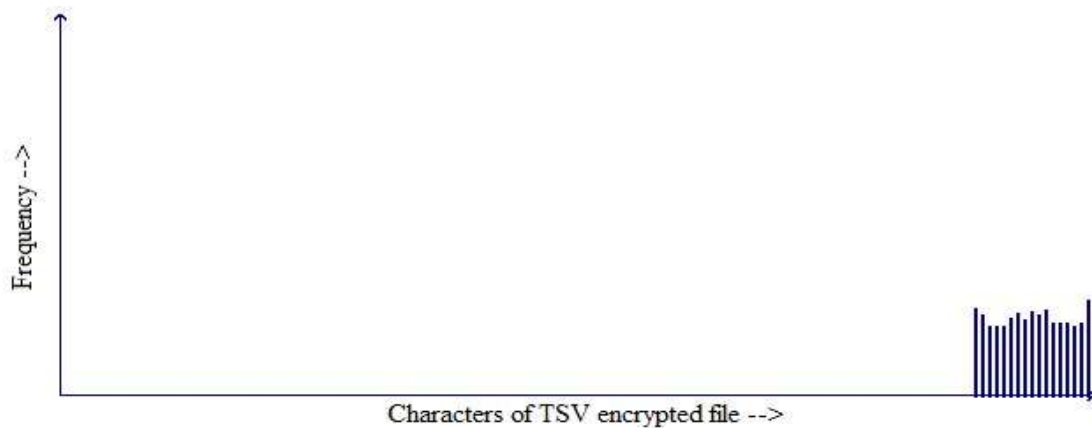


Figure 8.25: Frequency distribution graph of TSV encrypted files

The results shown are obtained after calculating the respective Frequency Distributions of the source file 'genesis.txt'. Figure 8.22 shows the frequency distribution graph of source file, RSA encrypted file and TPRT encrypted file. Figure 8.23 shows frequency distribution graph of TMAT encrypted file and ROBAST encrypted file. Figure 8.24 shows the frequency distribution graph of SRAT encrypted file and figure 8.25 shows frequency distribution graph of TSV encrypted file. It obvious that TSV is giving much better result than that of RSA. The frequencies of TSV aggregated to a specific range thus it is very difficult for cryptanalysis.

8.6.3 The Non-Homogeneity Test

Another way to analyze the technique is to test the non-homogeneity of the source and the encrypted file. The Chi-Square test has been performed for this purpose.

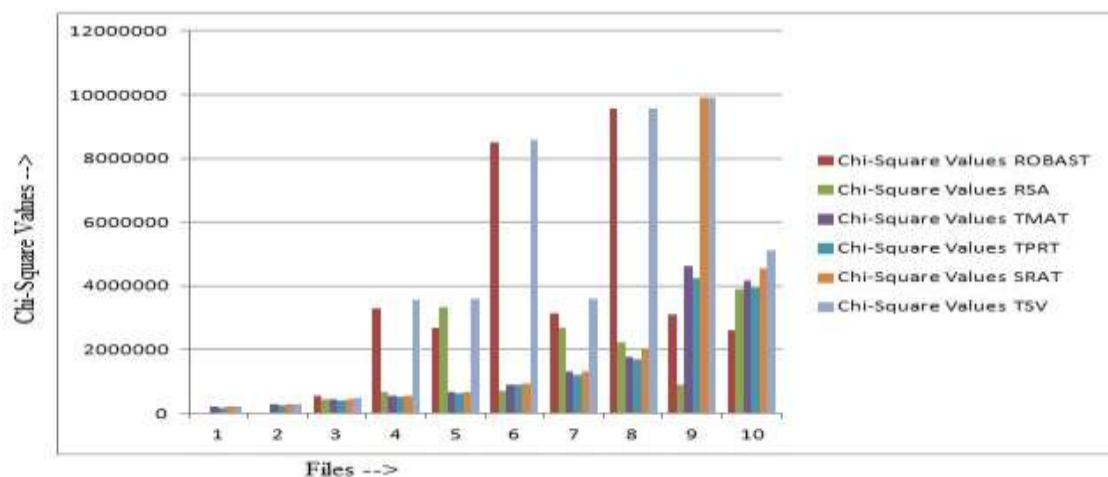


Figure 8.26: Pictorial representation of Chi-Square values

Table 8.4: Comparison of Chi-Square values of ROBAST, RSA, TPRT, TMAT, SRAT and TSV

Source File	File Size (Bytes)	Chi-Square Values					
		ROBAST	RSA	TMAT	TPRT	SRAT	TSV
license.txt	17,632	6472	5668	201530	191382	201960	210050
cs405(ei).doc	25,422	4407	2654	286025	253470	305590	306000
acread9.txt	35,121	560357	447984	440184	410735	451125	475590
deutsch.txt	47,829	3307374	685963	555220	505121	558330	3567900
genesis.txt	49,600	2679799	3318506	659045	638592	683128	3580050
pod.exe	69,981	8495675	694410	905416	896405	937565	8590100
mspaint.exe	136,463	3131296	2667664	1297256	1203665	1308890	3595000
cmd.exe	152,028	9559993	2216429	1759014	1692655	2009956	9569921
d3dim.dll	193,189	3102369	906300	4630652	4250652	9900630	9910550
clbcatq.dll	403,901	2590855	3896171	4167801	3922143	4525650	5125590

Table 8.5: Comparison of degree of freedom of ROBAST, RSA, TPRT, TMAT, SRAT and TSV

Source File	File Size (Bytes)	Degree of Freedom					
		ROBAST	RSA	TMAT	TPRT	SRAT	TSV
license.txt	17,632	253	253	255	255	253	255
cs405(ei).doc	25,422	253	253	255	255	254	255
acread9.txt	35,121	253	253	255	255	255	254
deutsch.txt	47,829	253	253	255	255	240	253
genesis.txt	49,600	253	253	255	255	255	255
pod.exe	69,981	253	253	255	255	255	255
mspaint.exe	136,463	254	254	255	255	255	254
cmd.exe	152,028	253	253	255	255	255	255
d3dim.dll	193,189	253	253	255	255	255	253
clbcatq.dll	403,901	253	253	255	255	255	255

The Chi-Square test has been performed for this purpose. Table 8.4 and figure 8.26 show the file size and the corresponding Chi-Square values for ten different files. Table 8.5 gives the degree of freedom values. The Chi-Square values for the proposed technique are

comparatively lower than those obtained by RSA. The value of degree of freedom is on an average 127. Hence the source and the corresponding encrypted files are considered to be heterogeneous. The degree of freedom is listed in table 8.5. Average Chi-Square value of TSV is 4165, RSA is 47505, TMAT is 1490214, TPRT is 1396482, ROBAST is 3343860 and Shuffle-RAT is 21076. Thus in terms of non-homogeneity TSV doesn't show optimal result.

8.6.4 The Time Complexity Analysis

In this section it will discuss time complexity analysis of the proposed technique, TSV, the time complexity analysis is broadly divided into two categories, namely encryption time and decryption time. The encryption time is the time required to convert a plaintext into a ciphertext and the decryption time is the time required to convert the ciphertext into the plaintext for a given block size and key. Here ten different sample files are taken and their complexities are noted down.

Table 8.6: Comparison of encryption time of ROBAST, RSA, TMAT, TPRT, SRAT and TSV

Source File	File Size (Bytes)	Encryption Time					
		ROBAST	RSA	TMAT	TPRT	SRAT	TSV
license.txt	17,632	0.00	0.01	0.03	0.02	0.00	0.00
cs405(ei).doc	25,422	0.01	0.06	0.00	0.00	0.01	0.00
acread9.txt	35,121	0.02	0.07	0.13	0.10	0.01	0.01
deutsch.txt	47,829	0.03	0.11	0.25	0.20	0.01	0.01
genesis.txt	49,600	0.04	0.12	0.28	0.25	0.02	0.01
pod.exe	69,981	0.04	0.12	0.39	0.35	0.02	0.02
mspaint.exe	136,463	0.06	0.20	0.44	0.40	0.03	0.02
cmd.exe	152,028	0.07	0.25	0.55	0.50	0.05	0.03
d3dim.dll	193,189	0.08	0.28	0.55	0.52	0.05	0.04
clbcatq.dll	403,901	0.08	0.32	0.67	0.60	0.05	0.05

Table 8.7: Comparison of decryption time of ROBAST, RSA, TMAT, TPRT, SRAT and TSV

Source File	File Size (Bytes)	Decryption Time					
		ROBAST	RSA	TMAT	TPRT	SRAT	TSV
license.txt	17,632	0.01	0.15	0.11	0.10	0.00	0.00
cs405(ei).doc	25,422	0.02	0.71	0.00	0.00	0.01	0.00
acread9.txt	35,121	0.03	1.15	0.13	0.10	0.01	0.01
deutsch.txt	47,829	0.03	1.36	0.15	0.11	0.01	0.01
genesis.txt	49,600	0.04	1.61	0.25	0.20	0.02	0.02
pod.exe	69,981	0.04	1.86	0.39	0.35	0.02	0.02
mspaint.exe	136,463	0.05	2.71	0.48	0.40	0.02	0.03
cmd.exe	152,028	0.06	3.34	0.52	0.42	0.05	0.03
d3dim.dll	193,189	0.07	3.73	0.60	0.50	0.05	0.04
clbcatq.dll	403,901	0.08	4.25	0.65	0.55	0.05	0.05

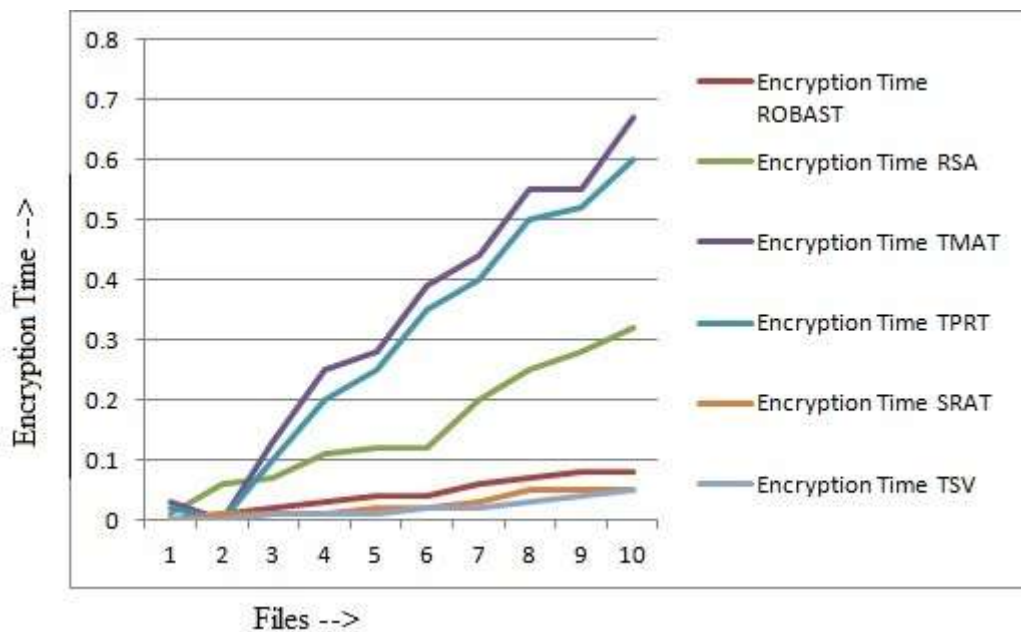


Figure 8.27: Pictorial representation of encryption time against file size

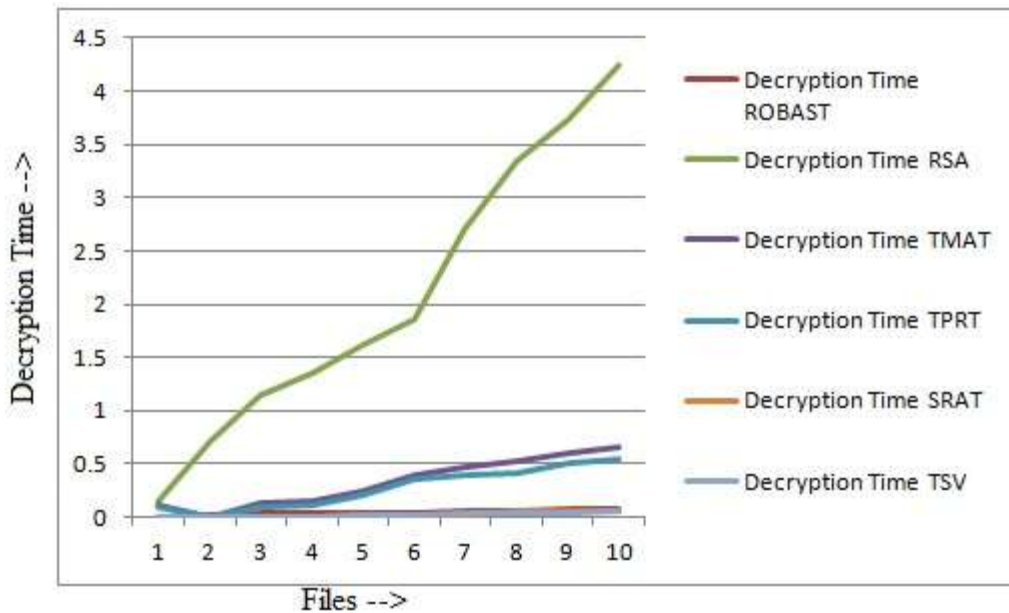


Figure 8.28: Pictorial representation of decryption time against file size

Table 8.6 and table 8.7 illustrate the encryption time and decryption time of the proposed techniques with RSA. This section compares the time complexity of the TSV with that of RSA by taking the encryption and decryption times into consideration. The graphical analysis of the encryption and decryption time of the Triple SV and RSA has been depicted in Figure 8.27 and figure 8.28 respectively. The time complexity of the proposed technique is well comparable to RSA. It is also observed that the time complexity of the proposed technique, TSV, is quite less than the previously proposed techniques, ROBAST, TMAT, TPRT and Shuffle-RAT. Thus TSV gives optimal solution in respect to time complexity analysis taking account both encryption time and decryption time.

8.6.5 The Avalanche Ratio Test

Avalanche Effect refers to a desirable property of any cryptographic algorithm where, if an input is changed slightly (for example, flipping a single bit) the output changes significantly (e.g., more than half the output bits flip).

Table 8.8: Comparison of avalanche ratio of ROBAST, RSA, TPRT, TMAT, SRAT and TSV encrypted files

Source File	File Size (Bytes)	Avalanche Ratio of ROBAST encrypted files (in %)	Avalanche Ratio of RSA encrypted files (in %)	Avalanche Ratio of TPRT encrypted file (in %)	Avalanche ratio of TMAT encrypted file (in %)	Avalanche ratio of SRAT encrypted file (in %)	Avalanche ratio of TSV encrypted file (in %)
license.txt	17,632	71.90	58.0	77.7	80.8	91.5	99.9
cs405(ei).doc	25,422	99.69	60.0	80.0	85.5	90.5	99.8
acread9.txt	35,121	99.93	75.0	88.8	90.0	98.0	99.9
deutsch.txt	47,829	99.96	78.9	89.0	91.5	99.5	99.7
genesis.txt	49,600	97.72	80.9	87.0	94.7	99.9	99.6
pod.exe	69,981	77.00	58.0	77.0	80.0	99.9	99.8
mspaint.exe	136,463	98.22	58.9	76.0	80.0	98.0	99.9
cmd.exe	152,028	99.97	67.0	77.0	80.0	97.0	99.8
d3dim.dll	193,189	99.98	67.9	82.9	85.0	97.5	99.7
clbcataq.dll	403,901	75.55	68.0	88.5	90.5	99.0	99.9

Table 8.8 compares the avalanche effect ratio for TSV, RSA and previous proposed techniques/algorithm and which are obtained after calculating the respective Avalanche Effect by making a change of a few (approx 3) characters in each file. It is observed that the proposed technique is showing an average avalanche ratio percentage of 99.7% which is way higher than that obtained using RSA. High avalanche ratio ensures higher security from brute force attack. It is also observed that this avalanche ratio test of TSV is better than Shuffle-RAT, TPRT, TMAT and ROBAST.

8.7 Discussions

The cryptographic technique, Triple SV is a symmetric block cipher using a 256-bit block and 112-bit key. From the above discussions it can be inferred that Triple SV is potentially a promising algorithm which can find its efficient implementation in different fields. Triple SV has a way better Avalanche Effect than any of the other existing algorithms and hence can be incorporated in the process of encryption of any plaintext. The high avalanche ratio and a key size of 112 bits ensure sound security from brute force attacks. The implementation in CBC mode ensures low predictability and tougher cryptanalysis. Even the time complexity of the proposed algorithm is considerably viable and even better than RSA at many instances. The proposed model(s) and conclusion(s) of this thesis is given in next part of this thesis.

Chapter 9

Modified Forward Backward Overlapped Modulo Arithmetic Technique (MFBOMAT)

9.1 Introduction

In this chapter, a new Cryptosystem based on block cipher has been proposed where the encryption is done through Modified Forward Backward Overlapped Modulo Arithmetic Technique (MFBOMAT). The original message is considered as a stream of bits, which is then divided into a number of blocks, each containing n bits, where n is any one of 2, 4, 8, 16, 32, 64, 128, 256. The first and last blocks are then added where the modulus of addition is 2^n . The result replaces the last block (say N th block), first block remaining unchanged (Forward mode). In the next attempt the second and the N th block (the changed block) are added and the result replaces the second block (Backward mode). Again the second (the changed block) and the $(N-1)$ th block are added and the result replaces the $(N-1)$ th block (Forward mode). The modulo addition has been implemented in a very simple manner where the carry out of the MSB is discarded to get the result. The technique is applied in a cascaded manner by varying the block size from 2 to 256. The whole technique has been implemented by using a modulo subtraction technique for decryption.

In the proposed scheme the source file is taken as input as streams of binary bits. For its implementation the stream size to be 512 bits have been taken though the scheme may be implemented for larger stream sizes also. The input stream, S , is first broken into a number of blocks, each containing n bits ($n=2^k$, $k=1,2,3,\dots,8$) so that $S = B_1B_2B_3\dots B_m$ where $m=512/n$. Starting from the MSB, the blocks are paired as (B_1, B_m) , (B_2, B_{m-1}) , (B_3, B_{m-2}) and so on. So there is a common member in any two non-adjacent block-pairs, i.e. the block-pairs are overlapping and hence the name given to the technique. The FBOMAT operation is applied to each pair of blocks. The process is repeated, each time increasing the block size till $n=256$. The proposed scheme has been implemented by using the reverse technique, i.e. modulo subtraction technique, for decryption.

Section 9.2 discussed the algorithm of MFBOMAT with a block level diagram, section 9.3 gives a detailed example of encryption and decryption process, section 9.4 discussed the implementation issues with key generation, section 9.5 gives a brief analysis, section 9.6 discussed the results obtained based on implementation and a brief discussions are given in section 9.7.

9.2 The Algorithm of MFBOMAT

After chapping the input stream into blocks of 2 bits each and pairing the blocks as explained in Section 1, the following operations are performed starting from the most significant side:

- **Round 1:** The whole plaintext is divided into finite number of blocks of 2-bit block size. Then get a number of blocks $B_1, B_2, B_3, \dots, B_n$. B_1 is modulo added to B_n and the result replaces the B_n , then B_n is modulo added to B_2 , result replacing the B_2 , then B_2 is modulo added to B_{n-1} and the result replaces the B_{n-1} . Then B_{n-1} is modulo added to B_3 and the result replaces the B_3 and it continues in similar manner. In each pair of blocks, the first member of the pair is added to the second member where the modulus of addition is 2^n for block size n . Therefore for 2-bit blocks, the modulus of addition will be 4. This round is repeated for a finite number of times and the number of iterations will form a part of the session key as discussed in section 9.4.
- **Round 2:** The same operation as in Round 1 is performed with block size 4. In the next round the block size of 8-bits is taken and the same operation is repeated. In this fashion several rounds are completed till it reaches **Round 8** where the block size is 256 and get the encrypted bit-stream. The operations of the non adjacent block-pairs increase the complexity of the algorithm resulting in the enhancement of security.
- During decryption, the reverse operation, i.e. modulo subtraction, is performed instead of modulo addition, starting from the blocks $B_{n/2}$ and $((B_n)/2) + 1$ and then $((B_n)/2)$ and $((B_n)/2) + 2$ and then $((B_n)/2) - 1$ and $((B_n)/2) + 2$. The process continues until all the remaining blocks are decrypted. Where the n th block is the last block of the 512-bits stream.

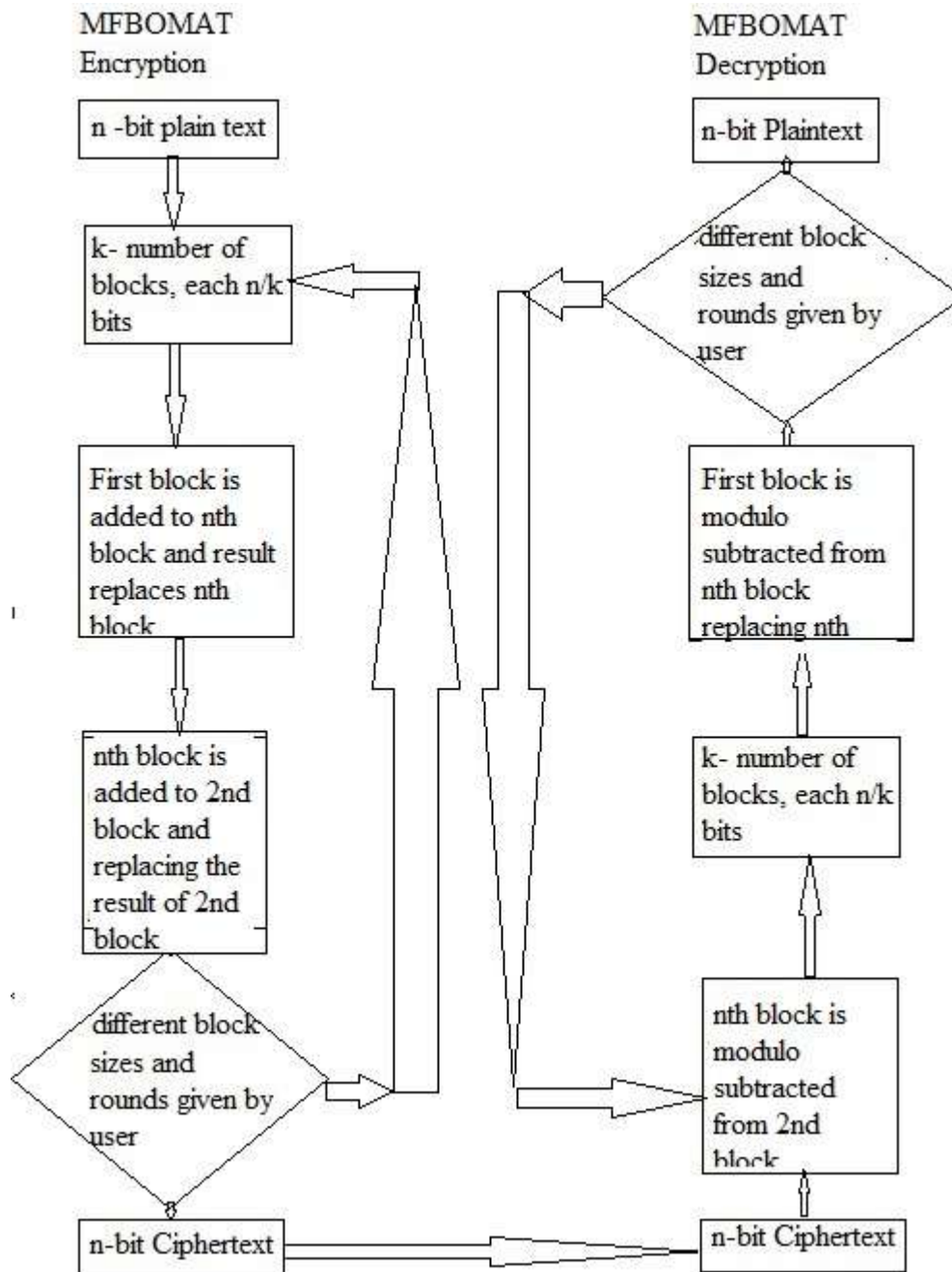


Figure 9.1: Block diagram of MFBOMAT

Figure 9.1 gives the block diagram of MFBOMAT, The whole plaintext is divided into finite number of blocks of 2-bit block size. Then get a number of blocks $B_1, B_2, B_3, \dots, B_n$. B_1 is modulo added to B_n and the result replaces the B_n , then B_n is modulo added to B_2 , result replacing the B_2 , then B_2 is modulo added to B_{n-1} and the result replaces the B_{n-1} . Then B_{n-1} is modulo added to B_3 and the result replaces the B_3 and it continues in similar manner. In each pair of blocks, the first member of the pair is added to the second member where the modulus of addition is $2n$ for block size n . Therefore for 2-bit blocks, the modulus

of addition will be 4. This round is repeated for a finite number of times and the number of iterations will form a part of the session key as discussed in section 9.4.

