Towards Design and Implementation of Soft computing based Cryptographic Techniques for Wireless Communication

**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

Arindam Sarkar

Under the supervison of

Prof. Jyotsna Kumar Mandal

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

November, 2014

University of Kalyani
FACULTY OF ENGINEERING, TECHNOLOGY & MANAGEMENT

Prof. J. K. Mandal
Professor, Computer Science and
Engineering, University of Kalyani

Kalyani-741235, Nadia
West Bengal, India
Phone: (033) 25809617 (O)
Mobile: +91- 9434352214
e-mail: jkm.cse@gmail.com

Certificate

This is to certify that the thesis entitled "**TOWARDS DESIGN AND IMPLEMENTATION OF SOFT COMPUTING BASED CRYPTOGRAPHIC TECHNIQUES FOR WIRELESS COMMUNICATION**" submitted by Arindam Sarkar, who got his research proposal registered on 30.05.2012 (Ref. No. Ph.D/Regn./N.Rgl./Eg-34/Com.Sc./AS/2012) for the award of Ph.D. (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 Arindam Sarkar has fulfilled all the requirements according to rules of this University regarding the work embodied in this thesis.

(Dr. Jyotsna Kumar Mandal)
Professor
Department of Computer Science and Engineering
University of Kalyani, Kalyani

Date:
Place: Kalyani

Dedicated
To
My family
&
All my beloved
Teachers

ACKNOWLEDGEMENTS

This thesis is the end of my journey in obtaining my Ph.D. I have not traveled in a vacuum in this journey. This thesis has been kept on track and been seen through to completion with the support and encouragement of numerous people including my well wishers, my friends, colleagues and various institutions. At the end of my thesis I would like to thank all those who contributed in many ways to the success of this study and made it an unforgettable experience for me.

My first debt of gratitude must go to my supervisor and guide, Dr. Jyotsna Kumar Mandal, Professor, Department of Computer Science and Engineering, University of Kalyani, Kalyani, India. He patiently provided the vision, encouragement and advice necessary for me to proceed through the doctorial program and complete my dissertation. I am indebted to him for continuous inspiration, support and motivation and all kind of helps.

This gives me great pleasure to express my thanks to all teachers and staff members of the Department of Computer Science and Engineering, University of Kalyani, Department of Computer Science, The University of Burdwan and Department of Computer and System Sciences, VISVA-BHARATI, West Bengal, India, who helped me time to time in many ways during the period of my research work.

I take this opportunity to sincerely acknowledge the Department of Science & Technology (DST), Government of India, New Delhi, for providing financial assistance in the form of INSPIRE Fellowship which buttressed me to perform my work comfortably.

Last but not least, I would like to pay high regards to my parents Mr. Apurba Kumar Sarkar and Mrs. Kaberi Sarkar. I could not have completed this dissertation without bless and love of them. Besides this, several people have knowingly and unknowingly helped me in the successful completion of this thesis.

…………………………………

(Arindam Sarkar)

Department of Computer Science and Engineering

University of Kalyani, Kalyani, West Bengal, India.

ABSTRACT

The objectives of the proposed thesis is to enhance the security of the wireless communication system in such a way that the instead of exchanging the whole session key, soft computing based synchronization technique is used to construct a cryptographic key exchange protocol for generating the identical session key at sender and receiver. Here the partners benefit from mutual interaction, so that a passive attacker is usually unable to learn the generated key in time. This synchronized network can be used for message communication by encrypting the plaintext using any light weight encryption/decryption technique with the help of synchronized session key at both ends. Also grouped synchronization has been proposed to synchronize group of n party to form a synchronized grouped session key. The candidate searched some of such techniques which are simple and easy to understand, and also to trade-off between security and performance of light weight devices as well as energy awareness during the course of research.

The thesis considered synchronization of sender and receiver using soft computing tool for generating identical session key and light weight soft computing based encryption/decryption technique as an example corresponding to each technique. Here five such techniques based on soft computing based synchronization have been designed, implemented and tested through High Level Languages. These techniques have been discussed with their merits and demerits. Identical soft computing based network has been considered at sender and receiver. Both the communicating networks receive an indistinguishable input vector, produce an output bit and are trained based on the output bit. The dynamics of the two networks and their weight vector are found to a novel experience, where the demonstrate networks synchronize to an identical time dependent weight vector. This observable fact has been used to form a secured variable length secret session key using a public channel. Any light weight message encryption technique is used to encrypt the plaintext. In this thesis as an example light weight soft computing based message encryption technique is used to illustrate the cryptographic technique. Encrypted text get further encrypted using synchronized session key and transmitted to the receiver. During decryption receiver has the same synchronized session key which is used to perform first round of decryption operation and outcomes of this further decrypted by message decryption technique (exactly reverse process of message encryption) are performed and plaintext is regenerated. Comparison of all proposed techniques with each other and also with Tree Parity Machine (TPM) and Permutation Parity Machine (PPM), RSA, Triple-DES (168 bits), AES (128 bits),