9.2.1 The Modulo Addition

An alternative method for modulo addition is proposed here to make the calculations simple. The need for computation of decimal equivalents of the blocks is avoided here since it will get large decimal integer values for large binary blocks. The method proposed here is just to discard the carry out of the MSB after the addition to get the result. For example, if add 1101 and 1001 and get 10110. In terms of decimal values, $13+9=22$. Since the modulus of addition is 16 (24) in this case, the result of addition should be 6 ($22-16=6$). Discarding the carry from 10110 is equivalent to subtracting 10000 (i.e. 16 in decimal). So the result will be 0110, which is equivalent to 6 in decimal. The same is applicable to any block size.

9.3 Example

Although the proposed scheme is applied to a 512-bit input stream, for the sake of brevity, consider a stream of 16 bits, say $S = 1101001100011011$ each round is performed only once to make the process simple for understanding.

9.3.1 The Encryption

Round 1: Block size = 2, number of blocks = 8

The blocks are $B1 = 11$, $B2 = 11$, $B3 = 01$, $B4 = 01$, $B5 = 00$, $B6 = 10$, $B7 = 01$ and $B8 = 10$.

The MFBOMAT is applied to these eight blocks

Output

11	11	01	01	00	10	01	10
B1	B2	B3	B4	B5	B6	B7	B8

(B4, B6) mod4, Change B4

Input

11	11	01	01	00	10	01	10
B1	B2	B3	B4	B5	B6	B7	B8

Output

11	11	01	01	01	10	01	10
B1	B2	B3	B4	B5	B6	B7	B8

(B4, B5) mod4, Change B5

Round 2: Block size = 4, number of blocks = 4

Input

1111	0101	0110	0110
B1	B2	B3	B4

Output:

1111	0101	0110	0101
B1	B2	B3	B4

(B1, B4) mod16, Change B4

Input

1111	0101	0110	0101
B1	B2	B3	B4

Output

1111	1010	0110	0101
B1	B2	B3	B4

(B2, B4) mod16, Change B2

Input

1111	1010	0110	0101
B1	B2	B3	B4

Output

1111	1010	0000	0101
B1	B2	B3	B4

(B2, B3) mod16, Change B3

Round 3: Block size = 8, number of blocks = 2

Input

11111010	00000101
B1	B2

Output

11111010	11111111
B1	B2

(B1, B2) mod 256, Change B2

Since it has been considered only a 16-bit stream cannot be proceed further. The output from Round 3, say S', is the encrypted stream, i.e. S' = 1111101011111111. For decryption the opposite method i.e. modular subtraction is used to get back the original bit stream in S.

9.3.2 The Decryption

For decryption the opposite method i.e. modular subtraction is used to get back the original bit stream in S.

Round 1:Block size=8, number of blocks =2

Input	
1111010	1111111
B1	B2
Output	
1111010	0000101
B1	B2
(B1, B2) mod 256, Change B2	

Round 2:Block size=4, number of blocks=4

Input			
1111	1010	0000	0101
B1	B2	B3	B4

Output			
1111	1010	0110	0101
B1	B2	B3	B4

(B2, B3) mod16, Change B3

Input			
1111	1010	0110	0101
B1	B2	B3	B4

Output			
1111	0101	0110	0101
B1	B2	B3	B4

(B2, B4) mod16, Change B2

Input			
1111	0101	0110	0101
B1	B2	B3	B4

Output			
1111	0101	0110	0110
B1	B2	B3	B4

(B1, B4) mod4, Change B4

Round 3: Block size=2, number of blocks =8

Input

11	11	01	01	01	10	01	10
B1	B2	B3	B4	B5	B6	B7	B8

Output

11	11	01	01	00	10	01	10
B1	B2	B3	B4	B5	B6	B7	B8

(B4, B5) mod4, Change B5

Input

11	11	01	01	00	10	01	10
B1	B2	B3	B4	B5	B6	B7	B8

Output

11	11	01	11	00	10	01	10
B1	B2	B3	B4	B5	B6	B7	B8

(B4, B6) mod4, Change B4

Input

11	11	01	11	00	10	01	10
B1	B2	B3	B4	B5	B6	B7	B8

11	11	01	11	00	01	01	10
B1	B2	B3	B4	B5	B6	B7	B8

(B3, B6) mod4, Change B6

Input

11	11	01	11	00	01	01	10
B1	B2	B3	B4	B5	B6	B7	B8

Output

11	11	00	11	00	01	01	10
B1	B2	B3	B4	B5	B6	B7	B8

(B3, B7) mod4, Change B3

Input

11	11	00	11	00	01	01	10
B1	B2	B3	B4	B5	B6	B7	B8

Output

11	11	00	11	00	01	10	10
B1	B2	B3	B4	B5	B6	B7	B8

(B2, B7) mod4, Change B7

Input

11	11	00	11	00	01	10	10
B1	B2	B3	B4	B5	B6	B7	B8

Output

11	01	00	11	00	01	10	10
B1	B2	B3	B4	B5	B6	B7	B8

(B2, B8) mod4, Change B2

Input

11	01	00	11	00	01	10	10
B1	B2	B3	B4	B5	B6	B7	B8

Output

11	01	00	11	00	01	10	11
B1	B2	B3	B4	B5	B6	B7	B8

(B1, B8) mod4, Change B8

The decrypted bit stream: S''=1101001100011011. So S=S''.

9.4 Implementation and Key Generation

The technique executes modulo addition between two blocks, the first iteration performs in forward basis and then backward operation is performed. Next, final permutation is done to get the final cipher text.

```
library std;
library ieee;
use ieee.std_logic_arith.all;
use work.pack.all;
use std.textio.all;
use ieee.std_logic_TEXTIO.all;
entity MFBOMAT_VHDL is
Port ( input_bits : in BIT_VECTOR (16 downto 1);
output_bits : out BIT_VECTOR (16 downto 1); key_bits : in
BIT_VECTOR (8 downto 1);
EN_DN : in BIT);
end MFBOMAT_VHDL;
architecture Behavioral of MFBOMAT_VHDL is
begin
process(EN_DN)
variable varin_bits,varout_bits: bit_vector(16 downto 1);
begin
if (EN_DN='1')then varin_bits:=input_bits;
AA: MFBOMAT_Encryption(varin_bits,key_bits,varout_bits);
output_bits<=varout_bits;
else
BB: MFBOMAT_Decryption(varin_bits,key_bits,varout_bits);
output_bits<=varout_bits;
end if;
end process;
end Behavioral;
```

Figure 9.2: MFBOMAT entity and its function

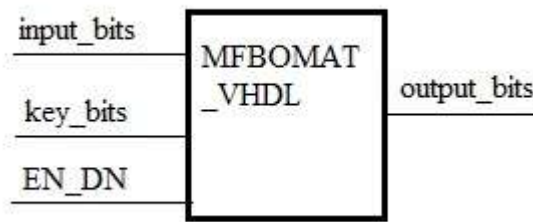


Figure 9.3: Top level RTL design of MFBOMAT

FPGA based implementation of the technique has been done in VHDL. In both implementation, the technique takes input from file as a source stream and encryption is performed. The cipher text generated is finally written in another file. The data blocks (8, 16, 32, 64, 128 and 256-bits) from the input file have been stored in array. Then encryption is performed and also stored in array. The reading and writing of data from and in file is based on 8-bit ASCII codes. Xilinx ISE 8.1i software has been used for writing codes in VHDL.

Figure 9.2 gives the implementation of MFBOMAT entity and its function. The encryption/decryption entity input bit vector (16-bit), output bit vector (16-bit), key bit vector (8-bit) and EN_DN signal. If EN_DN = 1 then encryption is performed else decryption is performed. During encryption the input bit vector of 16-bits is the plaintext and output 16-bit vector is the ciphertext where as EN_DN value is '1'. During decryption the input bit vector of 16-bits is the ciphertext and the output 16-bit vector is the plaintext where as EN_DN value is '0'. Figure 9.3 shows the top RTL diagram of MFBOMAT.

When EN_DN = 1, the 'MFBOMAT_Encryption' function is called with the parameters, 'varin_bits' which is the plaintext, 'varout_bits' which is the ciphertext, both of these are of 16-bits and third parameter is the 'key_bits' which is the session key of the encryption of 8-bits. When EN_DN = 0, the 'MFBOMAT_Decryption' function is called with the parameters, 'varin_bits' which is the ciphertext, 'varout_bits' which is the plaintext, both of these are of 16-bits and third parameter is the 'key_bits' which is the session key of the decryption of 8-bits. This code is written in VHDL using behavioral model of coding. The 'MFBOMAT_VHDL' entity in this coding has three ports, 'input_bits' of IN type of bit vector of 16-bits, 'output_bits' of OUT type of bit vector of 16-bits, 'key_bits' of IN type of bit vector of 16-bits and 'EN_DN' bit of IN type. 'Behavioral' is the architecture of the entity 'MFBOMAT_VHDL', this architecture contains a process which is called on the signal 'EN_DN' that is whenever there is a signal in 'EN_DN' this process is called. This process

contains two functions, ‘MFBOMAT_Encryption’ and ‘MFBOMAT_Decryption’. These two functions are called according to the value of signal bit ‘EN_DN’ which is already discussed. The implementation here is both functional and files type. These means that the code can be implemented in Xilinx FPGA and the simulation takes the input from a text file and the output is written into another text file. There are various libraries are used, library ‘std’ and library ‘ieee’, it is important to note that library ‘ieee.std_logic_TEXTIO.all’ is used for the implementation of text file reading and writing. Figure 9.2 gives the main MFBOMAT entity coded in VHDL.

Section 9.4.1 deals with the key generation process, section 9.4.2 illustrates an example.

9.4.1 The Key Generation Process of MFBOMAT

In this section key generation process has been illustrated, the session key is 128-bits for generalized MFBOMAT implementation.

Table 9.1: Representation of number of iterations in each round by bits, the key generation for MFBOMAT

Round	Block Size	Number of Iterations	
		Decimal	Binary
8.	256	50021	1100001101100101
7.	128	49870	1100001011001110
6.	64	48950	1011111100110110
5.	32	44443	1010110110011011
4.	16	46250	1011010010101010
3.	8	4321	0001000011100001
2.	4	690	0000001010110010
1.	2	72	0000000001001000

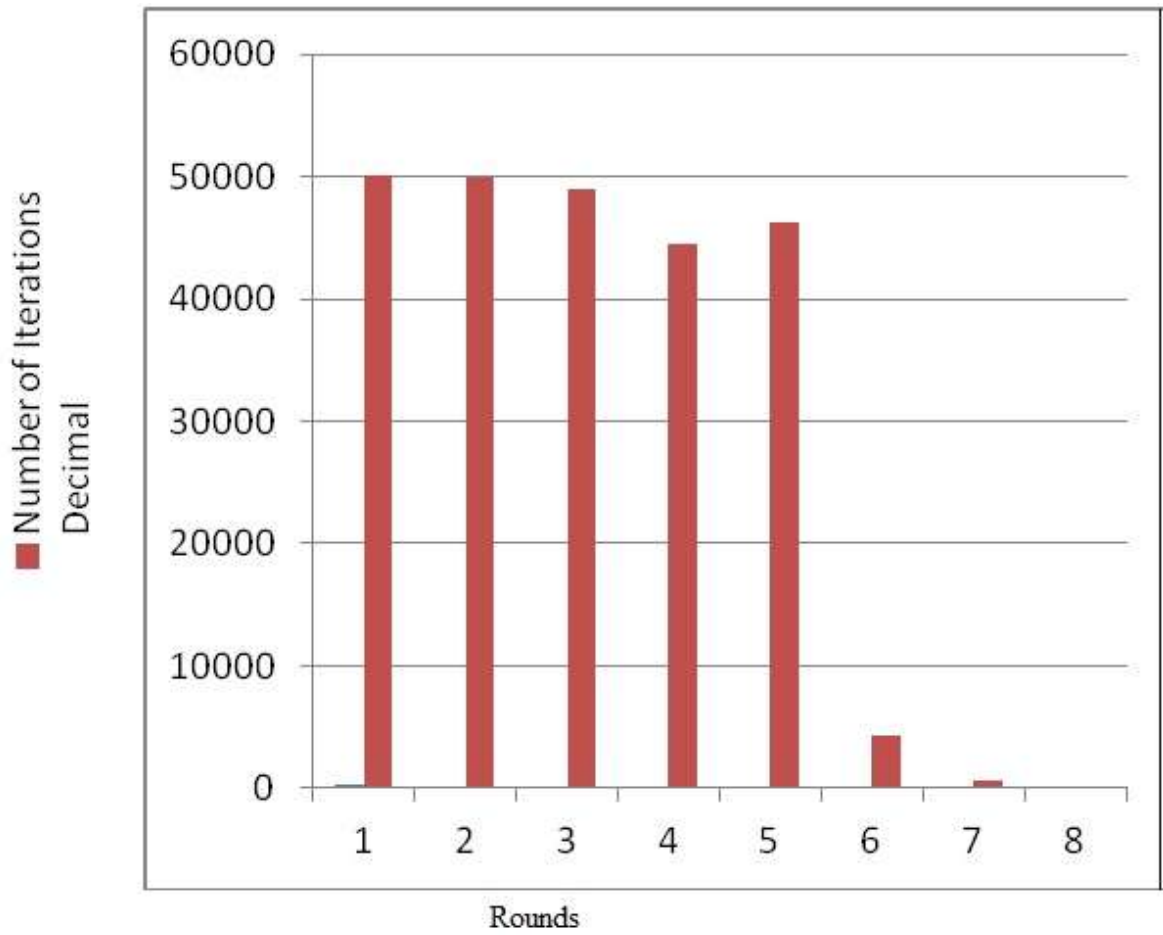


Figure 9.4: Graphical representation of key generation of MFBOMAT

The key generation process depends on block size, iteration of each block and final permutation performed. Thus, in the proposed scheme, eight rounds have been considered, each for 2, 4, 8, 16, 32, 64, 128, and 256 block size. As mentioned in each round is repeated for a finite number of times and the number of iterations will form a part of the encryption-key. Although the key may be formed in many ways, for the sake of brevity it is proposed to represent the number of iterations in each round by a 16-bit binary string. The binary strings are then concatenated to form a 128-bit key for a particular key. Table 9.1 gives the key generation process and the same is shown graphically in figure 9.4. For the block size of 2-bits are considering 72 rounds, for block size of 4-bits are considering 690 rounds and so on and finally for block size of 256-bits 50021 rounds have been considered for encryption. Since the technique is symmetric block cipher so for decryption same number of rounds will be required. These numbers of rounds have been considered in binary value, for each block size the number of rounds is considered in 16-bits of binary value. So there is eight block sizes and their corresponding eight 16-bits rounds, the key is formed by concatenating all the 16-

bits binary values. Therefore, the size of the session key proposed here is $16 \times 8 = 128$ -bits, which is now a day's considered the secure key length.

An example of key generation is illustrated in section 9.4.2. Section 9.4.3 describes the modulo addition used in MFBOMAT, which is an important operation in the technique and should be taken into account while forming the session key.

9.4.2 An Example of Key Generation

Consider a particular session where the source file is encrypted using iterations for block sizes 2, 4, 8, 16, 32, 64, 128, and 256 bits, respectively. Table 9.1 shows the corresponding binary value for the number of iterations in each round. If consider the block size of 256-bits then the binary value of round is '1100001101100101', for block size of 128-bit the binary value of round is '1100001011001110' and so on finally for block size of 2-bits the binary value of round is '0000000001001000'. These eight 16-bits binary strings are concatenated together to form the 128-bit binary string, which is given below.

- 110000110110010111000010110011101011111100110110101011011001101110110110010101010000100001110000100000010101100100000000001001000

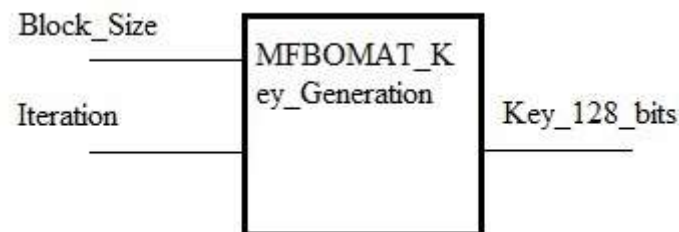


Figure 9.5: Session key generation of MFBOMAT

This 128-bit binary string will be the encryption-key for this particular session. During decryption, the same key is taken to iterate each round of modulo-subtraction for the specified number of times and reverse permutation. Figure 9.5 shows the top level RTL diagram of session key generation of MFBOMAT.

9.5 Analysis

Some of the analyses of this technique are as follows:-

- The algorithmic complexity of MFBOMAT is $O(n^2)$.
- This technique incorporates forward and backward mode of encryption.
- This technique incorporates two rounds, round 1 for forward mode of encryption and round 2 is for backward mode of encryption.
- Decryption is same as encryption where the round keys are provided in reverse order.
- The modulo addition is incorporated as a main functional block of the technique.
- 128-bit key is proposed for encryption and decryption using MFBOMAT.

9.6 Results and Simulations

This section will discuss some of the results of TSV and the various comparisons made with the proposed techniques and RSA. Section 9.6.1 discuss results of RTL/Hardware implementation, section 9.6.2 discuss the results of frequency distribution graph, section 9.6.3 discuss the results of Chi-Square test for non-homogeneity of source files and encrypted files, section 9.6.4 discuss the results of time complexity and section 9.6.5 discuss the results of avalanche ratio test.

9.6.1 RTL Simulation Based Result

In this section some of the results found on implementing of the proposed technique in VHDL have been given. The code has been simulated and synthesized in Xilinx 8.1i. The main objective is to find an efficient FPGA-based cryptographic technique for implementation in embedded systems.

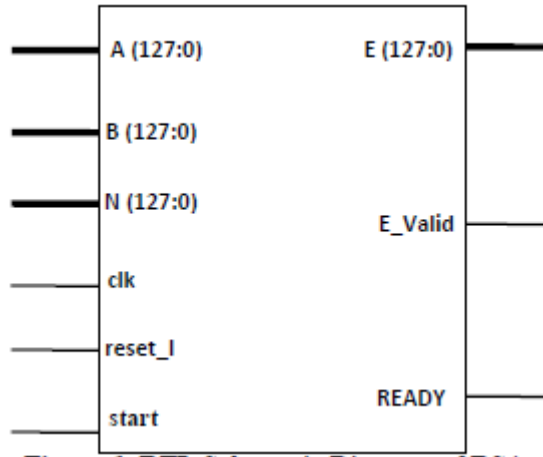


Figure 9.6: RTL diagram of RSA

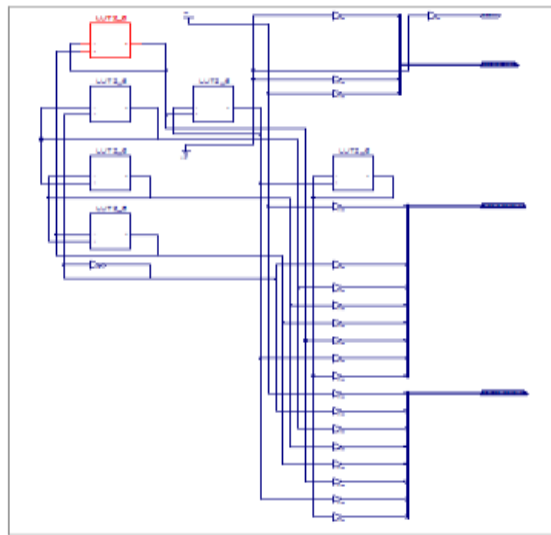


Figure 9.7: Spartan 3E RTL diagram of TPRT

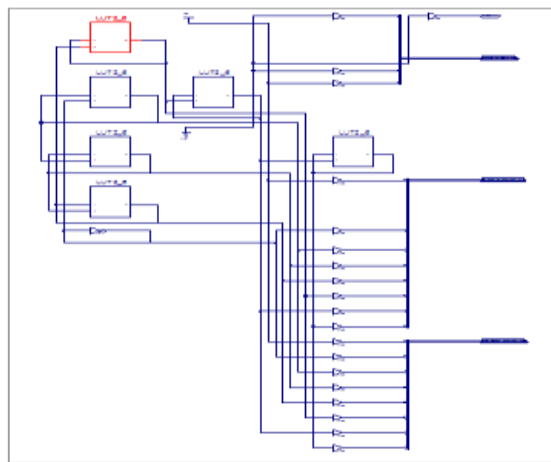


Figure 9.8: Spartan 3E RTL diagram of TMat

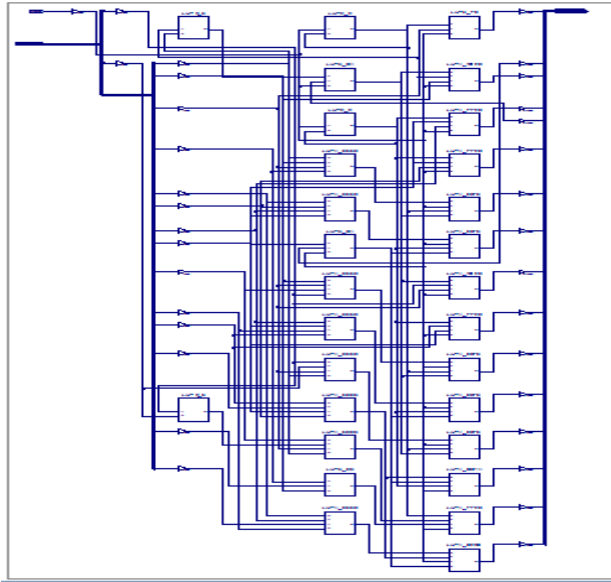


Figure 9.9: Spartan 3E schematic of ROBAST

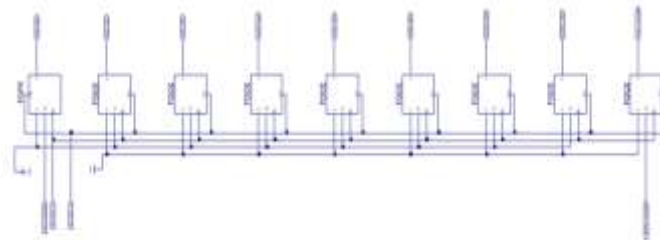


Figure 9.10: Spartan 3E RTL schematic of the main controller module of Shuffle-RAT

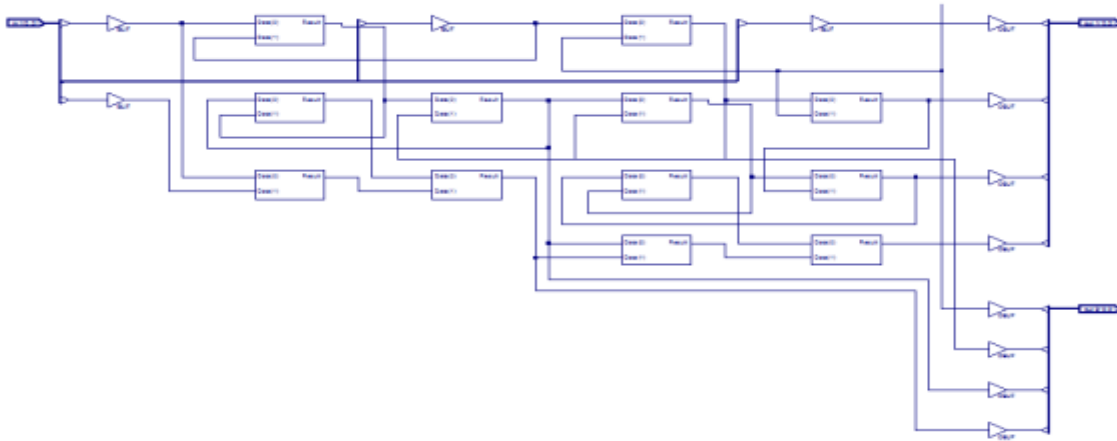


Figure 9.11: Spartan 3E RTL diagram of TSV

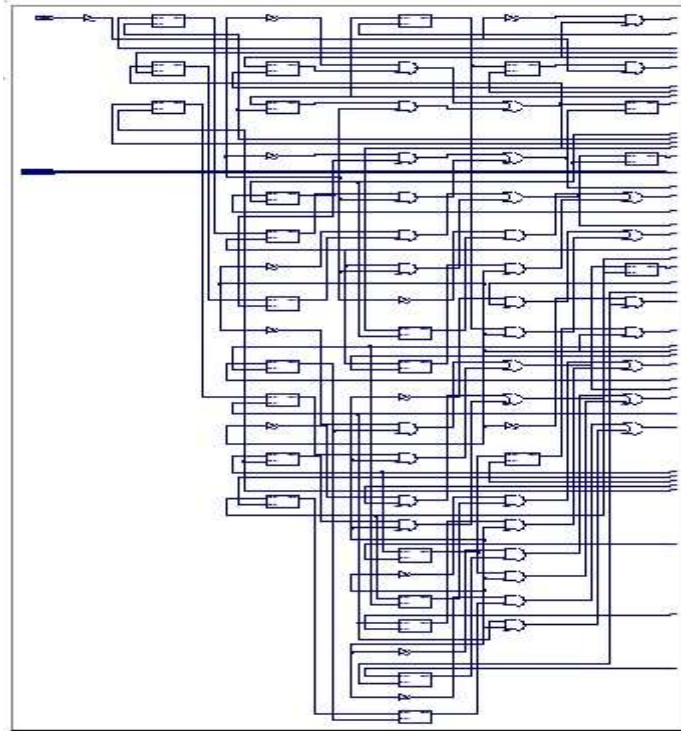


Figure 9.12: Spartan 3E RTL diagram of MFBOMAT

The design of MFBOMAT is done using VHDL and implemented in Xilinx Spartan-3E XC3S100E-5VQ100 (package: VQ100, speed grade: -5) FPGA using the ISE 8.1i design tool. Figure 9.6 shows the RTL of RSA, figure 9.7 shows the RTL of TPRT, figure 9.8 shows RTL of TMAT, figure 9.9 shows RTL of ROBAST, figure 9.10 shows RTL of SRAT, figure 9.11 shows the RTL diagram of TSV and figure 9.12 shows RTL diagram of MFBOMAT. Here 256-bit implementation timing diagram is illustrated, plaintext is of 256-bit and user encryption/decryption key is of 128-bit, the output 256-bit ciphertext is got after 450ns.

Table 9.2: HDL synthesis report (Netlist generation of RSA, TPRT, TMAT, ROBAST, SRAT, TSV and MFBOMAT)

Sr No.	Netlist Components	Number						
		RSA	TPRT	TMAT	ROBAST	SRAT	TSV	MFBOMAT
1	ROMs/RAMs	430	10	14	25	28	12	09
2	Adders/Subtractions	3	0	2	20	28	0	15
3	Registers	420	20	30	50	641	10	10
4	Latches	80	0	0	10	80	0	0
5	Multiplexers	120	0	0	10	136	0	0

Table 9.2 illustrates the hardware implementation analysis of MFBOMAT and its comparisons with other techniques/algorithms, namely, RSA, TPRT, TMAT, ROBAST, SRAT and TSV. This technique uses 15 adder/subtractions and no latches and multiplexers. MFBOMAT uses 09 memory units (ROM/RAM) and 10 registers which are quite less than that of other techniques/algorithms. Observing the above table it is seen that RSA consumes maximum of resources, then comes ROBAT followed by SRAT. TPRT, TMAT and MFBOMAT consume the minimum resources.

Table 9.3 illustrates the entire timing summary obtained after HDL synthesis. The speed grade and maximum frequency is same as all the techniques/algorithms have been implemented in Xilinx Spartan-3E XC3S100E-5VQ100 (package: VQ100, speed grade: -5).

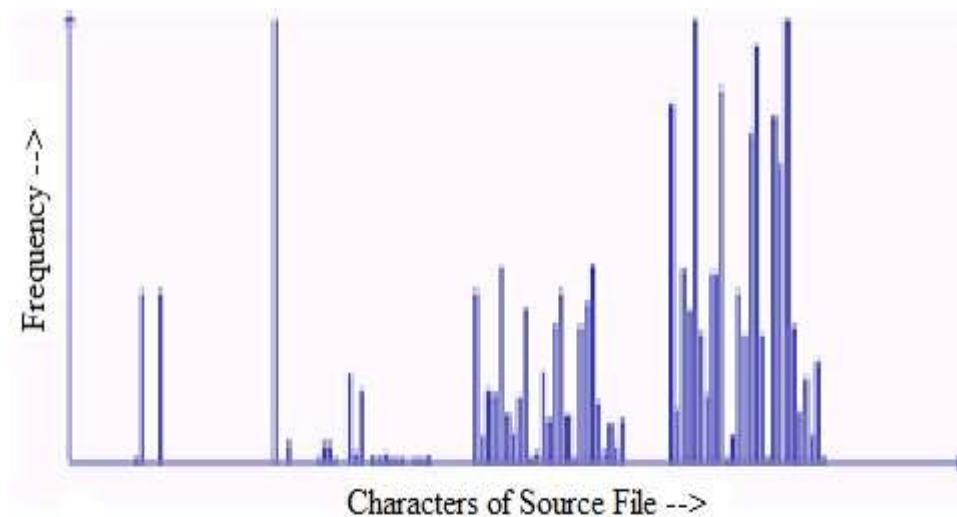
Table 9.3: HDL synthesis report (Timing summary of RSA, TPRT, TMAT, ROBAST, SRAT and TSV)

Sr No.	Timing Constraint	Values						
		RSA	TPRT	TMAT	ROBAST	SRAT	TSV	MFBOMAT
1	Speed Grade	-5	-5	-5	-5	-5	-5	-5
2	Minimum period (ns)	9.895	5.66	7.95	5.55	5.50	10.22	4.99
3	Maximum Frequency (MHZ)	101.06	101.06	101.06	101.06	101.06	101.06	101.06
4	Minimum input arrival time before clock (ns)	6.697	4.33	5.55	5.55	4.25	6.66	4.20
5	Maximum output required time after clock (ns)	4.31	3.33	4.25	4.44	3.33	5.55	3.30

MFBOMAT is obtained minimum period of 4.99ns followed by RSA 9.89ns, TPRT 5.66ns, TMAT 7.95ns, ROBAST 5.55ns, SRAT 5.50ns and TSV 10.22ns. MFBOMAT is also require minimum input arrival time before clock of 4.20ns followed by RSA 6.70ns, TPRT 4.33ns, TMAT 5.55ns, ROBAST 5.55ns, SRAT 4.25ns and TSV 6.66ns. MFBOMAT requires minimum value in maximum output required time after clock of 3.30ns followed by RSA 4.31ns, TPRT 3.33ns, TMAT 4.25ns, ROBAST 4.44ns, SRAT 3.33ns and TSV 5.55ns. Thus it can be said that MFBOMAT is giving optimal result in terms of hardware implementation.

9.6.2 The Frequency Distribution Graph

The frequency distribution is the distribution of the all 256 ASCII characters present in the respective files. This is also a cryptographic parameter which measures the degree of cryptanalysis. The analysis has been given after the following figures showing the frequency distribution encrypted by all the proposed techniques.



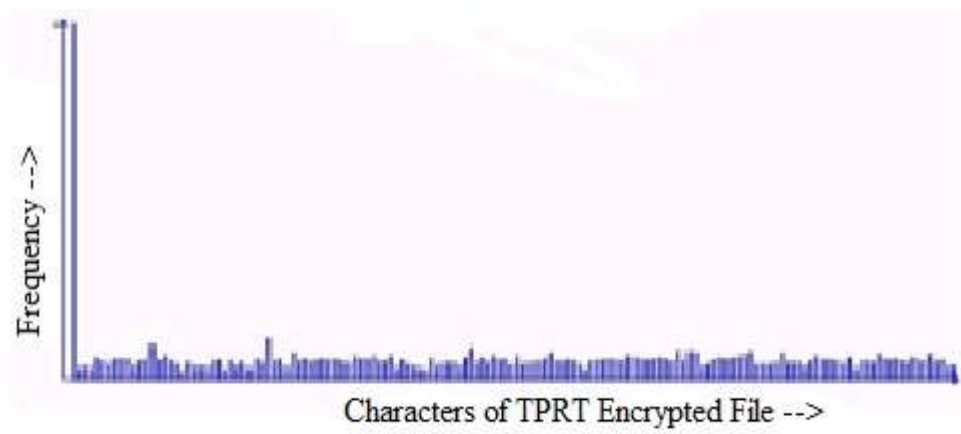
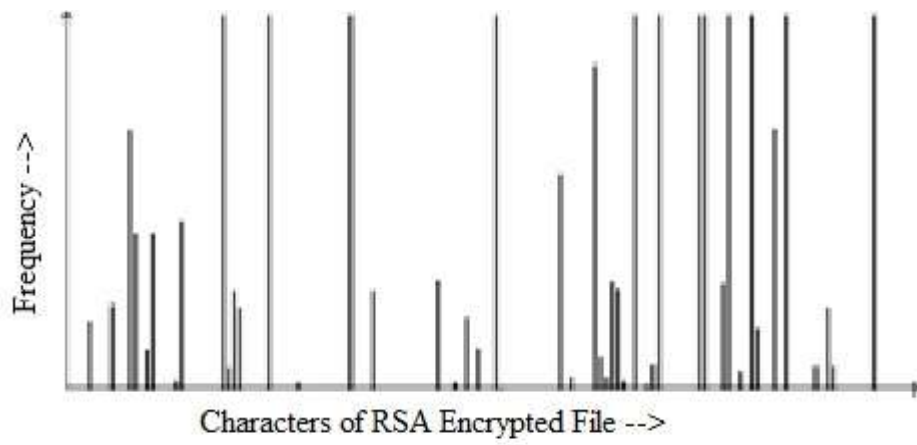


Figure 9.13: Frequency distribution graph of source, RSA encrypted and TPRT encrypted files



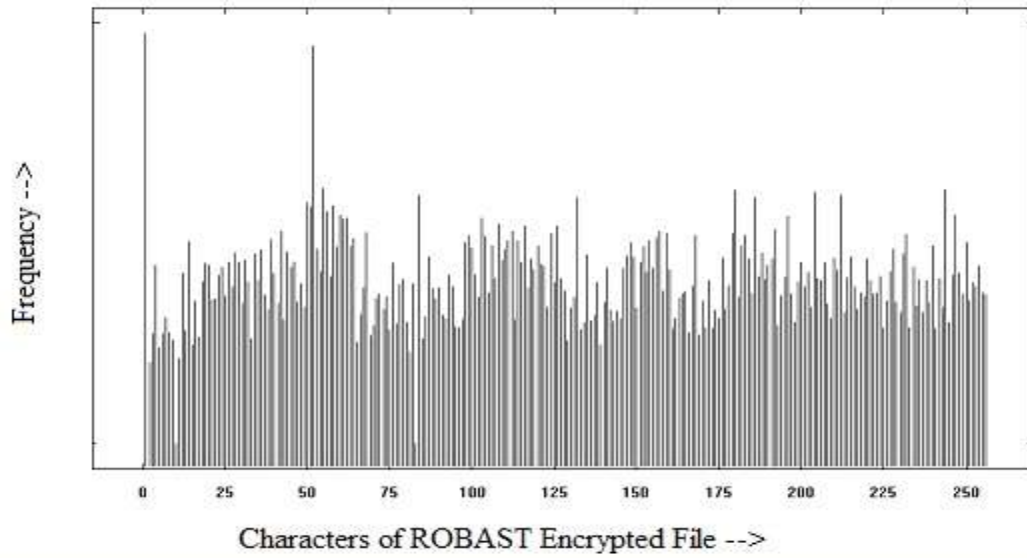


Figure 9.14: Frequency distribution graph of TMAT and ROBAST encrypted files

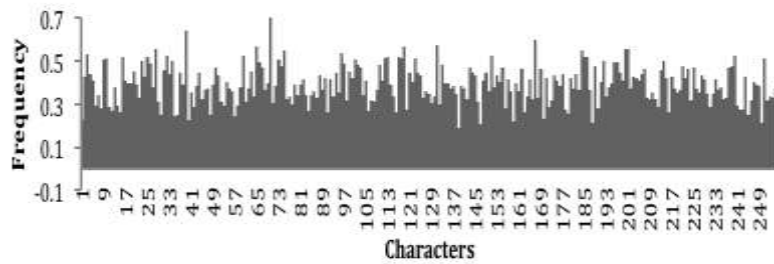


Figure 9.15: Frequency distribution graph of SRAT encrypted files

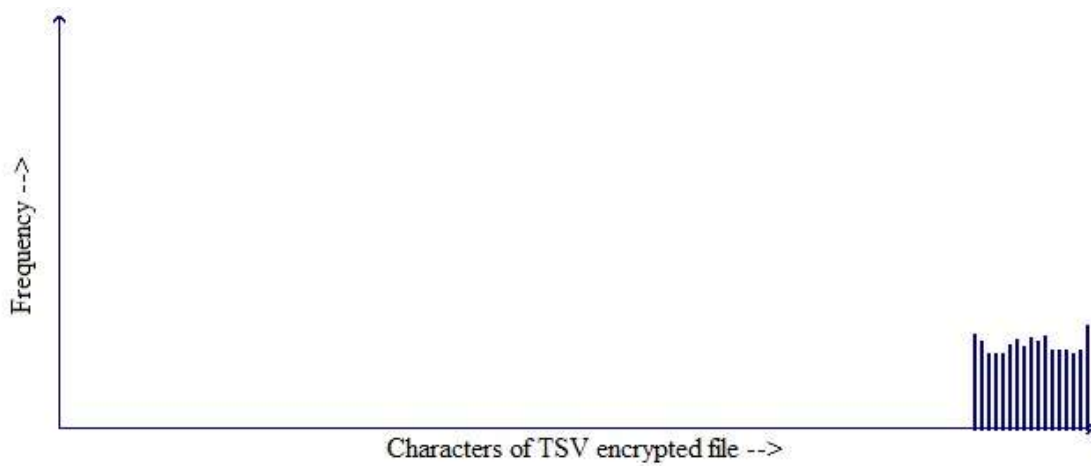


Figure 9.16: Frequency distribution graph of TSV encrypted files

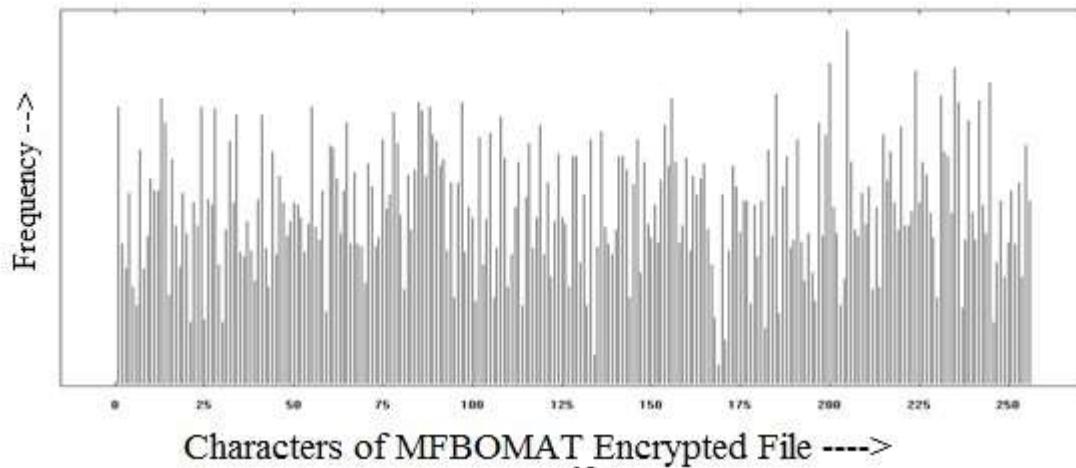


Figure 9.17: Frequency distribution graph of MFBOMAT encrypted files

The results shown are obtained after calculating the respective Frequency Distributions of the source file 'genesis.txt'. Figure 9.13 shows the frequency distribution graph of source file, RSA encrypted file and TPRT encrypted file. Figure 9.14 shows frequency distribution graph of TMAT encrypted file and ROBAST encrypted file. Figure 9.15 shows the frequency distribution graph of SRAT encrypted file, figure 9.16 shows frequency distribution graph of TSV encrypted file and figure 9.17 shows the frequency distribution graph of MFBOMAT encrypted files. It obvious that MFBOMAT is giving much better result than that of RSA.

9.6.3 The Non-Homogeneity Test

The extent of non-homogeneity between source file/plaintext and encrypted file/ciphertext is computed using Chi-Square value. In this context the observed frequency is the plaintext files and the expected frequency is the ciphertext files. Thus it gives the extent of non-homogeneity between plaintext files and ciphertext files.

Table 9.4: Comparison of Chi-Square values of ROBAST, RSA, TPRT, TMAT, SRAT, TSV and MFBOMAT

Source File	File Size (Bytes)	Chi-Square Values						
		ROBAST	RSA	TMAT	TPRT	SRAT	TSV	MFBOMAT
license.txt	17,632	6472	5668	201530	191382	201960	210050	221045
cs405(ej).doc	25,422	4407	2654	286025	253470	305590	306000	317698
acread9.txt	35,121	560357	447984	440184	410735	451125	475590	476075
deutsch.txt	47,829	3307374	685963	555220	505121	558330	3567900	3882550
genesis.txt	49,600	2679799	3318506	659045	638592	683128	3580050	4001235
pod.exe	69,981	8495675	694410	905416	896405	937565	8590100	8955655
mspaint.exe	136,463	3131296	2667664	1297256	1203665	1308890	3595000	4125600
cmd.exe	152,028	9559993	2216429	1759014	1692655	2009956	9569921	9570030
d3dim.dll	193,189	3102369	906300	4630652	4250652	9900630	9910550	9928915
clbcataq.dll	403,901	2590855	3896171	4167801	3922143	4525650	5125590	6290590

Table 9.4 shows the Chi-Square values of all the techniques, the cumulative Chi-Square value of MFBOMAT is 47769393, TSV is 44930751, SRAT is 20882824, TPRT is 13964820, TMAT is 14902143, ROBAST is 33438597 and RSA is 14841749. So, MFBOMAT generates the optimal result therefore MFBOMAT is the most heterogeneous technique.

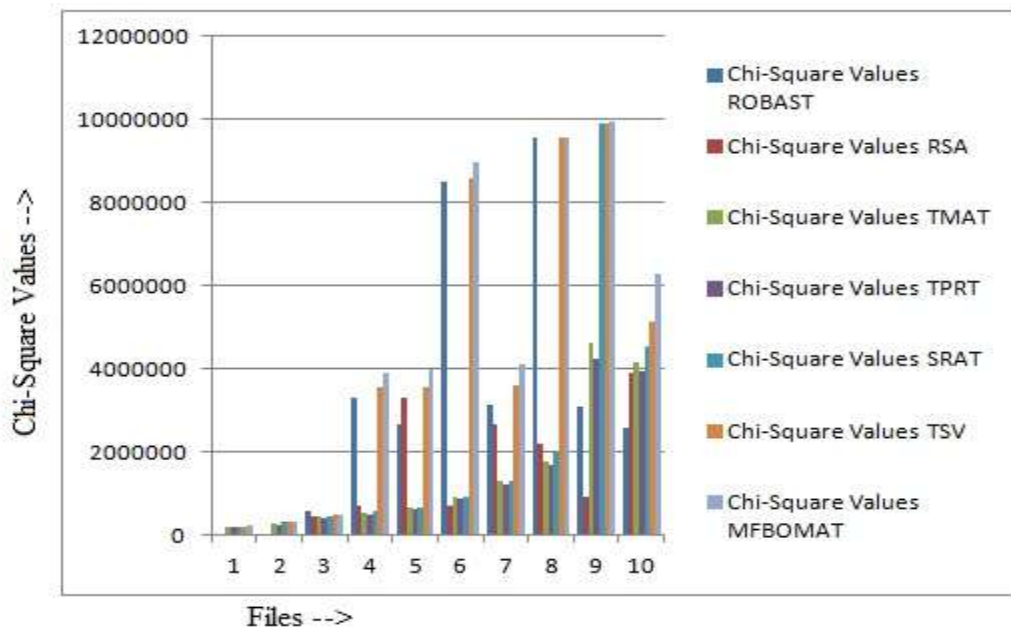


Figure 9.18: Pictorial representation of Chi-Square values against file size

Table 9.5: Comparison of degree of freedom of ROBAST, RSA, TPRT, TMAT, SRAT, TSV and MFBOMAT

Source File	File Size (Bytes)	Degree of Freedom						
		ROBAST	RSA	TMAT	TPRT	SRAT	TSV	MFBOMAT
license.txt	17,632	253	253	255	255	253	255	255
cs405(ei).doc	25,422	253	253	255	255	254	255	255
acread9.txt	35,121	253	253	255	255	255	254	254
deutsch.txt	47,829	253	253	255	255	240	253	255
genesis.txt	49,600	253	253	255	255	255	255	255
pod.exe	69,981	253	253	255	255	255	255	253
mspaint.exe	136,463	254	254	255	255	255	254	255
cmd.exe	152,028	253	253	255	255	255	255	255
d3dim.dll	193,189	253	253	255	255	255	253	255
clbcataq.dll	403,901	253	253	255	255	255	255	255

Figure 9.18 gives the Chi-Square graph where it can be seen that MFBOMAT is giving the optimal result. Table 9.5 giving the degree of freedom values where MFBOMAT is giving almost 255 values.

9.6.4 The Time Complexity Analysis

In this section time complexity analysis has been taken and for this encryption time and decryption time has been taken for analysis.

Table 9.6 gives the encryption times of ten different files. The cumulative time of MFBOMAT is 0.15 seconds, TSV is 0.19 seconds, SRAT is 0.25 seconds, TPRT is 2.94 seconds, TMAT is 3.29 seconds, ROBAST is 0.43 seconds and RSA is 1.54 seconds. Therefore MFBOMAT is giving optimal result in terms of encryption time complexity analysis.

Table 9.6: Comparison of encryption time of ROBAST, RSA, TMAT, TPRT, SRAT, TSV and MFBOMAT

Source File	File Size (Bytes)	Encryption Time						
		ROBAST	RSA	TMAT	TPRT	SRAT	TSV	MFBOMAT
license.txt	17,632	0.00	0.01	0.03	0.02	0.00	0.00	0.00
cs405(ei).doc	25,422	0.01	0.06	0.00	0.00	0.01	0.00	0.00
acread9.txt	35,121	0.02	0.07	0.13	0.10	0.01	0.01	0.01
deutsch.txt	47,829	0.03	0.11	0.25	0.20	0.01	0.01	0.01
genesis.txt	49,600	0.04	0.12	0.28	0.25	0.02	0.01	0.01
pod.exe	69,981	0.04	0.12	0.39	0.35	0.02	0.02	0.01
mspaint.exe	136,463	0.06	0.20	0.44	0.40	0.03	0.02	0.02
cmd.exe	152,028	0.07	0.25	0.55	0.50	0.05	0.03	0.02
d3dim.dll	193,189	0.08	0.28	0.55	0.52	0.05	0.04	0.03
clbcataq.dll	403,901	0.08	0.32	0.67	0.60	0.05	0.05	0.04

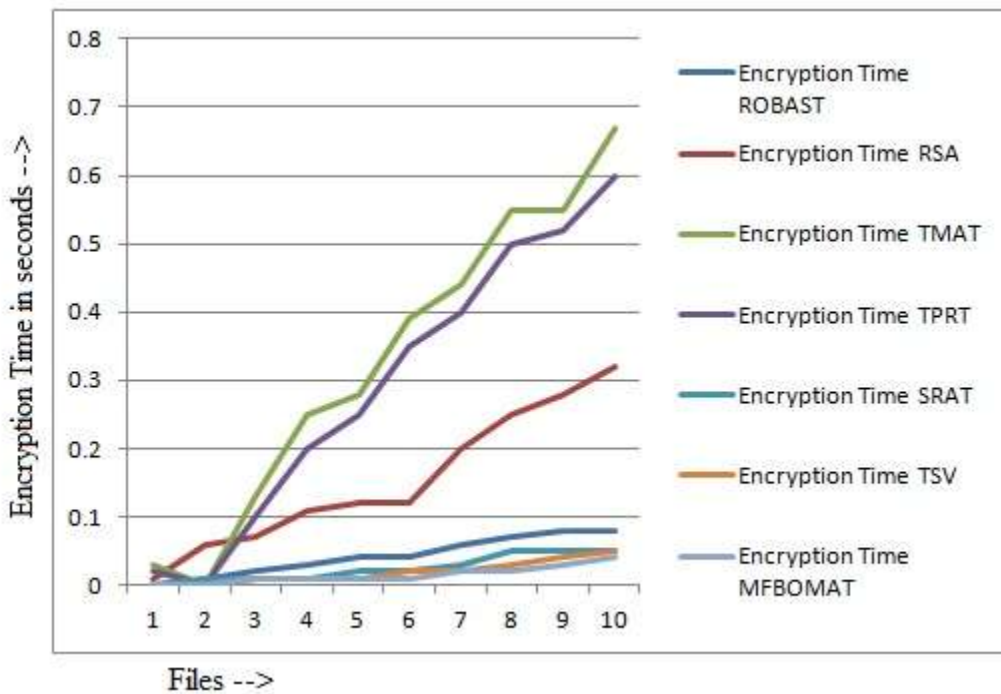


Figure 9.19: Pictorial representation of encryption time against file size

Table 9.7: Comparison of decryption time of ROBAST, RSA, TMAT, TPRT, SRAT and TSV

Source File	File Size (Bytes)	Decryption Time						
		ROBAST	RSA	TMAT	TPRT	SRAT	TSV	MFBOMAT
license.txt	17,632	0.01	0.15	0.11	0.10	0.00	0.00	0.00
cs405(ei).doc	25,422	0.02	0.71	0.00	0.00	0.01	0.00	0.00
acread9.txt	35,121	0.03	1.15	0.13	0.10	0.01	0.01	0.00
deutsch.txt	47,829	0.03	1.36	0.15	0.11	0.01	0.01	0.01
genesis.txt	49,600	0.04	1.61	0.25	0.20	0.02	0.02	0.01
pod.exe	69,981	0.04	1.86	0.39	0.35	0.02	0.02	0.01
mspaint.exe	136,463	0.05	2.71	0.48	0.40	0.02	0.03	0.02
cmd.exe	152,028	0.06	3.34	0.52	0.42	0.05	0.03	0.03
d3dim.dll	193,189	0.07	3.73	0.60	0.50	0.05	0.04	0.03
clbcattq.dll	403,901	0.08	4.25	0.65	0.55	0.05	0.05	0.04

Table 9.7 gives the decryption times of ten different files. The cumulative time of MFBOMAT is 0.15 seconds, TSV is 0.21 seconds, SRAT is 0.24 seconds, TPRT is 2.73 seconds, TMAT is 3.28 seconds, ROBAST is 0.43 seconds and RSA is 20.87 seconds. Therefore MFBOMAT is giving optimal result in terms of decryption time complexity analysis.

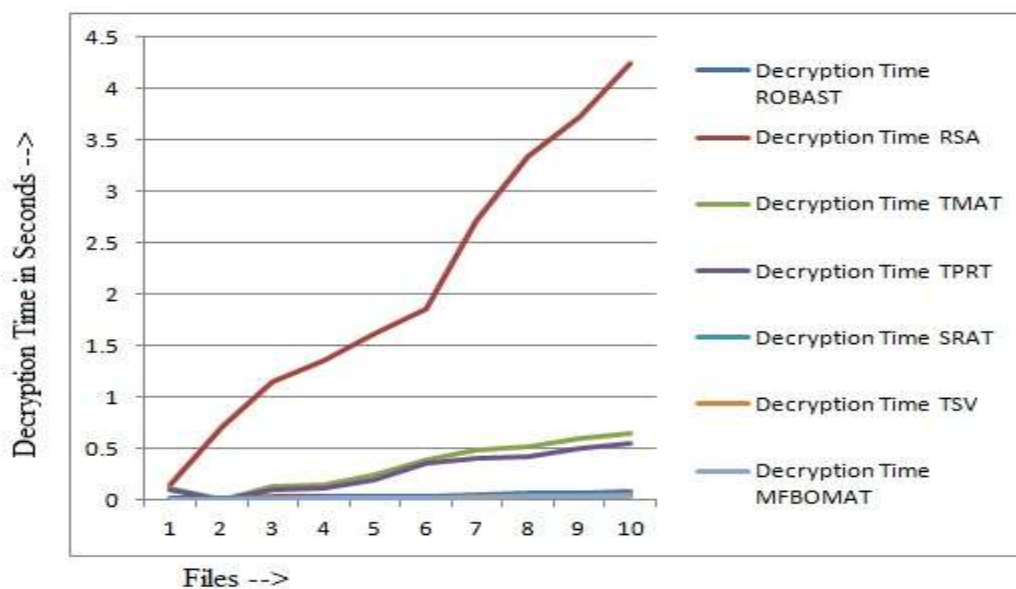


Figure 9.20: Pictorial representation of decryption time against file size

Figure 9.19 and figure 9.20 gives the encryption time complexity graph and decryption time complexity graph respectively. Thus it is clear from the graph that MFBOMAT gives the optimal result in time complexity analysis.

9.6.5 The Avalanche Ratio Test

Avalanche Effect refers to a desirable property of any cryptographic algorithm where, if an input is changed slightly (for example, flipping a single bit) the output changes significantly (e.g., more than half the output bits flip).

Table 9.8: Comparison of avalanche ratio of ROBAST, RSA, TPRT, TMAT, SRAT and TSV encrypted files

Source File	File Size (Bytes)	Avalanche Ratio of ROBAST encrypted files (in %)	Avalanche Ratio of RSA encrypted files (in %)	Avalanche Ratio of TPRT encrypted file (in %)	Avalanche ratio of TMAT encrypted file (in %)	Avalanche ratio of SRAT encrypted file (in %)	Avalanche ratio of TSV encrypted file (in %)	Avalanche ratio of MFBOMAT encrypted file (in %)
license.txt	17,632	71.90	58.0	77.7	80.8	91.5	99.9	99.5
cs405(ei).doc	25,422	99.69	60.0	80.0	85.5	90.5	99.8	99.5
acread9.txt	35,121	99.93	75.0	88.8	90.0	98.0	99.9	99.5
deutsch.txt	47,829	99.96	78.9	89.0	91.5	99.5	99.7	98.0
genesis.txt	49,600	97.72	80.9	87.0	94.7	99.9	99.6	98.5
pod.exe	69,981	77.00	58.0	77.0	80.0	99.9	99.8	99.0
mspaint.exe	136,463	98.22	58.9	76.0	80.0	98.0	99.9	99.0
cmd.exe	152,028	99.97	67.0	77.0	80.0	97.0	99.8	98.5
d3dim.dll	193,189	99.98	67.9	82.9	85.0	97.5	99.7	99.0
clbcatq.dll	403,901	75.55	68.0	88.5	90.5	99.0	99.9	99.0

Table 9.8 compares the avalanche effect ratio for TSV, RSA and previous proposed techniques/algorithm and which are obtained after calculating the respective Avalanche Effect by making a change of a few (approx 3) characters in each file. It is observed that the proposed technique is showing an average avalanche ratio percentage of 99.7% which is way higher than that obtained using RSA. High avalanche ratio ensures higher security from brute force attack. It is also observed that this avalanche ratio test of TSV is better than Shuffle-RAT, TPRT, TMAT, ROBAST and MFBOMAT. Therefore MFBOMAT is giving optimal result in avalanche ratio test but not better than TSV.

9.7 Discussions

The technique proposed takes little time to encode and decode though the block length is high. The encoded string will not generate any overhead bits. The block length may further increased beyond 256 bits, which may enhance the security. Selecting the block pairs in random order, rather than taking those in consecutive order may enhance security. The proposed scheme may be applicable to embedded systems.

Chapter 10
Proposed Models

10.1 Proposed Models

In this chapter two models have been proposed out of research carried out during the generic of study. Section 10.2 describes a proposed model derived from microprocessor based solutions and section 10.3 gives another proposed model derived from FPGA-Based solutions. Discussions on the proposed model are given in section 10.4.

10.2 The Proposed Model for Microprocessor-Based Solutions

Complete functionalities could be obtained through models and this chapter proposed models for the same. Figure 10.1 shows the proposed model for microprocessor-based solutions. In this model a 64-bit plaintext is encrypted to produce a 64-bit ciphertext, with a key of 128-bit key size. In this model there are three parts, first one is the encryption process, second one is the decryption process and the third one is the key generation or sub-key generation process.

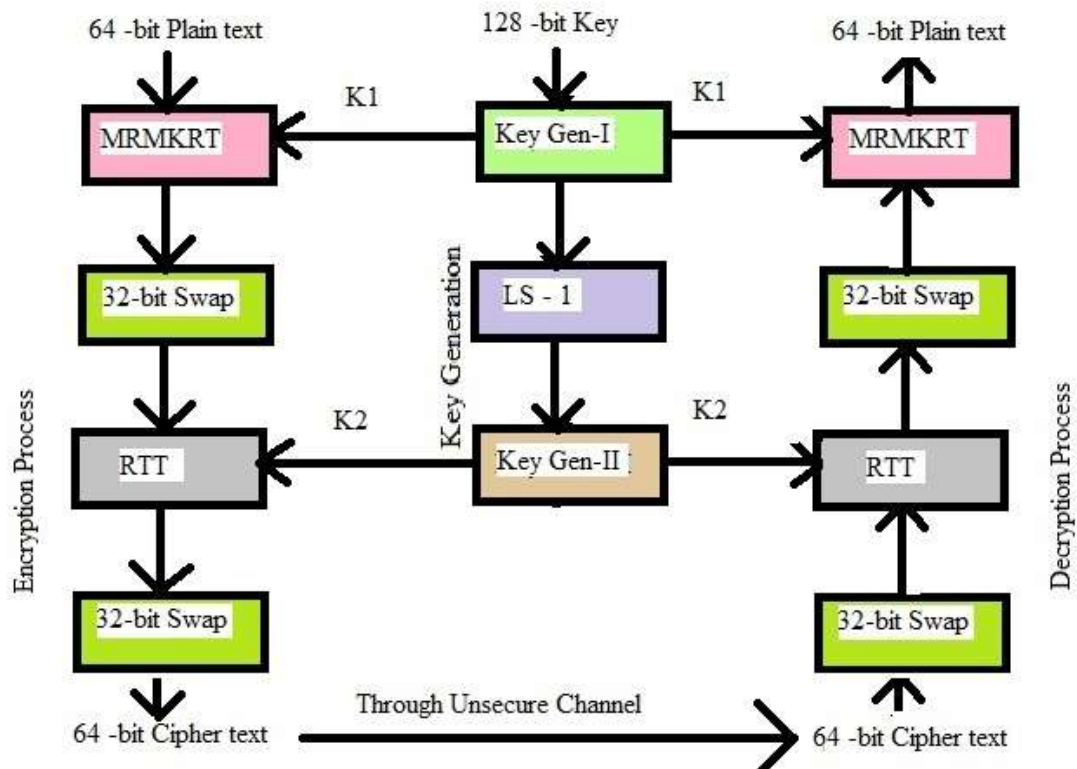


Figure 10.1: Proposed model for microprocessor-based solutions

64-bit plaintext is fed into Modified Recursive Modulo-2ⁿ and Key Rotation Technique (MRMKRT) block which gets the round-key/sub-key, K1, and this plaintext is encrypted by MRMKRT encryption. The output from this is divided into two blocks of 32-bit each, the left 32-bit block and right 32-bit block, then a 32-bit swap is done, here the left 32-bit block becomes the right 32-bit block and right 32-bit block becomes the left 32-bit block. After swapping two blocks are merged to form 64-bit block. This 64-bit block is then fed to Recursive Transposition Technique (RTT), which gets round-key/sub-key, K2; this 64-bit input is encrypted by RTT encryption. The output from this phase is again performed 32-bit swap. The 64-bit output from this final phase is the ciphertext. Let's now discuss the decryption process.

The 64-bit ciphertext produced above by the encryption process travels through unsecure channel and reaches the destination. The decryption process is just the reverse of the encryption process and the round-keys/sub-keys is applied in reverse order. At first a 32-bit swap operation is performed, this means the 32-bit left input block becomes right output block and the 32-bit right input block becomes left output block. This 64-bit block is fed to RTT decryption, round-key/sub-key, K2. Again the output from this phase is performed a similar 32-bit swap. Finally, this 64-bit block is fed to MRMKRT decryption with round-key/sub-key, K1, this produce back the original 64-bit plain text. Now discuss the key generation process.

In this proposed model 128-bit key size has been considered. First, 128-bit key is fed to first stage of key generation (Key Gen – I), this round-key/sub-key generation process which is already described in section 2.5 of chapter 2. First round-key/sub-key, K1 generated is passed to MRMKRT in both encryption process and decryption process. This 128-bit key is performed a circular left shift of 1 – bit (LS – 1). The output from this phase is fed to second phase of key generation (Key Gen – II), this round-key/sub-key generation process which is already described in section 3.3 of chapter 3. This phase generates a round-key/sub-key, K2, which is fed to RTT encryption process and decryption process. This proposed model has been implemented both in 8085 microprocessor and C-programming language. Detailed analysis of results of the proposed model is given in section 10.2.1.

10.2.1 Results and Comparisons

The main emphasis is given on comparisons of the results with RSA. Section 10.2.1.1 gives implementation based results, section 10.2.1.2 gives frequency distribution graph,

section 10.2.1.3 gives non-homogeneity test, section 10.2.1.4 deals with time complexity analysis and avalanche ratio is given in section 10.2.1.5.

10.2.1.1 Implementation based result

In this section results obtained from implementation of the proposed model is compared with MRMKRT and RTT as there is no low level implementation is available of RSA.

Table 10.1: Implementation based results of MRMKRT, RTT and proposed model

Characteristics ↓	Proposed Techniques →	MRMKRT	RTT	Proposed Model
Block Cipher		√	√	√
Fixed Length Block Cipher		√	-	-
Variable Length Block Cipher		-	√	√
Implementation in Bit-Level		√	√	√
Implementation other than Bit-Stream		-	-	-
Private/Symmetric Key System		√	√	√
Substitution Technique		√	√	√
Transposition Technique		-	√	√
Boolean as Basic Operation		√	√	-
Non-Boolean as Basic Operation		√	-	-
No Alteration in Size		√	√	√
Formation of Cycle		√	√	-
Non-formation of Cycle		-	-	√
Number of sub-programs used		4	7	12
Number of IO/M operations per block of encryption/decryption		9	5	15
Number of Boolean operations used per block of encryption/decryption		1	1	1
Number of Non Boolean operations used per block of encryption/decryption		5	0	5
Calculated T-states per block of encryption/decryption		760	544	1350

Table 10.1 gives the summary of implementation based results where this proposed model is compared with MRMKRT and RTT.

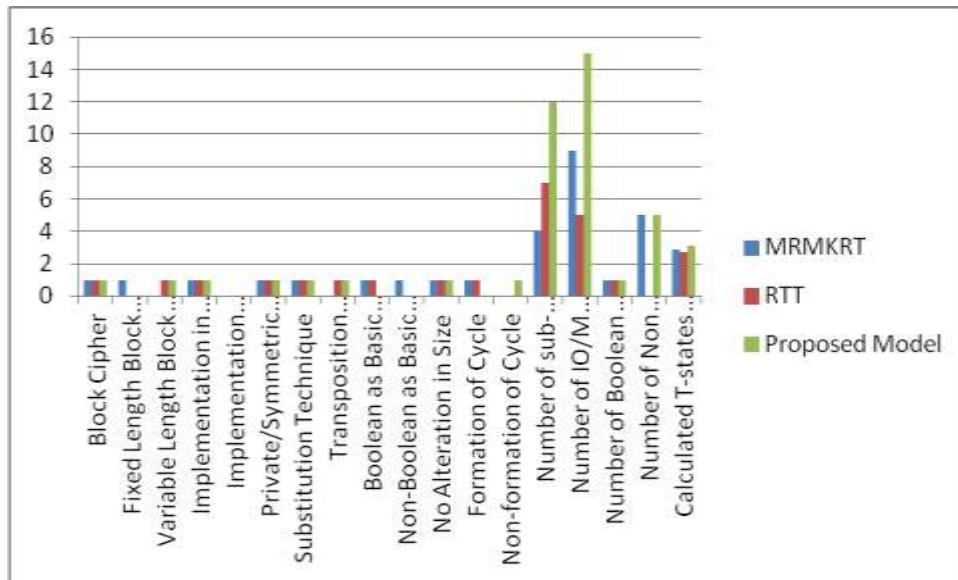


Figure 10.2: Graphical representation of implementation based results of the model, MRMKRT and RTT

Figure 10.2 shows the graphical representation of implementation based result. This proposed model is variable size block cipher, the implementation is bit level with private key stream, it also involve both permutation and substitution technique, this proposed model also have Boolean and non Boolean operations and there is no alteration of cycle and also cycle is not formed.

Number of subprogram is used is 12, IO/M operation is 15, Boolean operation is 1, non Boolean operation is 5 and calculated T-states is 1350. Therefore with these results it can be said that the proposed model is successfully implemented in low level that in 8085.

10.2.1.2 Frequency Distribution Graph

All 255 ASCII characters are taken for this test for both plaintext/source file and ciphertext/encrypted file. The frequency graph of source file, RSA encrypted file and proposed model encrypted file is taken for consideration. There are ten files taken for various result and analysis but only one file is taken for frequency distribution analysis and other nine files giving the similar result.

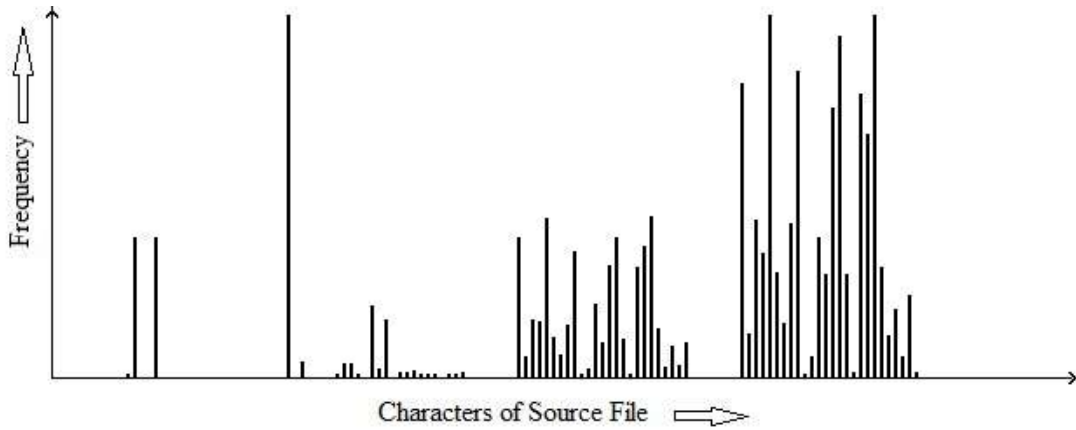


Figure 10.3: Frequency distribution of source file

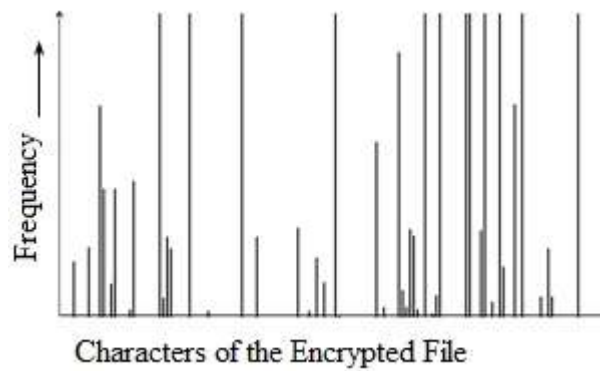


Figure 10.4: The frequency distribution graph of RSA encrypted file



Figure 10.5: Graphical representation of frequency distribution of proposed model (encrypted file)

Figure 10.3 gives the frequency distribution graph of source file, figure 10.4 shows the same for RSA encrypted file and figure 10.5 shows the same for this proposed model encrypted file. Thus frequency distribution graph of this proposed model is well comparable with RSA.

10.2.1.3 Non-Homogeneity Test

Chi-square test is performed to find the non-homogeneity of the proposed model with RSA. Ten files of different file typed and different file sizes are taken for this test.

Table 10.2: Comparisons of Chi-Square values of RSA and proposed model

Source File	File Size (Bytes)	Chi-Square Value		Degree of Freedom	
		Proposed model	RSA	Proposed model	RSA
license.txt	17,632	225000	40159	255	64
cs405(ei).doc	25,422	299125	199354	255	66
acread9.txt	35,121	460050	179524	255	73
deutsch.txt	47,829	588660	344470	255	77
genesis.txt	49,600	690010	416029	255	75
pod.exe	69,981	901556	751753	255	76
mspaint.exe	136,463	1550000	1204193	255	88
cmd.exe	152,028	1908000	585857	255	73
d3dim.dll	193,189	496590	328677	255	10
clbcatq.dll	403,901	3907125	328511	255	11

Table 10.2 shows the Chi-Square values of ten source files of RSA and the proposed model. It is clearly observed that the extent of non-homogeneity of the proposed model is quite higher than that of RSA. So, this proposed model is giving optimal solution in terms of non-homogeneity by using Chi-Square values.

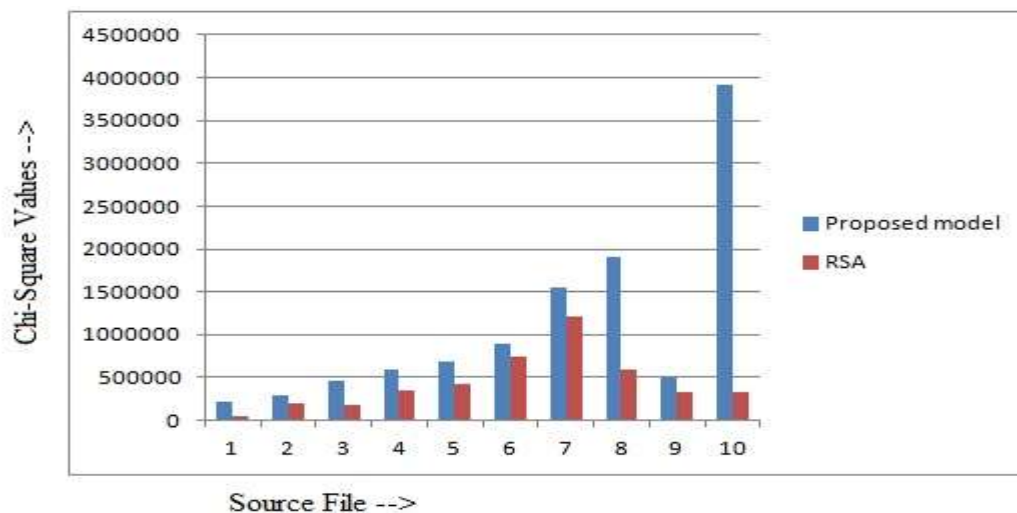


Figure 10.6: Graphical representation of Chi-Square for RSA and proposed model

Figure 10.6 shows the result graphically, where it can be see that for all the ten source files the Chi-Square value of this proposed model is quite higher than that of RSA.

10.2.1.4 Time Complexity Analysis

Time complexity analysis is taken for encryption time and decryption time. This analysis is taken for all the ten source files.

Table 10.3 gives the tabulation for encryption time and decryption time of both RSA and proposed model. Figure 10.7 gives the graphical representation of encryption time and figure 10.8 gives the graphical representation of decryption time. The cumulative encryption time of this proposed model is 4.34 seconds and the cumulative encryption time of RSA is 4.35 seconds. The cumulative decryption time of the proposed model is 5.69 seconds and cumulative decryption time of RSA is 51.97 seconds. Therefore it can be said that the proposed model is giving much better result than that of RSA.

Table 10.3: Comparison of time complexity analysis of RSA and proposed model

Source File	File Size (Bytes)	Encryption time (in Seconds)		Decryption time (in seconds)	
		Proposed Model	RSA	Proposed model	RSA
license.txt	17,632	0.02	0.01	0.15	0.28
cs405(ei).doc	25,422	0.02	0.03	0.15	0.30
acread9.txt	35,121	0.20	0.21	0.20	1.67
deutsch.txt	47,829	0.30	0.35	0.30	3.51
genesis.txt	49,600	0.40	0.40	0.55	5.06
pod.exe	69,981	0.80	0.39	0.45	4.34
mspaint.exe	136,463	0.70	0.65	0.80	8.37
cmd.exe	152,028	0.50	0.61	0.90	6.59
d3dim.dll	193,189	0.70	0.75	0.99	10.15
clbcattq.dll	403,901	0.70	0.95	1.20	11.70

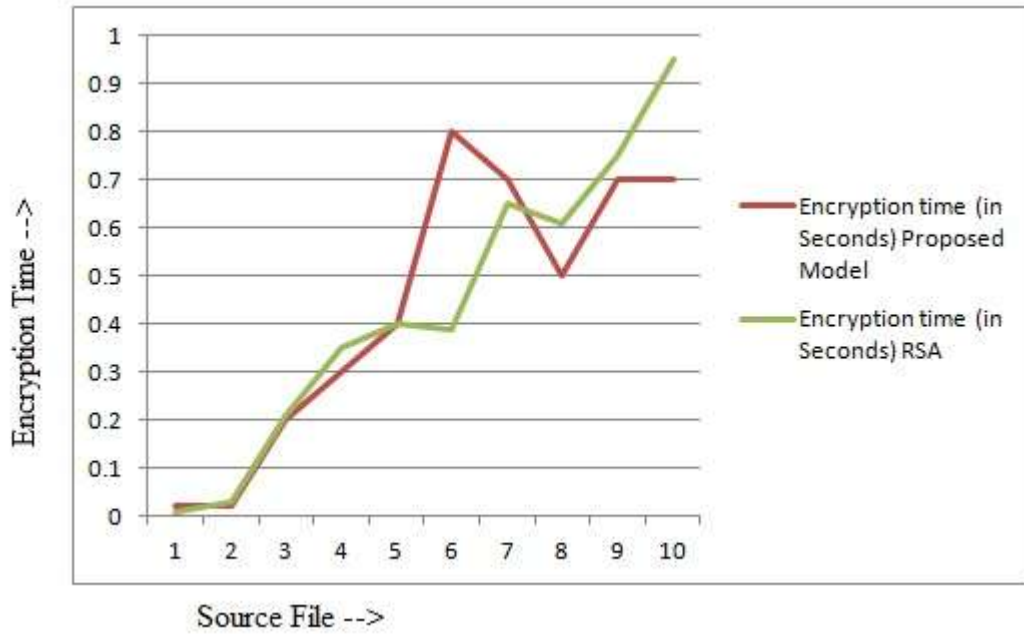


Figure 10.7: Pictorial representation of encryption time

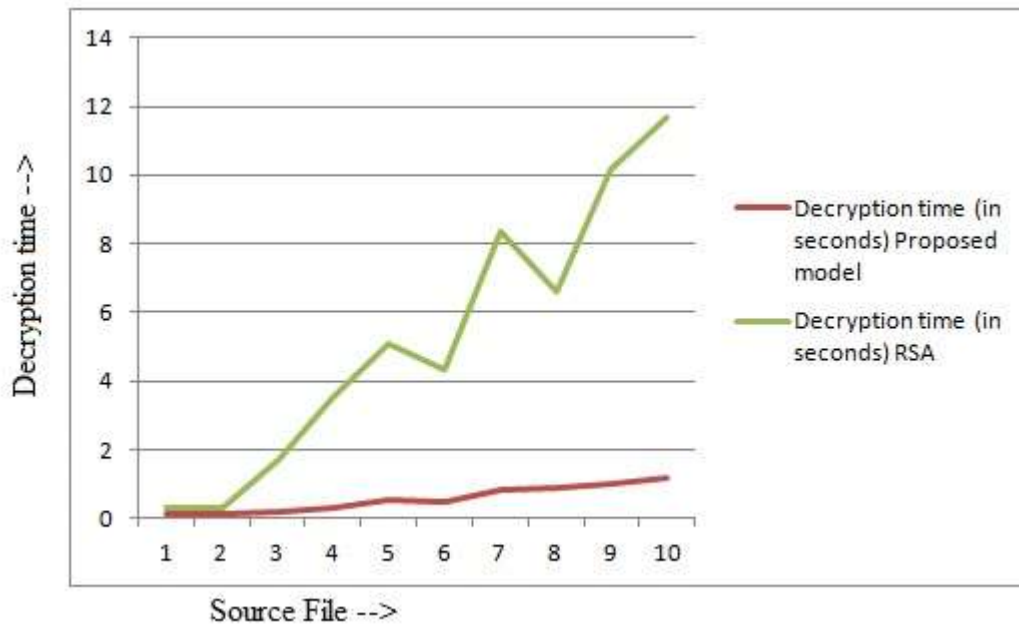


Figure 10.8: Graphical representation of decryption time

10.2.1.5 The Avalanche Ratio Test

The avalanche ratio test is the extent of which the ciphertext bits flipped when one or more bits of plaintext or key are flipped.

Table 10.4: Comparison of avalanche ratio of RSA and proposed model

Source File	File Size (Bytes)	Avalanche Ratio(in Percentage)	
		RSA	Proposed model
license.txt	17,632	58.0	88.8
cs405(ei).doc	25,422	60.0	80.0
acread9.txt	35,121	75.0	88.8
deutsch.txt	47,829	78.9	90.5
genesis.txt	49,600	80.9	95.5
pod.exe	69,981	58.0	90.0
mspaint.exe	136,463	58.9	96.5
cmd.exe	152,028	67.0	87.0
d3dim.dll	193,189	67.9	85.0
clbcatq.dll	403,901	68.0	95.5

Table 10.4 is giving the avalanche ratio test and here the proposed model is giving far better result than that of RSA.

10.3 The Proposed Model for FPGA-Based Solutions

An FPGA-based model has also been proposed here where, the complete functionalities of FPGA-based techniques is described here through another proposed model. Figure 10.9 shows the proposed model for FPGA-based solutions. In this model also a 64-bit plaintext is encrypted to produce a 64-bit ciphertext, with a key of 128-bit key size. In this model there are three parts, first one is the encryption process, second one is the decryption process and the third one is the key generation or sub-key generation or round-key generation process. Let explain it one by one.

At first 64-bit plaintext is fed into Triangular Modulo Arithmetic Technique (TMAT) block which gets the round-key/sub-key, K1, and this plaintext is encrypted by TMAT encryption. The output from this is now performed a circular left shift of 1 – bit (LS – 1). After shifting operation this 64-bit block is then fed to Recursively Oriented Block Addition and Substitution Technique (ROBAST), which get round-key/sub-key, K2, this 64-bit input is encrypted by ROBAST encryption. The output from this phase is again performed circular left shift of 2 – bits (LS – 2).

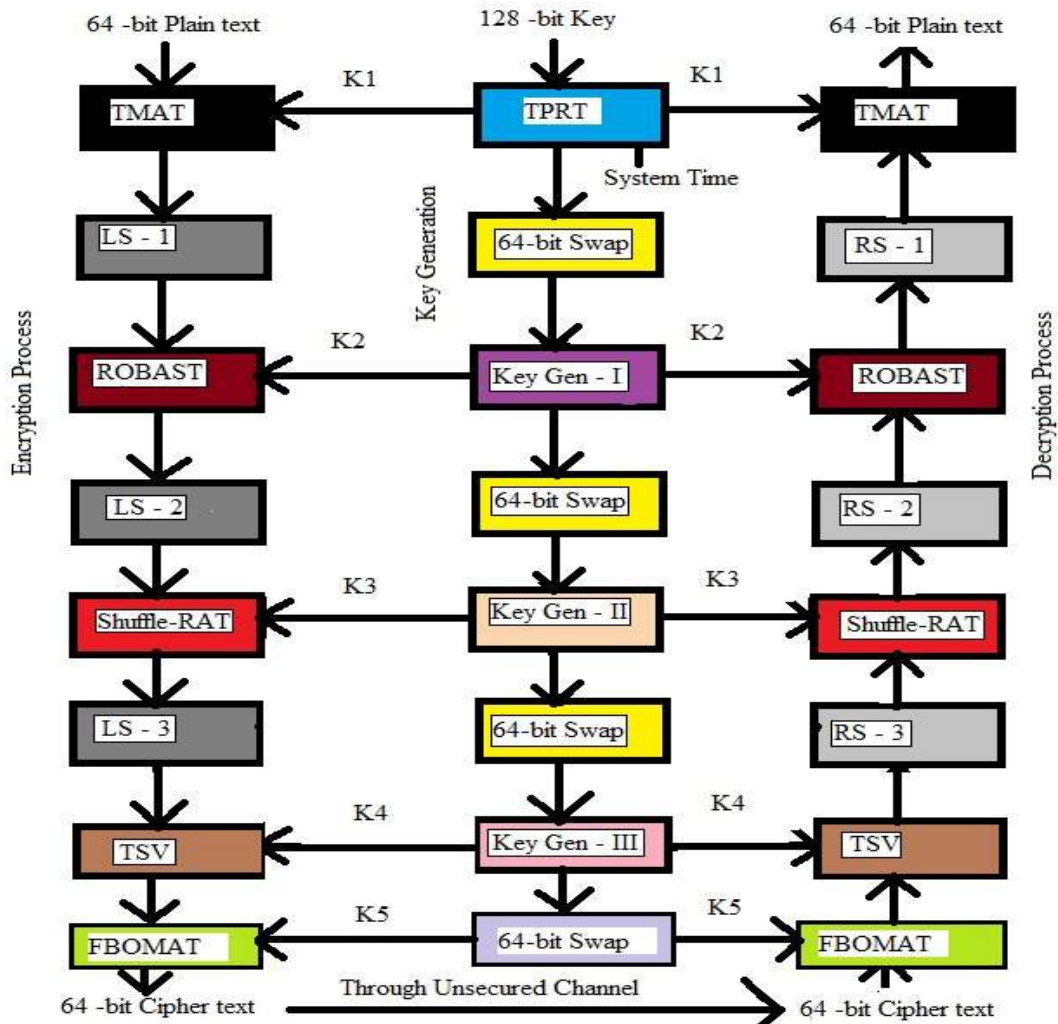


Figure 10.9: Proposed model for FPGA-based solutions

After shifting operation the 64-bit block is fed to Shuffle-RAT (SRAT) phase, which get round-key/sub-key, K3, from the key generation process. This 64-bit block is encrypted with SRAT. After this the 64-bit block is now performed a circular left shift of 3 – bits (LS – 3). The 64-bit output is now fed to Triple-SV (TSV) phase, which get round-key/sub-key, K4, from the key generation process. Now, finally the 64-bit block is Forward Backward Overlapped Modulo Arithmetic Technique (FBOMAT) encrypted to produce 64-bit ciphertext with a round key of K5. Let’s now discuss the decryption process.

The 64-bit ciphertext produced above by the encryption process travels through unsecure channel and reaches the decryption process. Since the techniques were symmetric in nature so the proposed model is symmetric too. The decryption process is just the reverse of the encryption process and the round-keys/sub-keys is applied in reverse order. At first 64-bit ciphertext is decrypted through FBOMAT decryption with round key K5. Then 64-bit

ciphertext is TSV decrypted, with the round key/sub key, K4. This 64-bit block is now performed a circular right shift of 3 – bits ($RS - 3$). The 64-bit output from this phase is now fed to SRAT, with round-key/sub-key, K3, this is now SRAT decrypted. Then this 64-bit block is performed a circular right shift of 2 – bits ($RS - 2$). Now, this 64-bit block is fed to ROBAST decryption, round-key/sub-key, K2. Again the output from this phase is performed a similar circular right shift of 1 – bits ($RS - 1$). Finally, this 64-bit block is fed to TMAT decryption with round-key/sub-key, K1, this produce back the original 64-bit plain text. Now discuss the key generation process.

In this proposed model, the key is considered to be 128-bit key size, which is now recommended. First, 128-bit key is fed to Two Pass Replacement Technique (TPRT), this whole 128-bit key is encrypted with one round of TPRT which takes system time as a key input. This phase produce the first round-key/sub-key, K1, which is passed to TMAT in both encryption process and decryption process. Now this 128-bit key is divided into two 64-bit blocks and swap operation is done, this means the right input 64-bit becomes left output 64-bit and left input 64-bit becomes right output 64-bit, finally the output blocks are merged to form 128-bit block. The output from this phase is fed to Key Gen – I, this round-key/sub-key generation process which is already described in section 4.4.1 of chapter 4. This phase generates a round-key/sub-key, K2, which is fed to ROBAST encryption process and decryption process. The 128-bit output from this phase is again performed a 64-bit swap operation as discussed earlier. After the swapping operation, this 128-bit block key is fed to Key Gen – II, already discussed in section 8.4.2 of chapter 8, to produce the third round-key/sub-key, K3, which is fed to both SRAT encryption process and SRAT decryption process. The 128-bit output from the previous phase is again performed 64-bit swap operation described earlier. The next round-key/sub-key is produced by performing the Key Gen – III operation as already described in section 9.4.1 of chapter 9, this round-key/sub-key, K4, is fed to both TSV encryption process and decryption process. Finally another 64-bit swap operation is performed and it forms the final round key, K5, which is fed to both FBOMAT encryption and decryption. Here one should note that all the round-keys/sub-keys is of 128-bit key size. This proposed model has been implemented in both FPGA-based systems by VHDL and C-programming language. Section 10.3.1 gives the results to analyse this proposed model for its acceptance.

10.3.1 Results and Simulations

The main emphasis is given on comparisons with RSA. Section 10.3.1.1 gives RTL simulation based results, section 10.3.1.2 gives frequency distribution graph, section 10.3.1.3 gives non-homogeneity test, section 10.3.1.4 deals with time complexity analysis and section 10.3.1.5 chalk out the avalanche ratio test.

10.3.1.1 RTL Simulation Based Result

In this section gives some of the results found after implementing the proposed model in VHDL. This code has been simulated and synthesized in Xilinx 8.1i. The main objective is to find an efficient FPGA-based cryptographic technique for implementation in embedded systems.

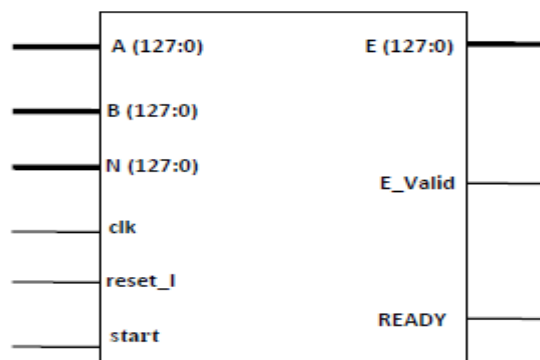


Figure 10.10: RTL diagram of RSA

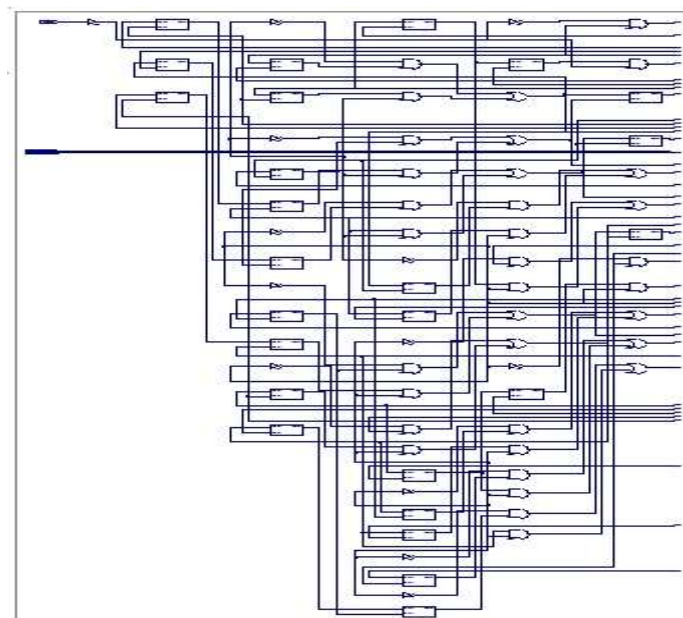


Figure 10.11: Spartan 3E RTL diagram of proposed model

The design of proposed model is done using VHDL and implemented in Xilinx Spartan-3E XC3S100E-5VQ100 (package: VQ100, speed grade: -5) FPGA using the ISE 8.1i design tool. Figure 10.10 shows the RTL of RSA, figure 10.11 shows the RTL of proposed model.

Table 10.5: HDL synthesis report (Netlist generation of RSA and proposed model)

Sr No.	Netlist Components	Number	
		RSA	Proposed model
1	ROMs/RAMs	430	110
2	Adders/Subtractions	3	70
3	Registers	420	770
4	Latches	80	80
5	Multiplexers	120	120

Table 10.5 gives the HDL synthesis report, specifically net list generation of RSA and proposed model. Proposed model uses 110 ROMs/RAMs where as RSA uses 430 ROMs/RAMs, proposed model uses 70 adders/subtractions where RSA uses only 3 adder/subtractions, proposed model uses 770 registers where as RSA uses 420 registers, both proposed model and RSA uses 80 lathes and 120 multiplexers. Therefore this proposed model is well comparable with RSA.

Table 10.6 gives the HDL synthesis report specifically timing summary of RSA and proposed model. The minimum period of RSA is 9.895ns and proposed model is 9.55ns, minimum input arrival time before clock of RSA is 6.697ns and proposed model is 6.55 ns and maximum output required time after clock of RSA is 4.31ns and the proposed model is 4.30ns. This implementation has been made on speed grade of -5 and maximum frequency of 101.06 MHZ. Therefore it can be said that proposed model is giving much better result than that of RSA and this proposed model is well comparable with RSA.

10.3.1.2 The Frequency Distribution Graph

The frequency distribution is the distribution of the all 256 ASCII characters in the respective files. This is also a cryptographic parameter which measures the degree of cryptanalysis.

Table 10.6: HDL synthesis report (Timing summary of RSA and proposed model)

Sr No.	Timing Constraint	Values	
		RSA	Proposed model
1	Speed Grade	-5	-5
2	Minimum period (ns)	9.895	9.55
3	Maximum Frequency (MHZ)	101.06	101.06
4	Minimum input arrival time before clock (ns)	6.697	6.55
5	Maximum output required time after clock (ns)	4.31	4.30

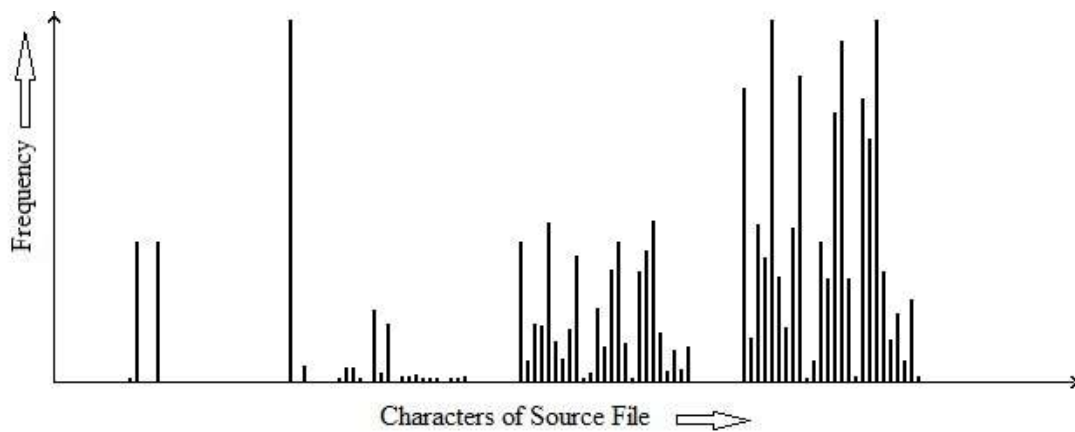


Figure 10.12: Frequency distribution of source file



Figure 10.13: The frequency distribution graph of RSA encrypted file

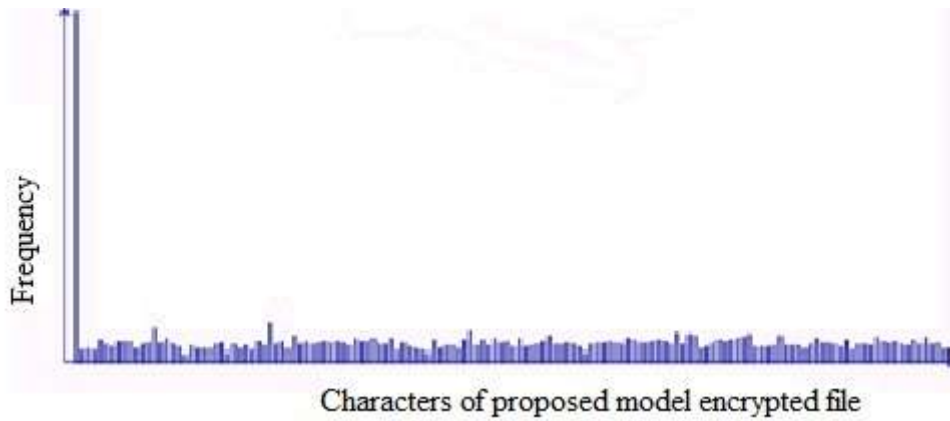


Figure 10.14: The frequency distribution graph of proposed model

Figure 10.12 gives the frequency distribution graph of source file, figure 10.13 shows the same for RSA encrypted file and figure 10.14 shows the same for this proposed model encrypted file. Thus frequency distribution graph of this proposed model is well comparable with RSA.

10.3.1.3 The Non-Homogeneity Test

Chi-square test is performed to find the non-homogeneity of the proposed model with RSA. Ten files of different file typed and different file sizes are taken for this test.

Table 10.7: Chi-Square values of RSA and proposed model

Source File	File Size (Bytes)	Chi-Square Values		Degree of freedom	
		RSA	Proposed model	RSA	Proposed model
license.txt	17,632	5668	6005	64	253
cs405(ei).doc	25,422	2654	338690	66	254
acread9.txt	35,121	447984	475859	73	255
deutsch.txt	47,829	685963	3885550	77	255
genesis.txt	49,600	3318506	4112060	75	254
pod.exe	69,981	694410	8992436	76	255
mspaint.exe	136,463	2667664	4560124	88	255
cmd.exe	152,028	2216429	9956700	73	240
d3dim.dll	193,189	906300	9925690	10	255
clbcattq.dll	403,901	3896171	6556900	11	254

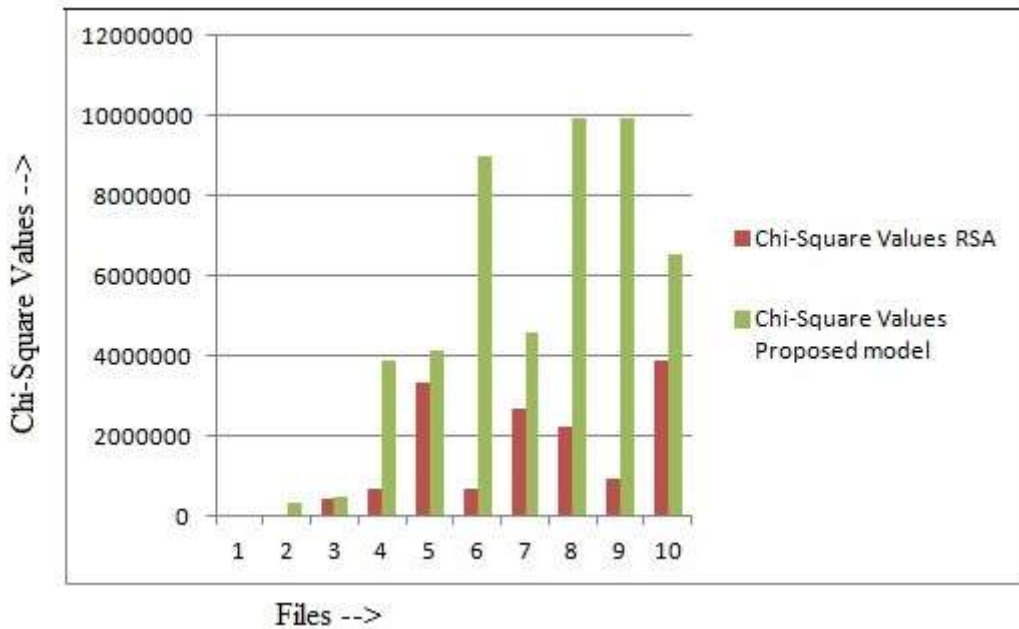


Figure 10.15: Graphical representation of Chi-Square value of RSA and proposed model

Table 10.7 shows the Chi-Square values of ten source files of RSA and the proposed model. It is clearly observed that the extent of non-homogeneity of the proposed model is quite higher than that of RSA. So, this proposed model is giving optimal solution in terms of non-homogeneity by using Chi-Square values.

Figure 10.15 shows the result graphically, from it is seen that that for all the ten source files the Chi-Square value of this proposed model is quite higher than that of RSA. The degree of freedom of this proposed model is also quite higher than that of RSA.

10.3.1.4 The Time Complexity Analysis

Another way to analyze any algorithm is to take the time complexity analysis. Here encryption time and decryption time have been taken into account.

Table 10.8: Encryption and decryption time of RSA and proposed model

Source File	File Size (Bytes)	Encryption Time		Decryption Time	
		RSA	Proposed model	RSA	Proposed model
license.txt	17,632	0.01	0.06	0.28	0.15
cs405(ei).doc	25,422	0.06	0.06	0.30	0.25
acread9.txt	35,121	0.07	0.07	1.67	0.80
deutsch.txt	47,829	0.11	0.10	3.51	0.90
genesis.txt	49,600	0.12	0.10	5.06	2.30
pod.exe	69,981	0.12	0.10	4.34	3.50
mspaint.exe	136,463	0.20	0.20	8.37	4.50
cmd.exe	152,028	0.25	0.20	6.59	6.10
d3dim.dll	193,189	0.28	0.25	10.15	8.50
clbcattq.dll	403,901	0.32	0.35	11.70	10.50

Table 10.8 gives the tabulation for encryption time and decryption time of both RSA and proposed model. Figure 10.16 gives the graphical representation of encryption time and figure 10.17 gives the graphical representation of decryption time. The cumulative encryption time of this proposed model is 1.49 seconds and the cumulative encryption time of RSA is 1.54 seconds. The cumulative decryption time of the proposed model is 37.5 seconds and cumulative decryption time of RSA is 51.97 seconds. Therefore it can be said that the proposed model is giving much better result than that of RSA.

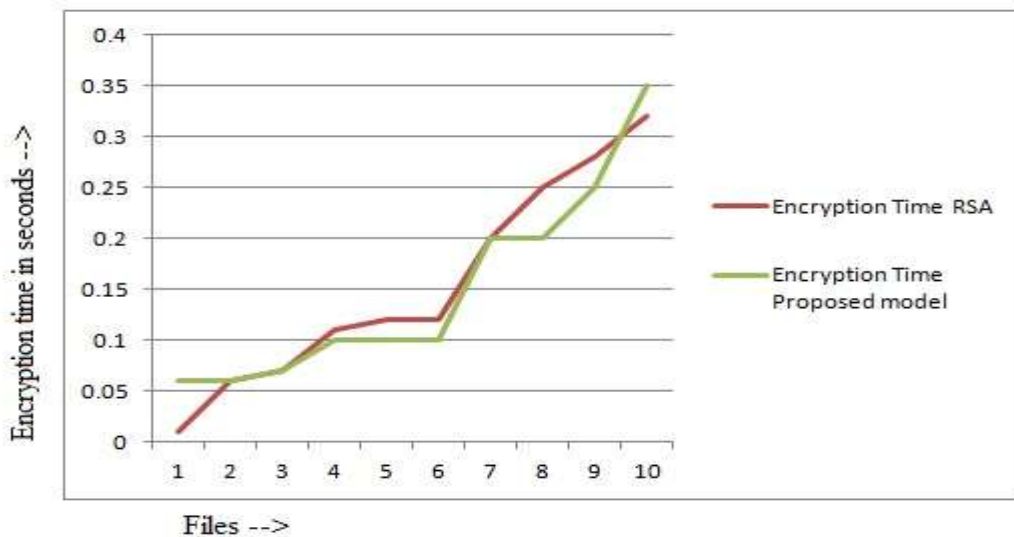


Figure 10.16: Pictorial representation of encryption time of RSA and proposed model

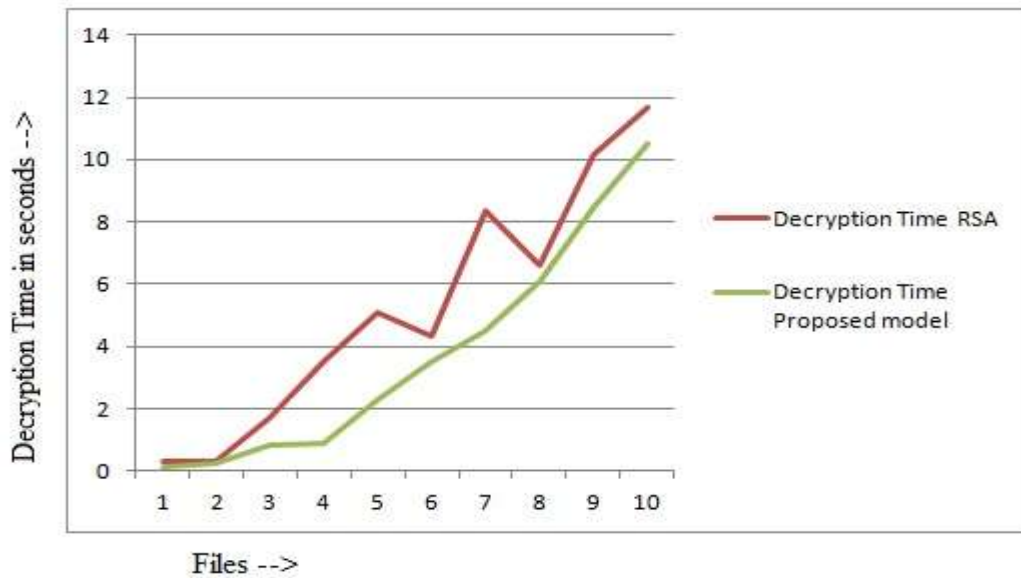


Figure 10.17: Pictorial representation of decryption time of RSA and proposed model

10.3.1.5 The Avalanche Ratio Test

The avalanche ratio test is the extent of which the ciphertext bits flipped when one or more bits of plaintext or key are flipped.

Table 10.9: The avalanche ratio of RSA and proposed model

Source File	File Size (Bytes)	Avalanche Ratio(in Percentage)	
		RSA	Proposed model
license.txt	17,632	58.0	88.8
cs405(ei).doc	25,422	60.0	80.0
acread9.txt	35,121	75.0	88.8
deutsch.txt	47,829	78.9	90.5
genesis.txt	49,600	80.9	95.5
pod.exe	69,981	58.0	90.0
mspaint.exe	136,463	58.9	96.5
cmd.exe	152,028	67.0	87.0
d3dim.dll	193,189	67.9	85.0
clbcatq.dll	403,901	68.0	95.5

Table 10.9 is giving the avalanche ratio test and here the proposed model is giving far better result than that of RSA.

10.4 Discussions

In this chapter two proposed models have been discussed. In all the models 64-bit plaintext is encrypted to get 64-bit ciphertext. The key size of all the models are 128-bit. The 128-bit key ensure better cryptographic strength and 64-bit block length is also recommended for symmetric block cipher.

In microprocessor-based model two proposed techniques, MRMKRT and RTT, are incorporated. In this model three round keys are generated from 128-bit input key. MRMKRT is found optimal for frequency distribution analysis and degree of freedom analysis, RTT is found optimal for Chi-square value (non-homogeneity) analysis and avalanche ratio. Thus, by implementing this proposed model, hope to get better cryptographic strength and algorithmic properties.

In FPGA-based model, TMAT, ROBAST, Shuffle-RAT, TSV have been incorporated for encryption/decryption process and FBOMAT is used in round key generation process. TMAT is found suitable for frequency distribution analysis, degree of freedom and hardware implementation based results, ROBAST is optimal for Chi-Square and degree of freedom analysis, Shuffle-RAT gives high confusion and diffusion and TSV is giving high avalanche effect. Thus, by implementing this proposed models. Better cryptographic strength and algorithmic properties may be achieved.

Thus, by implementing these proposed models better cryptographic strength and algorithmic properties may be achieved.

Chapter 11
Conclusions

11.1 Conclusive Discussions

In this thesis eight novel techniques are proposed. Modified Recursive Modulo-2ⁿ and Key Rotation Technique (MRMKRT) and Recursive Transposition Technique (RTT) are microprocessor-based proposal and implemented techniques. Two Pass Replacement Technique (TPRT), Triangular Modulo Arithmetic Technique (TMAT), Recursively Oriented Block Addition and Substitution Technique (ROBAST), Shuffle-RAT (SRAT), Triple-SV (TSV / 3SV) and Forward Backward Overlapped Modulo Arithmetic Technique (FBOMAT) are FPGA-based proposal and implemented techniques. The conclusions on microprocessor-based techniques, FPGA-based techniques and all the eight proposed techniques are discussed in subsequent paragraphs. Section 11.2 illustrates the future works with concluding remarks.

Table 11.1 illustrates overall conclusion scenario for microprocessor-based solutions. In this table two proposed algorithms/techniques are compared along with the existing, renowned and industrially accepted RSA. The symbol “√” shows the optimal solution. Here also six properties are considered for the evaluation. The properties for evaluation are frequency distribution graph, Chi-Square values for non-homogeneity, degree of freedom, avalanche ratio, encryption time and decryption time.

If frequency distribution graph is considered and it is seen that the entire proposed algorithm generate optimal solutions, which means the frequencies are well distributed in ciphertext as compared to plaintext; here exception is the RSA, whose frequency is not well distributed. Now considering Chi-Square values, the proposed technique, RTT, obtained the higher and best result, thus the RTT encrypted ciphertext is most non-homogeneous/heterogeneous among all the proposed techniques and also from RSA. If degree of freedom is considered almost all the proposed techniques are obtained the optimal solution except RSA.

Frequency distribution graph has a uniform distribution with higher degree of freedom. Now, taking avalanche ratio, the proposed technique, RTT, obtained the optimal solution. This means if a single bit/byte in plaintext and or key is changed then there is a large alteration in ciphertext. Now consider encryption time, the proposed technique, RTT, shows the best result, thus the time of encryption of RTT is least than other techniques and RSA.

Table 11.1: Characteristics of microprocessor-based solutions

Techniques →	MRMKRT	RTT	RSA
Properties ↓			
Frequency Distribution Graph	√	√	-
Chi-Square Values	-	√	-
Degree of Freedom	√	√	-
Avalanche Ratio	-	√	-
Encryption Time	-	√	-
Decryption Time	-	√	-

Now, considering decryption time, the proposed technique, RTT, gives the good solution, this means time of decryption of RTT is least than other proposed techniques and RSA.

Techniques are implemented in bit-level with private/symmetric key cryptography where as RSA is public key cryptography. MRMKRT is substitution cipher where as RTT is substitution and transposition technique and RSA is substitution cipher, RTT uses Boolean as basic operation and MRMKRT uses both modulo addition (non Boolean) and Boolean as a basic operation and RSA is non-Boolean operation.

The plaintext size and ciphertext size remains same for both proposed techniques where as for RSA the plaintext size and ciphertext size are not equal. MRMKRT and RTT forms cycle where the plaintext regenerates after some finite number of iteration depends on block size and number of iteration used during encryption and for RSA plaintext never regenerates. MRMKRT, RTT and RSA used 4, 7 and 10 sub-programs respectively. MRMKRT used 9 IO/M operations, RSA uses 50 IO/M operations and RTT used 5 IO/M operations per block encryption/decryption. MRMKRT and RTT used one Boolean operation per block of encryption/decryption but MRMKRT also used 5 non Boolean operations. RSA uses 50 Boolean operations and uses 10 non-Boolean operations per block of encryption/decryption.

Table 11.2: Comparisons of MRMKRT, RTT and RSA

Characteristics ↓	Proposed Techniques →	MRMKRT	RTT	RSA
Block Cipher		√	√	√
Fixed Length Block Cipher		√	-	√
Variable Length Block Cipher		-	√	-
Implementation in Bit-Level		√	√	√
Implementation other than Bit-Stream		-	-	-
Private/Symmetric Key System		√	√	-
Substitution Technique		√	√	√
Transposition Technique		-	√	-
Boolean as Basic Operation		√	√	-
Non-Boolean as Basic Operation		√	-	√
No Alteration in Size		√	√	-
Formation of Cycle		√	√	-
Non-formation of Cycle		-	-	√
Number of sub-programs used		4	7	10
Number of IO/M operations per block of encryption/decryption		9	5	50
Number of Boolean operations used per block of encryption/decryption		1	1	50
Number of Non Boolean operations used per block of encryption/decryption		5	0	10
Calculated T-states per block of encryption/decryption		760	544	950

So, T-states calculated for MRMKRT, RTT and RSA are 760, 544 and 950 respectively. Thus it can be said that in microprocessor based implementation perspective RTT is the faster than MRMKRT and RSA in terms of execution speed per block of encryption/decryption.

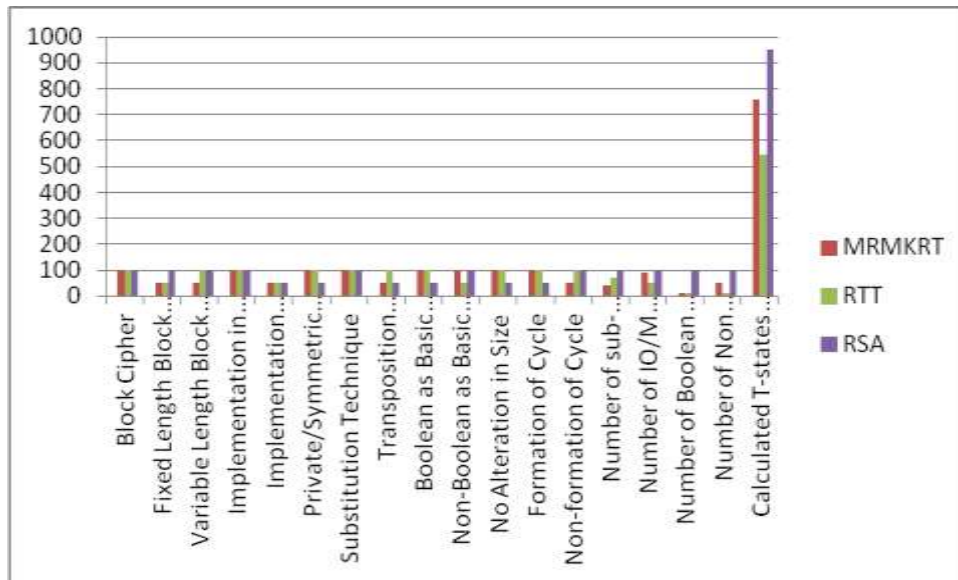


Figure 11.1: Graphical representation of comparisons of MRMKRT, RTT and RSA

Thus RTT is giving the optimal solution in respect to microprocessor based implementation. Table 11.2 and figure 11.1 summarize these discussions.

Table 11.3 illustrates the overall conclusion scenario for FPGA based solutions. In this table the six proposed algorithms/techniques are compared along with the existing, renowned and industrially accepted RSA. The symbol “√” shows the optimal solution got against a property or the best solution. Here also seven properties are considered for the evaluation, in this chapter and also throughout the thesis. The proposed techniques are TPRT, TMAT, ROBAST, SRAT, TSV and FBOMAT. The properties taken are frequency distribution graph, Chi-Square values for non-homogeneity, degree of freedom, avalanche ratio, encryption time, decryption time and simulation based results.

Taking frequency distribution graph, it is seen that the all proposed algorithm obtain the optimal solution that means the frequencies are well distributed in ciphertext as compared to plaintext; here exception is the TSV and RSA, whose frequency is not well distributed. Now taking the Chi-Square values, the proposed, MFBOMAT, obtain the higher and best result, thus the MFBOMAT encrypted ciphertext is most non-homogeneous/heterogeneous among all the proposed technique and also from RSA. Now taking the degree of freedom, almost all the proposed techniques are obtained the optimal solution except TSV and RSA, so, this result is at par with the result of frequency distribution graph. Now consider avalanche ratio, the technique, ROBAST and TSV, obtained the optimal solution.

Table 11.3: Characteristics of FPGA-based solutions

Techniques →	TPRT	TMAT	ROBAST	Shuffle- RAT	TSV	RSA	MFBOMAT
Properties ↓							
Frequency Distribution Graph	√	√	√	√	-	-	√
Chi-Square Values	-	-	√	-	-	-	√
Degree of Freedom	√	√	√	√	-	-	√
Avalanche Ratio	-	-	√	-	√	-	-
Encryption Time	-	√	-	-	-	-	√
Decryption Time	-	-	-	-	√	-	√
Simulation Based Results	√	-	-	-	-	-	√

As these two techniques are implemented as Cipher Block Chaining (CBC) mode, so, the result is also at par with the theory of cryptography. This means if a bit/byte in plaintext and or key is changed then there is a large change in ciphertext. Now taking encryption time, the proposed technique, MFBOMAT, shows the best result, thus the time of encryption of MFBOMAT is least than other techniques and RSA. In terms of decryption time, the proposed, MFBOMAT, gives the best solution. In terms of simulation based results, this property is based on results of RTL schematic, less number of Look-Up-Tables thus less area, less time slices, less timing simulation parameters, thus the proposed, MFBOMAT, gives the best solution in this respect, it is obvious because the MFBOMAT is the simplest technique than all the other proposed techniques.

Table 11.4: HDL synthesis report (Netlist generation of RSA, TPRT, TMAT, ROBAST, SRAT, TSV and MFBOMAT)

Sr No.	Netlist Components	Number						
		RSA	TPRT	TMAT	ROBAST	SRAT	TSV	MFBOMAT
1	ROMs/RAMs	430	10	14	25	28	12	09
2	Adders/Subtractions	3	0	2	20	28	0	15
3	Registers	420	20	30	50	641	10	10
4	Latches	80	0	0	10	80	0	0
5	Multiplexers	120	0	0	10	136	0	0

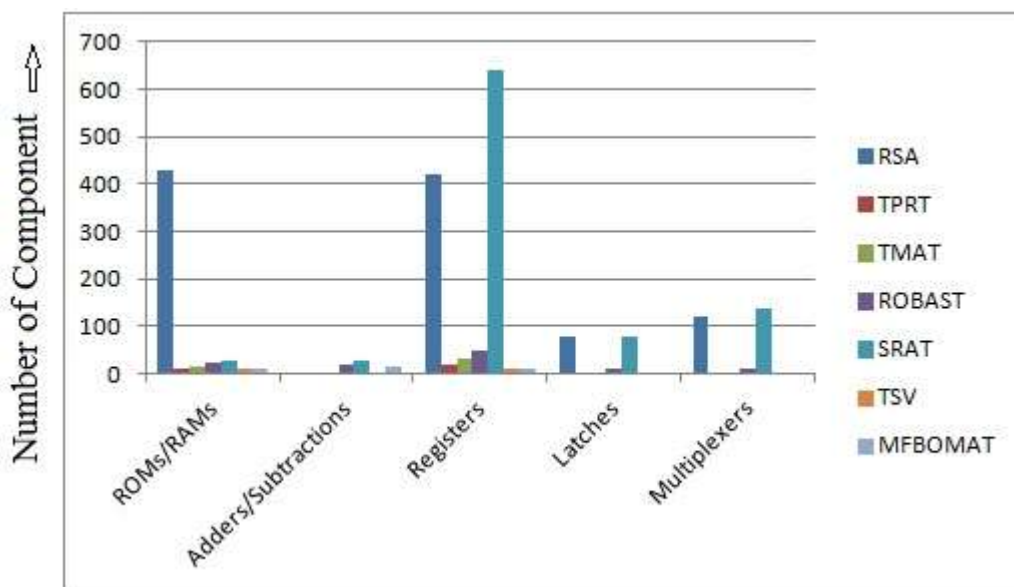


Figure 11.2: Pictorial representation of HDL synthesis report of net-list generation

Table 11.4 illustrates the hardware implementation analysis of MFBOMAT, RSA, TPRT, TMAT, ROBAST, SRAT and TSV. RSA uses 430, TPRT uses 10, TMAT uses 14, ROBAST uses 25, SRAT uses 28, TSV uses 12 and MFBOMAT uses 9 numbers of ROMs/RAMs. RSA, TPRT, TMAT, ROBAST, SRAT, TSV and MFBOMAT uses 3, nil, 2, 20, 28, nil and 15 adders/substrations respectively. RSA uses 420, TPRT uses 20, TMAT uses 30, ROBAST uses 50, SRAT uses 641, TSV uses 10 and MFBOMAT uses 10 numbers of registers. RSA uses 80, ROBAST uses 10, SRAT uses 80 and others use nil number of latches. RSA uses 120, ROBAST uses 10, SRAT uses 136 and others use nil number of latches. Thus from these analysis we can conclude that MFBOMAT is giving optimal result

in terms of HDL synthesis of net list generation for FPGA-based implementation. Figure 11.2 gives the summarized result.

Table 11.5 illustrates the entire timing summary obtained after HDL synthesis. The speed grade and maximum frequency is same as all the techniques/algorithms have been implemented in Xilinx Spartan-3E XC3S100E-5VQ100 (package: VQ100, speed grade: -5).

Table 11.5: HDL synthesis report (Timing summary of RSA, TPRT, TMAT, ROBAST, SRAT and TSV)

Sr No.	Timing Constraint	Values						
		RSA	TPRT	TMAT	ROBAST	SRAT	TSV	MFBOMAT
1	Speed Grade	-5	-5	-5	-5	-5	-5	-5
2	Minimum period (ns)	9.895	5.66	7.95	5.55	5.50	10.22	4.99
3	Maximum Frequency (MHZ)	101.06	101.06	101.06	101.06	101.06	101.06	101.06
4	Minimum input arrival time before clock (ns)	6.697	4.33	5.55	5.55	4.25	6.66	4.20
5	Maximum output required time after clock (ns)	4.31	3.33	4.25	4.44	3.33	5.55	3.30

MFBOMAT is obtained minimum period of 4.99ns followed by RSA 9.89ns, TPRT 5.66ns, TMAT 7.95ns, ROBAST 5.55ns, SRAT 5.50ns and TSV 10.22ns. MFBOMAT is also require minimum input arrival time before clock of 4.20ns followed by RSA 6.70ns, TPRT 4.33ns, TMAT 5.55ns, ROBAST 5.55ns, SRAT 4.25ns and TSV 6.66ns.

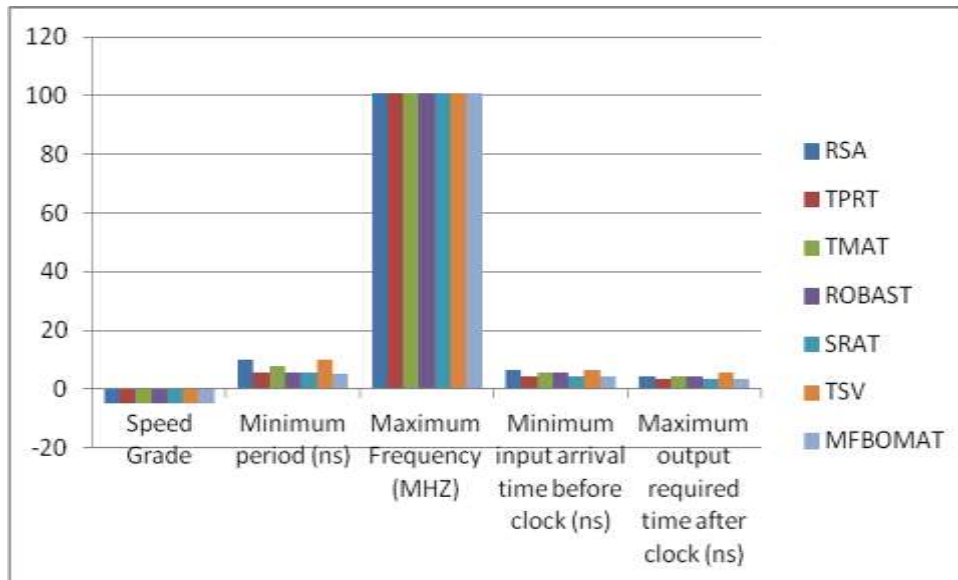


Figure 11.3: Pictorial representation of HDL synthesis report of timing summary

MFBOMAT requires minimum value in maximum output required time after clock of 3.30ns followed by RSA 4.31ns, TPRT 3.33ns, TMAT 4.25ns, ROBAST 4.44ns, SRAT 3.33ns and TSV 5.55ns. Thus from these analysis we can conclude that MFBOMAT is giving optimal result in terms of HDL synthesis of timing summary for FPGA-based implementation. Figure 11.3 gives the summarized result. Therefore from the discussions of HDL synthesis report of both netlist generation and timing summary it can be concluded that MFBOMAT is the optimal solution for FPGA-based implementation.

11.2 The Future Work

Authors have implemented eight techniques and compared with RSA. There are other existing algorithm exist such as TDES and AES. Author have also not carried out cryptanalysis of these techniques. The following are the main points where a further research may be carried out in future.

- Design and implementation of public key cryptography in FPGA and microprocessor based system.
- Design and implementation of Elliptic Curve Cryptography in microprocessor based and FPGA-based systems.

- Study on cryptanalysis like differential attack, linear attack, power analysis of crypto-hardware/crypto-processor, key boomerang attack, side channel attack, improved meet-in-the-middle attack etc.
- FPGA-based systems like implementation of fast, secure crypto solution for FPGA(s), FPGA-based TLC schemes, design of firewalls, gateways in FPGA-based systems, design of memory systems in FPGA(s) etc.

In this thesis eight novel techniques are proposed, two of them are realized in microprocessor-based systems and six of them are realized in FPGA-based systems. MRMKRT and TPRT gives better result in frequency distribution graph analysis and degree of freedom analysis than RSA. MFBOMAT gives better result in FPGA simulation based analysis and time complexity analysis than RSA. ROBAST gives better result in Chi-Square value analysis (non-homogeneity) than RSA. SRAT adds better confusion and diffusion cryptographic properties. TSV gives better avalanche ratio analysis than RSA.

Therefore, proposed models can be used in modern Information and Communication Technology (ICT) and Information Technology Enabled Services (ITES) for providing the primary goal of data/information confidentiality.

References

1. Chester Rebeiro and Debdeep Mukhopadhyaya, “High Speed Compact Elliptic Curve Crypto-processor for FPGA platforms”, Progress in Cryptology – INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008, LNCS 5365, Springer.
2. Mridul Nandi, “Two New Efficient CCA-Secure Online Ciphers: MHCBC and MCBC”, Progress in Cryptology – INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008, LNCS 5365, Springer.
3. Jens-Peter Kaps, “Chai-Tea, Cryptographic Hardware Implementation of xTEA”, Progress in Cryptology – INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008, LNCS 5365, Springer.
4. Daniel J Berustine and Peter Schwabe, “New AES Software Speed Record”, Progress in Cryptology – INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008, LNCS 5365, Springer.
5. J Lu, Orr Dunkelman, Nathan Keller and J Kim, “New Impossible Differential Attack on AES”, Progress in Cryptology – INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008, LNCS 5365, Springer.
6. M Gorski and S Lucks, “New Related Key Boomerang Attacks on AES”, Progress in Cryptology – INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008, LNCS 5365, Springer.
7. M. Agarwal, S. Karmakar, D. Saha and Debdeep Mukhopadhyaya, “Scan Based Side Channel Attack on Stream Ciphers and Their Counter-Measures”, Progress in Cryptology – INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008, LNCS 5365, Springer.
8. Qiang Tang, “Type-Based Proxy Re-encryption and Its Construction”, Progress in Cryptology – INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008, LNCS 5365, Springer.

9. Y Ren and D Gu, "Secure Hierarchical Identity Based Encryption Scheme in the Standard Model", Progress in Cryptology – INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008, LNCS 5365, Springer.
10. DP Schmid and A Biryukov, "Slid Pairs in Salsa20 and Trivium", Progress in Cryptology – INDOCRYPT 2008, 9th International Conference on Cryptology in India, Kharagpur, India, December 14-17, 2008, LNCS 5365, Springer.
11. A Simmonds, Peter Sandilands and LV Ekert, "An Ontology for Network Security Attacks", Applied Computing, Second Asian Applied Computing Conference, AACC 2004, Kathmandu, Nepal, October 29-31, 2004, LNCS 3285, Springer.
12. S Dutta and JK Mandal, "Ensuring e-Security Using a Private-Key Cryptographic System Following Recursive Positional Modulo-2 Substitution", Applied Computing, Second Asian Applied Computing Conference, AACC 2004, Kathmandu, Nepal, October 29-31, 2004, LNCS 3285, Springer.
13. SJ Mamagi, P Chaurasia and MP Sing, "Merging of RC5 with AES – Incorporating More Flexibility and Security to AES", 12th International Conference on Information Technology, ICIT – 2009, Bhubaneswar, India, December 21- 24, 2009, IEEE.
14. S Ghosh, S Ray, R Chobasia, "Chaotic Cryptography using External Key", 12th International Conference on Information Technology, ICIT – 2009, Bhubaneswar, India, December 21- 24, 2009, IEEE.
15. PK Jha, JK Mandal, "Cascaded Encryption Through Recursive Carry Addition and Key rotation (CRCAKR) of a Session Key", 9th International Conference on Information Technology, ICIT – 2006, Bhubaneswar, India, December 18-21, 2006, IEEE and IEEE Computer Society.
16. Nalini N, R Rao G, "Cryptanalysis of Block Cipher via Improved Simulated Annealing Technique", 9th International Conference on Information Technology, ICIT – 2006, Bhubaneswar, India, December 18-21, 2006, IEEE and IEEE Computer Society.

17. S Sinha, JK Mandal and R Chakraborty, "A Microprocessor-Based Block Cipher through Overlapped Modulo Arithmetic Technique (OMAT)", 12th International Conference on Advanced Computing and Communication, ADCOM 2004, December 15-18, 2004, Ahmedabad, INDIA, IEEE Gujarat Section.
18. HS Dutta and A Dutta, "Bit-stream Authentication Technique for FPGA Security", International Conference on Information Technology, INTL-INFOTECH 2007, March 19-21, 2007, Haldia, West Bengal, India, ISSN 0973-6824.
19. BK Upadhaya, P Vanketasewaran, SK Sanyal and R Nandi, "A Novel FPGA based TLC Scheme", International Conference on Information Technology, INTL-INFOTECH 2007, March 19-21, 2007, Haldia, West Bengal, India, ISSN 0973-6824.
20. GH Mondal, P Chakraborti and CT Bhuiyan, "Various New and Modified Approaches for Encryption and Their Comparative Study", International Conference on Information Technology, INTL-INFOTECH 2007, March 19-21, 2007, Haldia, West Bengal, India, ISSN 0973-6824.
21. C Reddy, R Arora, M Kedia, I Sen Gupta, "Design and Implementation of a Packet Level Firewall on FPGA", International Conference on Information Technology, INTL-INFOTECH 2007, March 19-21, 2007, Haldia, West Bengal, India, ISSN 0973-6824.
22. B Bhuyan, T Bhunia, P Chakraborti, SR Bhadra Chaudhuri, and Atal Chaudhuri, "Practical Realization of Selective DES and Results Thereof", International Conference on Information Technology, INTL-INFOTECH 2007, March 19-21, 2007, Haldia, West Bengal, India, ISSN 0973-6824.
23. Ramkrishna Das, Durbadal Mandal, A.K. Bhatteerjee, P. Paul and S. Dutta, "An Evolutionary Approach in Developing Efficient Ciphering System", International Conference on Information Technology, INTL-INFOTECH 2007, March 19-21, 2007, Haldia, West Bengal, India, ISSN 0973-6824.

24. P Paul, S Dutta, AK Bhattacharya, "Ensuring Information Security through a 491-Bit Storage-Efficient Substitution-Based Block Cipher of Bit-Level Implementation", International Conference on Information Technology, INTL-INFOTECH 2007, March 19-21, 2007, Haldia, West Bengal, India, ISSN 0973-6824.
25. P Paul, S Dutta, AK Bhattacharya, "An Attempt to ascertain Security of Information through a Non Boolean-based Storage-Efficient Private-Key Block Cipher", International Conference on Information Technology, INTL-INFOTECH 2007, March 19-21, 2007, Haldia, West Bengal, India, ISSN 0973-6824.
26. K Rahimunnisa, MA Lincy, "A Hardware Implementation of Three Standard Cryptography Algorithms on an Universal Architecture for the Application of Smart Cards", International Conference on Information Technology, INTL-INFOTECH 2007, March 19-21, 2007, Haldia, West Bengal, India, ISSN 0973-6824.
27. PK Jha, JK Mandal, S Shakya, "Encryption Through Cascaded Recursive Key Rotation of a Session Key and Addition of Blocks (CRKRAB)", International Conference on Information Technology, INTL-INFOTECH 2007, March 19-21, 2007, Haldia, West Bengal, India, ISSN 0973-6824.
28. R Saram, M Khomdram, "Juxtaposition of RSA and Elliptic Curve Cryptosystem", International Journal of Computer Science and Network Security, IJCSNS, Volume 9. Number 9, September 2009, ISSN 1738-7906.
29. MH Rais and SM Qasim, "Efficient Hardware Realization of Advanced Encryption Standard Algorithm using Virtex-5 FPGA", International Journal of Computer Science and Network Security, IJCSNS, Volume 9. Number 9, September 2009, ISSN 1738-7906.
30. K Raj, B kumar, P Mittal, "FPGA Implementation and Mask Level CMOS Layout Design of Redundant Binary Signed Digit Comparator", International Journal of Computer Science and Network Security, IJCSNS, Volume 9. Number 9, September 2009, ISSN 1738-7906.

31. P Uppuluri, V Dwakara, A Tangaonkar, V Rajeagowda, "Securing the interaction between X clients", International Journal of Computer Science and Network Security, IJCSNS, Volume 9. Number 9, September 2009, ISSN 1738-7906.
32. MH Rais, SM Qasim, "A Novel Implementation of AES-128 using Reduced Residue of Prime Numbers based S-Box", International Journal of Computer Science and Network Security, IJCSNS, Volume 9. Number 9, September 2009, ISSN 1738-7906.
33. PK Jha and JK Mandal, "Cascaded Recursive Key Rotation and Key Arithmetic of A Session Key (CRKRKA)", International Journal of Intelligent Information Processing, 2(1) January-June 2008, pp- 9-20, ISSN 2093-1964.
34. S Dutta and JK Mandal, "Ensuring Information Security through 123-bit recursive substitution of bits through prime-nonprime detection of sub-stream (RSBP)", Journal of Scientific and Industrial Research, Volume 68, July – 2009, pp 584-591, ISSN 0022-4456.
35. PK Jha, JK Mandal, "A Bit Level Symmetric Encryption Technique Through Recursive key Rotation (RKR) of a Session Key", AMSE International Journal, Volume 2, 2007.
36. PK Jha, JK Mandal, S Shakya, "A Bit Level Symmetric Encryption Technique through Recursive Transposition Operation (RTO) to Enhance the Security of Transmission", Journal of the Institute of Engineering, Volume 5, No. 1, pp 106-116, ISSN 1810-3383, 2005.
37. PK Jha and JK Mandal, "Encryption through Cascaded Recursive Bit wise and Carry Addition on Blocks (CRBOCAB) of A Session Key", International Conference on Advances in Computer Vision and Information Technology, IEEE, IEEE Mumbai Section, 2008.
38. AK Sharma, CS Lamba, "Network Security and Networking Protocol", International Conference on Advances in Computer Vision and Information Technology, IEEE, IEEE Mumbai Section, 2008.

39. UK Mondal, SN Mandal, J PalChoudhury. JK Mandal, “Frame Based Symmetric Key Cryptography”, *Int. J. Advanced Networking and Applications*, Volume: 02, Issue: 04, Pages: 762-769 (2011), ISSN: 09750290, EISSN: 09750282.
40. Dr. Sastry JKR, Prof K. SubbaRao, Prof N Venkata Ram, Ms.J. Sasi Bhanu, “Attacking Embedded Systems through Power Analysis”, *Int. J. Advanced Networking and Applications*, Volume: 02, Issue: 04, Pages: 762-769 (2011), ISSN: 09750290, EISSN: 09750282.
41. Arturo Diaz-Perez, Nazar A. Saqib, and Francisco Rodriguez-Henriquez, “Some Guidelines for Implementing Symmetric-Key Cryptosystems on Re-configurable-Hardware”, Information Security Laboratory, Oregon State University, [http://islab.oregonstate.edu/papers/FRH/DiazNzRdgz-DES\(9\).pdf](http://islab.oregonstate.edu/papers/FRH/DiazNzRdgz-DES(9).pdf), published on 30-Jun-2004.
42. Jerome Burke, John McDonald, Todd Austin, “Architectural Support for Fast Symmetric-Key Cryptography”, *ACM journal*, Nov. 12-15, 2000.
43. Whitfield Diffie and Martin E. Hellman, “New Directions in Cryptography”, *IEEE Information Theory Workshop*, Lenox, MA, June 23–25, 1975 and the *IEEE International Symposium on Information Theory* in Ronneby, Sweden, June 21–24, 1976.
44. Philippe Paquet, “Sinople: a 128-bit symmetric block cipher”, Draft – 2, Revised, March 14, 2003, philippe@paquet.net, <http://philippe.paquet.net/sinople>.
45. V Canda, TV Trung, S Maglivaras, T Horvath, “Symmetric Block Ciphers Based on Group Bases”, *Selected Areas in Cryptography, SAC'2000*, Ed. D. Stinson and S. Tavares, LNCS, (2001) pp.98—105.
46. Reiner Dojen and Tom Coffey, “Applying Conditional Linear Cryptanalysis to Ciphers with Key- Dependant Operations”, *WSEAS TRANSACTIONS ON COMPUTERS* Issue 5, Volume 3, November 2004 ISSN: 1109-2750, pp 1425-1430.
47. Olivier Billet and Henri Gilbert, “A Traceable Block Cipher”, Appeared in C. -S. Laih (Ed.): *ASIACRYPT 2003*, LNCS 2894, pp. 331–346, 2003. Springer-Verlag Berlin Heidelberg 2003 IACR 2003.

48. Andrey Bogdanov and Vincent Rijmen, "Zero-Correlation Linear Cryptanalysis of Block Ciphers", Cryptology ePrint Archive, 2011.
49. Lei Wei, Christian Rechberger, Jian Guo, Hongjun Wu, Huaxiong Wang, and San Ling, "Improved Meet-in-the-Middle Cryptanalysis of KTANTAN", Cryptology ePrint Archive, 2011.
50. Celine Blondeau and Benot Gerard, "Multiple Differential Cryptanalysis: Theory and Practice", Cryptology ePrint Archive, 2011.
51. Farzaneh Abazari, Babak Sadeghian, "Cryptanalysis with Ternary Difference: Applied to Block Cipher PRESENT", Cryptology ePrint Archive, 2011.
52. Said E. El-Khamy¹, Fellow IEEE, M. Abou El-Nasr², Member IEEE, and Amina El-Zein³, "A Highly Secure Chaotic Stream Cipher Using Map and Orbit Hopping", 25th National Radio Science Conference (NRSC 2008), March 18-20, 2008, Faculty of Engineering, Tanta Univ., Egypt, ieeexplore.ieee.org.
53. Miodrag J. Mihaljevic and Hideki Imai, "A Stream Cipher Design Based on Embedding of Random Bits", International Symposium on Information Theory and its Applications, ISITA 2008, Auckland, New Zealand, 7-10, December, 2008, ieeexplore.ieee.org.
54. C .S Lamba, "Design and Analysis of Stream Cipher for Network Security", Second International Conference on Communication Software and Networks, 2010, ieeexplore.ieee.org.
55. Yunyi Liu, Tuanfa Qin, "The Key and IV Setup of the Stream Ciphers HC-256 and HC-128", International Conference on Networks Security, Wireless Communications and Trusted Computing, 2009, ieeexplore.ieee.org.
56. Tieming Chen and Liang Ge, "TinyStream: A Lightweight and Novel Stream Cipher Scheme for Wireless Sensor Networks", International Conference on Computational Intelligence and Security, 2010, ieeexplore.ieee.org.
57. Sheena Mathew, K. Paulose Jacob, "A New Fast Stream Cipher: MAJE4", -- Indco 2005- Cofeene Chnn India 1-13 Dec 2005, pp 60-63, ieeexplore.ieee.org.

58. Lin Gan, Stan Simmons and Stafford Tavares, "A New Family of Stream Ciphers Based on Cascaded Small S-Boxes", pp 0053 – 0058, ieeexplore.ieee.org, IEEE Conferences, 2000.
59. Shai Halevi, Don Coppersmith and Charanjit Jutla, "Scream: a software-efficient stream cipher", June 5, 2002, Stony Brook University, Dept of Computer Science, <http://www.cs.sunysb.edu/>.
60. Zhang Peng and Jia Jian Fang, "Comparing and Implementation of Public Key Cryptography Algorithms on Smart Card", International Conference on Computer Application and System Modeling (ICCASM 2010), ieeexplore.ieee.org.
61. Kashif Latif, Athar Mahboob, Nassar Ikram, "A Parameterized Design of Modular Exponentiation on Reconfigurable Platforms for RSA Cryptographic Processor", Publication Year: 2009, Page(s): 200 - 205 IEEE Conferences, ieeexplore.ieee.org.
62. Irma B. Fernandez, Wunnavu V. Subbarao, "Encryption based Security for ISDN Communication: Technique and Application", 0-7803-1797-1/94/1994 IEEE, ieeexplore.ieee.org.
63. Jin Park, Jeong-Tae Hwang, Young-Chul Kim, "FPGA and ASIC Implementation of ECC Processor for Security on Medical Embedded System", Proceedings of the Third International Conference on Information Technology and Applications (ICITA'05) 0-7695-2316-1/05 2005 IEEE, ieeexplore.ieee.org.
64. Yunpeng Zhang and XiaLin, "Fast Public Key Algorithm of Knapsack Type", International Conference on Computer and Communication Technologies in Agriculture Engineering, 2010, ieeexplore.ieee.org.
65. KT Ng, WN Chau, and YM Siu, "An Internet Security System for E-commerce", 0-7803-7474-6/02/0017.000 2002 IEEE, ieeexplore.ieee.org.
66. Jing-Shyang Hwu, Rong-Jaye Chen, Member, IEEE, and Yi-Bing Lin, Fellow, IEEE, "An Efficient Identity-based Cryptosystem for End-to-end Mobile Security", IEEE Transactions On Wireless Communications, vol. 5, no. 9, September 2006, ieeexplore.ieee.org.

67. Simson L. Garfinkel, "Public key cryptography", Internet Kiosk, June 1996, ieeexplore.ieee.org.
68. Frank E. Ferrante, "Maintaining Security and Privacy of Patient Information", MSEE, MSEPP, Proceedings of the 28th IEEE EMBS Annual International Conference New York City, USA, Aug 30-Sept 3, 2006, ieeexplore.ieee.org.
69. "Security at Internet layer", Cover feature, 0018-9162/98 1998 IEEE, September 1998, ieeexplore.ieee.org.
70. HoWon Kim, Member, IEEE, and Sunggu Lee, Member, IEEE, "Design and Implementation of a Private and Public Key Crypto Processor and Its Application to a Security System", IEEE Transactions on Consumer Electronics, Vol. 50, No. 1, February 2004, ieeexplore.ieee.org.
71. Zeng Ping, Hu Ronglei, Fang Yong, Yang Jianxi, Liu Yue, "A key management scheme for Ad hoc networks", 978-1-4244-3693-4/09 2009 IEEE, ieeexplore.ieee.org.
72. Byoung-Jo "J" Kim, Srividhya Srinivasan, "Simple Mobility Support for IPsec Tunnel Mode", AT&T Labs-Research, macsbug@research.att.com, Georgia Institute of Technology, vidya@cc.gatech.edu, 0-7803-7954-3/03 2003 IEEE, ieeexplore.ieee.org.
73. Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, and Tadayoshi Kohno, "Experimental Security Analysis of a Modern Automobile", Appears in IEEE Symposium on Security and Privacy. 2010. <http://www.autosec.org/>.
74. M. Blaze, "High-Bandwidth Encryption with Low-Bandwidth Smartcards", Cambridge Workshop on Fast Software Encryption, February 1996, <http://www.crypto.com/papers/>, 1995.
75. Matt Blaze, "Key Management in an Encrypting File System", USENIX Summer 1994 Technical Conference, Boston, MA, June 1994, <http://www.crypto.com/papers/>.

76. M. Bednara, M. Daldrup, J. Teich, J. von zur Gathen, J. Shokrollahi, "Tradeoff Analysis of FPGA Based Elliptic Curve Cryptography", Proceedings of the IEEE International Symposium on Circuits and Systems. 2002. ISCAS'02, 5:797–800, 2002.
77. Gael Rouvroy, Francois-Xavier Standaert, Jean-Jacques Quisquater and Jean-Didier Legat, "Compact and Efficient Encryption/Decryption Module for FPGA Implementation of the AES Rijndael Very Well Suited for Small Embedded Applications", 18-Dec-2003, <http://www.dice.ucl.ac.be/~fstandae/PUBLIS/15.pdf>.
78. M. Riaz and HM Heys, "The FPGA Implementation of the RC6 and CAST-256 Encryption Algorithms", Faculty of Engineering and Applied Science, Memorial University, Canada, 22-Jun-2004, <http://www.engr.mun.ca/~howard/PAPERS/fpga.pdf>.
79. "Protecting FPGAs from Power Analysis – Cryptography Research White paper", by Cryptography Research Inc, Version 1.0 April 20, 2010, <http://www.cryptography.com/public/pdf/FPGAsecurity.pdf>.
80. Ismail San and Nuray At, "Enhanced FPGA Implementation of the Hummingbird Cryptographic Algorithm", Cryptology ePrint Archive, 2010, <http://eprint.iacr.org/2010/>.
81. John Fry and Martin Langhamme, "RSA and Public Key Cryptography in FPGAs", www.altera.com, 2005.
82. M. Huang, K. Gaj, S. Kwon, and T. El-Ghazawi, "An Optimized Hardware Architecture for the Montgomery Multiplication Algorithm", PKC 2008: 11th International Workshop on Practice and Theory in Public Key Cryptography, Barcelona, Spain, pages 214-228, March, 2008, LNCS 4939.
83. R. Lien, T. Grembowski, and K. Gaj, "A 1 Gbit/s Partially Unrolled Architecture of Hash Functions SHA-1 and SHA-512", RSA Conference, Cryptographer's Track, CT-RSA 2004, San Francisco, CA, LNCS, volume 2964, pages 324–328, Feb., 2004.

84. Kris Gaj and Pawel Chodowiec, "Comparison of the hardware performance of the AES candidates using re-configurable hardware", The Third Advanced Encryption Standard Candidate Conference, April 13-14, 2000, New York, New York, USA. National Institute of Standards and Technology, Entire proceeding, <http://csrc.nist.gov/encryption/aes/round2/conf3/aes3conf.htm>, Contents.
85. JP. Kaps and C. Paar, "Fast DES implementations for FPGAs and its application to a universal key-search machine", Selected Areas in Cryptography, 5th Annual International Workshop, SAC'98, Proceedings, Lecture Notes in Computer Science (LNCS), volume 1556, Queen's University, Kingston, Ontario, Canada, Springer-Verlag, Berlin, pages 234–247, 1999.
86. Mike Hutton, Jay Schleicher, David Lewis, Bruce Pedersen, Richard Yuan, Sinan Kaptanoglu, Gregg Baeckler, Boris Ratchev, Ketan Padalia, Mark Bourgeault, Andy Lee, Henry Kim and Rahul Saini, "Improving FPGA Performance and Area Using an Adaptive Logic Module", FPL 2004, LNCS 3203, pp. 135–144, 2004. Š Springer-Verlag Berlin Heidelberg 2004.
87. Pawel Chodowiec and Kris Gaj, "Very Compact FPGA Implementation of the AES Algorithm", C.D. Walter et al. (Eds.): CHES 2003, LNCS 2779, pp. 319–333, 2003. Springer-Verlag Berlin Heidelberg 2003.
88. Arshad Aziz and Nassar Ikram, "An FPGA-based AES-CCM Crypto Core For IEEE 802.11i Architecture", International Journal of Network Security, Vol.5, No.2, PP.224–232, Sept. 2007, ISSN 1816-353X (Print), ISSN 1816-3548 (Online).
89. Pranam Paul, Saurabh Dutta and A K Bhattacharjee, "Enhancement of Security through an Efficient Substitution based Block Cipher of Bit-level Implementation with Possible Loss-less Compression", IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.4, April 2008, Journal ISSN: 1738-7906.
90. Mrinmoy Ghosh and Prof. Pranam Paul, "An Application to ensure Security through Bit-level Encryption", IJCSNS International Journal of Computer Science and Network Security, VOL.9 No.11, November 2009, Journal ISSN: 1738-7906.

91. Lin Han, Jun Han, Xiaoyang Zeng, Ronghua Lu, Jia Zhao, "A Programmable Security Processor for Cryptography Algorithms", 978-1-4244-2186-2/08 2008 IEEE, ieeexplore.ieee.org.
92. Xiao-Hui Yang, Zi-Bin Dai and Xue-Rong Yu, "The Research And Design Of Reconfigurable Cryptographic Chip Based On Block Cipher", 1-4244-0161-5/06 2006 IEEE, ieeexplore.ieee.org.
93. Jayanta Kumar Pal, J. K. Mandal, "A Random Block Length Based Cryptosystem through Multiple Cascaded Permutation- Combinations and Chaining of Blocks", Fourth International Conference on Industrial and Information Systems, ICIIS 2009, 28 - 31 December 2009, Sri Lanka, ieeexplore.ieee.org.
94. Jayanta Kumar Pal, J. K. Mandal, "A Novel Block Cipher Technique Using Binary Field Arithmetic Based Substitution (BCTBFABS)", Second International conference on Computing, Communication and Networking Technologies, 2010, ieeexplore.ieee.org.
95. Robert M. Best, "Preventing Software Piracy With Crypto-Microprocessor", Princeton University, Dept. of Computer Science, 19 Nov 2004, http://www.princeton.edu/~rblee/ELE572Papers/Fall04Readings/CryptoProc_Best.pdf, 1980.
96. HoWon Kim, Member, IEEE, and Sunggu Lee, Member, IEEE, "Design and Implementation of a Private and Public Key Crypto Processor and Its Application to a Security System", IEEE Transactions on Consumer Electronics, Vol. 50, No. 1, FEBRUARY 2004, ieeexplore.ieee.org.
97. Ingrid Verbauwhede, Frank Hoornaert, Joos Vandewalle, "Security and Performance Optimization of a New DES Data Encryption Chip", IEEE Journal of Solid-State Circuits, VOL. 23, NO. 3. June 1988, ieeexplore.ieee.org.
98. N. D. Goots, N. A. Moldovyan, P. A. Moldovyanu, and D. H. Summerville, "Fast DDP-Based Ciphers: from Hardware to Software", 0-7803-8294-3/04 2004 IEEE, ieeexplore.ieee.org.

99. Tarun Kochar, Sukumar Nandi and Santosh Biswas, "A Single chip implementation of AES cipher and Whirlpool hash function", 978-1-4244-4859-3/09/ 2009, ieeexplore.ieee.org.
100. Christos F Sotiriou and Yannis Papaefstathiou, "Design-Space Exploration of A Cryptography Algorithm", 0-7803-8163-7/03 2003 IEEE, ICECS-2003, ieeexplore.ieee.org.
101. Carl M. Campbell, "A_Microprocessor-Based Module to Provide Security in Electronic Funds Transfer Systems", CH1465-4/79/0000-0148 1979 IEEE, ieeexplore.ieee.org.
102. G. I. Davida, D. L. Wells, "Microprocessors and Data Encryption", CH1465-4/79/0000-0154 1979 IEEE, ieeexplore.ieee.org.
103. Tao Chen, Bin Yu, Jin-Hai Su, Zi-bin Dai, Jian-Guo Liu, "A Reconfigurable Modular Arithmetic Unit for Public-key Cryptography", 1-4244-1132-7/07/ 2007 IEEE, ieeexplore.ieee.org.
104. Christian Muller-Schloer and Siemens AG, "A Microprocessor Based Cryptoprocessor", 0272-1732/83/1000-0005 1983 IEEE, ieeexplore.ieee.org.
105. Sukumar S. Raghuram and Chaitali Chakrabarti, "A Programmable Processor for Cryptography", ISCAS 2000 - IEEE International Symposium on Circuits and Systems, May 28-31, 2000, Geneva, Switzerland, ieeexplore.ieee.org.
106. Dan Fay, Alex Shye, Sayantan Bhattacharya, Daniel A. Connors and Steve Wichmann, "An Adaptive Fault-Tolerant Memory System for FPGA-based Architectures in the Space Environment", University of Colorado a Boulder: Computer Engineering Research, 2007 NASA/ESA Conference on Adaptive Hardware Systems, August 2007, <http://ce.colorado.edu/Publications/>.
107. Vyacheslav Kharchenko, Olexander Siora, Volodymyr Sklyar, "Design and Testing Technique of FPGA-Based Critical Systems", CADSM'2009, 24-28 February, 2009, Polyana-Svalyava (Zakarpattya), UKRAINE, ieeexplore.ieee.org.

108. Xuemei LI, Qiuchen YUAN, Wuchen WU, Xiaohong PENG, Ligang HOU, "Implementation of GSM SMS Remote Control System Based on FPGA", 978-1-4244-7618-3 /10/ IEEE, ieeexplore.ieee.org, 2010.
109. GODWIN U, "A Microprocessor – Based Tunneling Machine Controller", W/CH 2935-5/90/~1879 1990 IEEE, ieeexplore.ieee.org.
110. Kamalnayan Jayaraman, Vivekananda M. Vedula and Jacob A. Abraham, "Native Mode Functional Self-Test Generation for Systems-on-Chip", Proceedings of the International Symposium on Quality Electronic Design (ISQED.02) 0-7695-1561-4/02 2002 IEEE, IEEE Computer Society, ieeexplore.ieee.org.
111. Mudduveerappa, S.Nagaraja.Rao, H.D.Maheshappa and R.Sriram, "Microprocessor Based Audiometer for Mass Screening", IEEE Engineering in Medicine and Biology Society 10th Annual International Conference—1625 CH2566-BjBBjOOOO--1625 1988 IEEE, ieeexplore.ieee.org.
112. Dr.V.P.Ramamurth, Mr.V.Vijayakumar, Mr S. Saravanan, "Design of A 8085 Microprocessor Controller for A Three Phase Converter With Fourth Leg", 1980, ieeexplore.ieee.org.
113. Zhining Lim and Braden J. Phillips, "An RNS-Enhanced Microprocessor Implementation of Public Key Cryptography", 978-1-4244-2110-7/08/2007 IEEE, ieeexplore.ieee.org.
114. Yadollah Eslami, Member, IEEE, Ali Sheikholeslami, Senior Member, IEEE, P. Glenn Gulak, Senior Member, IEEE, Shoichi Masui, Member, IEEE, and Kenji Mukaida, "An Area-Efficient Universal Cryptography Processor for Smart Cards", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, VOL. 14, NO. 1, January 2006, ieeexplore.ieee.org.
115. Peter A Ivey, Simon N Walker, Jon M Stern and Simon Davidson, "An Ultra-High Speed Public Key Encryption Processor", IEEE 1992 Custom Integrated Circuits Conference, ieeexplore.ieee.org.
116. Johann GroBschadl, "The Chinese Remainder Theorem and its Application in a High-speed RSA Crypto Chip", 1063-9527/00 2000 IEEE, ieeexplore.ieee.org.

117. Rajashekhar Modugu, Yong-Bin Kim and Minsu Choi, "Design and Performance Measurement of Efficient IDEA (International Data Encryption Algorithm) Crypto-Hardware using Novel Modular Arithmetic Components", 978-1-4244-2833-5/10/ 2010 IEEE, ieeexplore.ieee.org.
118. Bo Yang, Kaijie Wu, and Ramesh Karri, "Secure Scan: A Design-for-Test Architecture for Crypto Chips", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, VOL. 25, NO. 10, October 2006, ieeexplore.ieee.org.
119. Ms. Anuja R. Tungar, "Review paper: On, Comparative study of Embedded System architectures for implementation of ECC.", 978-1-4244-4570-7/09 2009 IEEE, ieeexplore.ieee.org.
120. Mohamed H. Abdel Rahman, E. Talkhan and Samir I. Shaheen, "Crypto-Algorithms maker Kit", ICM 2bb3, Dec. 9-11, Cairo, Egypt, ieeexplore.ieee.org, 2003.
121. William Stallings, "Cryptography and Network Security: Principle and Practice", Second Edition, Pearson Education Asia, Sixth Indian Reprint, 2002, ISBN: 81-7808-605-0.
122. Behrouz A. Forouzan, "Cryptography and Network Security", Special Indian Edition 2007, Tata Mc-Graw-Hill, ISBN-13: 978-0-07-066046-5, ISBN-10: 0-07-066046-8.
123. Ramesh Gaonkar, "Microprocessor Architecture, Programming and Application with the 8085", Fifth Edition, Penram International Publishing (India) Private Limited, ISBN: 81-87972-09-2.
124. Dr. Hafizur Rahaman, "Microprocessor Programming and Interfacing (Intel 8085 and 8086)", First Edition 2005, Everest Publishing House, ISBN: 81-7660-129-2.
125. Wayne Wolf, "FPGA-Based System Design", First Impression – 2009, Pearson Education, ISBN: 978-81-317-2465-1.

126. Zainalabedin Navabi, “Embedded Core Design with FPGA(s)”, Edition 2008, Tata Mc-Graw Hill, ISBN-13: 978-0-07-013978-7, ISBN-10: 0-07-013978-4.
127. J. Bhasker, “A VHDL Primer”, Thirteen Indian Reprint 2004, Pearson Education, ISBN: 81-7808-016-8.
128. Kanika Kaur, “Digital System Design”, Copyright © 2009, Scitech Publications (India) Pvt. Ltd., ISBN: 978-81-8371-188-3.
129. Yashavant P. Kanetkar, “Understanding Pointers in C”, First Indian Edition 2001, BPB Publications, ISBN: 81-7656-358-7.
130. “Dictionary of Computer and Information Technology Terms”, by LP Editorial Board, Law Point (Kolkata, India) Publishers, Distributers and Book Sellers, First Edition 2004.
131. Rajdeep Chakraborty and J.K. Mandal, “A Microprocessor-Based Block Cipher through Rotational Addition Technique (RAT)”, published & presented in 9th International Conference on Information Technology (ICIT 2006), held on 18-21 December 2006, at Bhubaneswar, India, organized and sponsored by IEEE Computer Society, IEEE, Orissa Information Technology Society (OITS), Institute of Technical Education and Research (ITER), New Jersey Institute of Technology (NJIT), Satyam Computers Ltd. and IEEE New jersey Section, and published by IEEE Computer Society Conference Publishing Services, ISBN-10: 0-7695-2635-7/06, ISBN-13: 978-0-7695-2635-5, pp 155–159.
132. Rajdeep Chakraborty and J.K. Mandal, “An Approach Towards Digital Content Protection Through Two Pass Replacement Technique (TPRT)”, published in First International Conference on Information Technology (INTL-INFOTECH 2007), held on March 19-21, 2007 at Haldia, INDIA, organized and sponsored by Department of Computer Science and Informatics, Haldia Institute of Technology (HIT), Technical Education Quality Improvement Program (TEQIP), Computer Society of India (CSI) (Kolkata), TEQIP network Partners, University of Calcutta (Kolkata), Govt. College of Engineering and Ceramic Technology (Kolkata), and published by Vitasta Publishing Pvt. Ltd., New Delhi, India, ISBN 81-89766-74-0, ISSN 0973-6824, pp 62–65.

133. Rajdeep Chakraborty and J.K. Mandal, “A Microprocessor-Based Stream Cipher Through Stream Addition Technique (SAT)”, published in International Conference on Systemics, Cybernetics and Informatics (ICSCI 2009), held on January 07-10, 2009 at Hyderabad, India, organized, sponsored and, published by Pentagram Research Center Pvt. Ltd. Hyderabad, India, Volume 1 of 2, pp 41–45.
134. Rajdeep Chakraborty and J.K. Mandal, “A Microprocessor-based Stream Cipher through Stream Parity Technique (SPT)”, published & presented in First International Conference on Computer, Communication, Control and Information Technology (C³IT 2009), held on 06-07 February, 2009 at Hoogly, West Bengal, INDIA, organized and sponsored by Academy of Technology (AOT), IEEE Leos Calcutta Chapter, IEEE EDS Calcutta Chapter, All India Council for Technical Education (AICTE), Indian Space Research Organization (ISRO), and CSIR, and published by MACMILLAN PUBLISHERS INDIA LTD Advanced Research Series, New Delhi, India, ISBN 10: 0230-63759-0, ISBN 13: 978-0230-63759-7, pp 417–423.
135. Rajdeep Chakraborty and J.K. Mandal, “Ensuring e-Security through Microprocessor-Based Recursive Transposition Technique (RTT)”, published & presented in 12th International Conference on Information Technology (ICIT 2009), held on December 21-24, 2009, at Bhubaneswar, India, organized and sponsored by IEEE, IEEE Computer Society, Orissa Information Technology Society (OITS), Temple City Institute of Technology and Engineering (TITE), Silicon Institute of Technology, IIIT, NIST, OCAC and Techno India Group (Kolkata), and published by Tata McGraw Hill Education Private Limited, New Delhi, India, ISBN-10: 0-07-068104-0, ISBN-13: 978-0-07-068104-2, pp 66–69.
136. Rajdeep Chakraborty and J. K. Mandal, “An RTL Based Design & Implementation of Block Cipher through Oriented Encryption Technique (OET)”, published & presented in International Conference on Computing and Systems (ICCS 2010), held on November 19-20, 2010, at Burdwan, West Bengal, India, organized and published by Department of Computer Science, The University of Burdwan, Burdwan – 713104, West Bengal, India, ISBN – 93-80813-01-5, pp 335 – 338.

137. Rajdeep Chakraborty and J. K. Mandal, “An RTL Based Design & Implementation of Block Cipher through Time-Stamp-Keyed-Oriented Encryption Technique (TSK-OET)”, published in International Journal of Advanced Research in Computer Science (IJARCS), ISSN 0976 – 5697, accepted & published in Volume 2 – No. 1 (Jan – Feb 2011) issue, pp-428 – 432, indexed by Index Copernicus, Directory of Open Access Journal (DAOJ), Open J Gate, Ulrichs Web, EBSCOhost, Electronic Journal Library, New Jour, ScienceCentral.com, Genamics, Mlibrary of University of Michigan, Kun Shan University Library and Dayang Journal System.
138. Rajdeep Chakraborty and J. K. Mandal, “FPGA Based Cipher Design & Implementation of Recursive Oriented Block Arithmetic and Substitution Technique (ROBAST)”, published in (IJACSA) International Journal of Advanced Computer Science and Applications, ISSN 2156-5570 (online), ISSN 2158-107X (print), accepted and published in Volume 2- Issue 4 (April 2011) issue pp-54 – 59, indexed by. docstoc, Scribd, getCITED, CiteSeer^x, EBSCO HOST, Directory of Open Access Journal (DAOJ), Google Scholar, Journal Seek, Index Copernicus, Georgetown University Library and Powered by Microsoft Research.
139. Rajdeep Chakraborty, Sananda Mitra and J. K. Mandal, “Shuffle-RAT: An FPGA-based Iterative Block Cipher”, published in International Journal of Advanced Research in Computer Science (IJARCS), ISSN 0976 – 5697, accepted & published in Volume 2 – No. 3 (May – June 2011) issue, pp-21 – 24, indexed by Index Copernicus, Directory of Open Access Journal (DAOJ), Open J Gate, Ulrichs Web, EBSCOhost, Electronic Journal Library, New Jour, ScienceCentral.com, Genamics, Mlibrary of University of Michigan, Kun Shan University Library and Dayang Journal System.

140. Rajdeep Chakraborty, Sonam Agarwal, Sridipta Misra, Vineet Khemka, Sunit Kr Agarwal and J. K. Mandal, “Triple SV: A Bit Level Symmetric Block-Cipher Having High Avalanche Effect”, published in (IJACSA) International Journal of Advanced Computer Science and Applications, ISSN 2156-5570 (online), ISSN 2158-107X (print), accepted and published in Volume 2- Issue 7 (July 2011) issue pp- 61 – 68, indexed by. docstoc, Scribd, getCITED, CiteSeer^x, EBSCO HOST, Directory of Open Access Journal (DAOJ), Google Scholar, Journal Seek, Index Copernicus, Georgetown University Library, Ulrichsweb, BASE, WorldCat and Powered by Microsoft Research.
141. Rajdeep Chakraborty, Sridipta Misra, Sunit Kumar Agarwal, Vineet Khemka, Sonam Agarwal and J. K. Mandal, “Efficient Hardware Realization of Triple Data Encryption Standard (TDES) Algorithm using Spartan-3E FPGA”, published & presented in International Conference on Issues and Challenges in Networking, Intelligence and Computing (ICNICT 2011), held on 2-3 September, 2011, at Ghaziabad, India, organized and sponsored by Department of Computer Science and Engineering, Krishna Institute of Engineering and Technology (KIET), India, Department of Science & Technology, Govt. of India, New Delhi, Computer Society of India (Ghaziabad Chapter), International Neural Network Society (India Regional Chapter), International Journal of Advanced Research in Computer Science and Ubiquitous Computing and Communication Journal, and published by Nandani Prakashan Pvt. Ltd., New Delhi, India, ISBN – 978-93-81126-27-1, pp 60 – 63.
142. Rajdeep Chakraborty, Debajyoti Guha and J. K. Mandal, “A Block Cipher Based Cryptosystem Through Forward Backward Overlapped Modulo Arithmetic Technique (FBOMAT)”, published in International Journal of Engineering & Science Journal (IJESR), ISSN 2277 – 2685, accepted & published in Volume 2 – Issue 5 (May 2012) issue, pp-349 – 360, indexed by Index Copernicus, Open J Gate, Ulrichs Web, Google Scholar.

143. Rajdeep Chakraborty, Avishek Datta and J. K. Mandal, “Modified Recursive Modulo 2^n and Key Rotation Technique (MRMKRT)”, published in International Journal of Engineering & Science Research (IJESR), ISSN 2277 – 2685, accepted & published in Volume 5 – Issue 2 (February 2015) issue, pp-76 – 81, indexed by Index Copernicus, Open J Gate, Ulrichs Web, Google Scholar.
144. Rajdeep Chakraborty, Santanu Basak and JK Mandal “An FPGA Based Crypto Processor through Triangular Modulo Arithmetic Technique (TMAT)”, published in International Journal of Multidisciplinary in Cryptology and Information Security (IJMCIS) ISSN 2320 –2610, accepted & published in Volume 3, No.3 (May – June 2014) issue, PP 14-20, indexed by Google scholar, Cite Seer, getCited, .docstoc, Scribd, Ulrich Web, Index Copernicus, Microsoft Academics, New Jour, DOAJ.
145. Debajyoti Guha, Rajdeep Chakraborty and Abhirup Sinha “A Block Cipher Based Cryptosystem Through Modified Forward Backward Overlapped Modulo Arithmetic Technique (MFBOMAT)”, published in IOSR Journal of Computer Engineering (IOSR–JCE), e-ISSN: 2278-0661, p-ISSN: 2278-8727, accepted & published in Volume 13, Issue 1 (Jul. - Aug. 2013) issue, PP 138-146, indexed by NASA, Cross Ref, Arxiv.org, Cabell’s, Index Copernicus, EBSCO Host, Ulrichs Web, Google Scholar, ANED, Jour Informatics.
146. Rajdeep Chakraborty, Sibendu Biswas and JK Mandal “Modified Rabin Cryptosystem through Advanced Key Distribution System”, published in IOSR Journal of Computer Engineering (IOSR–JCE), e-ISSN: 2278-0661, p-ISSN: 2278-8727, accepted & published in Volume 16, Issue 2 Ver XII (Mar. - Apr. 2014) issue, PP 01-07, indexed by NASA, Cross Ref, Arxiv.org, Cabell’s, Index Copernicus, EBSCO Host, Ulrichs Web, Google Scholar, ANED, Jour Informatics.

147. Rajdeep Chakraborty, Avishek Datta and JK Mandal “Secure Encryption Technique (SET): A Private Key Crypto System”, published in International Journal of Multidisciplinary in Cryptology and Information Security (IJMCIS) ISSN 2320 – 2610, accepted & published in Volume 4, No.1 (January – February 2015) issue, PP 10-13, indexed by Google scholar, Cite Seer, getCited, .docstoc, Scribd, Ulrich Web, Index Copernicus, Microsoft Academics, New Jour, DOAJ.
148. Sarkar, A, Mandal, J. K., “Secured wireless communication through Simulated Annealing Guided Triangularized Encryption by Multilayer Perceptron generated Session Key(SATMLP)”, CCSIT- 2013, Proceedings of Third International Conference on Computer Science & Information Technology(CCSIT-2013) vis Lecture Notes of the Institute for Computer Science, Social Informatics and Telecommunications Engineering(LNICST), Springer, ISSN No.1867-8211, Feb.18-20, 2013, pp. ...-...76, , Bangalore, India., 2013
149. Mandal, J. K., Chatterjee R, “Authentication of PCSs with Triangular Encryption Technique”, Proceedings of 6th Philippine Computing Science Congress(PCSC 2006), Ateneo de Manila University, Manila, Philippine, March 28-29,2006.
150. Avishek Datta, Rajdeep Chakraborty and J.K. Mandal, “The CRYPSTER: A Private Key Crypto System”, published & presented in 2015 IEEE International Conference on Computer Graphics, Vision and Information Security (IEEE CGVIS 2015) IEEE Conference Record number: #36759, held on November 02-03, 2015, at Bhubaneswar, India, organized and sponsored by IEEE Kolkata Section, KIIT University, Bhubaneswar, Odisha India and published in IEEE XPLORE CFP 15C89-ART ISBN: 978-1-4673-7437-8, CD-ROM CFP 15C89-CDR ISBN: 978-1-4673-7436-1, pp 35–37.