RC4 and Vernam Cipher has been done with respect to the parameters like fifteen statistical tests of the NIST test suite, analysis of the average time (in cycle) needed for generating 128/192/256 bit session key by synchronization between two party and group of party, memory heap used during synchronization, relative time spent in GC and thread required in synchronization phase, encryption and decryption time, character frequencies, Avalanche and strict Avalanche effects, Bit Independence effects, Chi-Square values, character frequency test, entropy test, floating frequency test and autocorrelation test.

A model of session key generation through synchronization and encryption through cascaded implementation embodied with proposed algorithms has been introduced. The approach of cascaded implementation is an attempt to integrate the five proposed techniques. The proposed model may introduce new dimension to ensure security at maximum possible level. The model is very much suitable for the security of the system where unify computing is an essential component and it is idle to trade-off between security and performance of light weight devices having very low processing capabilities or limited computing power in wireless communication.

List of Publications

Book published

1. Sarkar, A., & Mandal, J. K. (2012). *Artificial Neural Network Guided Secured Communication Techniques: A Practical Approach.* LAP LAMBERT Academic Publishing, ISBN-10: 3659119911, ISBN-13: 978-3659119910.

Book Chapter

2. Sarkar, A., & Mandal, J. K. (2014). *Particle Swarm Optimization based Session Key Generation for Wireless Communication (PSOSKG).* In A. Bhattacharyya, S., & B. Dutta, P. (Eds.), *Handbook of Research on Swarm Intelligence in Engineering*, chapter 20, 701 E. Chocolate Ave., Hershey, Pennsylvania (USA): IGI GLOBAL. (Accepted)

International Journals

3. Sarkar, A., & Mandal, J. K. (2014). Cryptanalysis of Key Exchange method in Wireless Communication (CKE). *International Journal of Network Security (IJNS)*, ISSN 1816 – 3548 [Online]; 1816 – 353X [Print]. (Accepted)

4. Sarkar, A., & Mandal, J. K. (2014). Computational Science guided Soft Computing based Cryptographic Technique using Ant Colony Intelligence for Wireless Communication (ACICT). *International Journal of Computational Science and Applications (IJCSA), 4*(5), 61-73, DOI: 10.5121/ijcsa.2014.4505, ISSN 2200 – 0011.

5. Sarkar, A., & Mandal, J. K. (2014). Intelligent Soft Computing based Cryptographic Technique using Chaos Synchronization for Wireless Communication (CSCT). *International Journal of Ambient Systems and Applications (IJASA), 2*(3), 11-20, DOI: 10.5121/ijasa.2014.2302, ISSN 2320 – 9259 [Online]; 2321 – 6344 [Print].

6. Sarkar, A., & Mandal, J. K. (2014). Soft Computing based Cryptographic Technique using Kohonen's Self-Organizing Map Synchronization for Wireless communication (KSOMSCT). *International Journal in Foundations of Computer Science & Technology (IJFCST), 4*(5), 85-100, DOI: 10.5121/ijfcst.2014.4508, ISSN 1839 – 7662.

7. Sarkar, A., & Mandal, J. K. (2014). Secured Transmission Through Multi LayerPerceptron in Wireless Communication (STMLP). *International Journal of Mobile Network Communications & Telematics (IJMNCT), 4*(4), 1-16, DOI: 10.5121/ijmnct.2014.4401, ISSN 1839 – 5678.

8. Sarkar, A., & Mandal, J. K. (2014). Cryptanalysis of Key Exchange method using Computational Intelligence guided Multilayer Perceptron in Wireless Communication (CKEMLP). *Advanced Computational Intelligence: An International Journal (ACII), 1*(1), 1-9, ISSN 2317 – 4113.

9. Sarkar, A., & Mandal, J. K. (2013). Neuro Genetic Key Based Recursive Modulo-2 Substitution Using Mutated Character for Online Wireless Communication (NGKRMSMC). *International Journal of Computational Science and Information Technology (IJCSITY), 1*(4), 49-59, DOI: 10.5121/ijcsity.2014.1404, ISSN 2320 - 7442 [Online]; 2320 - 8457 [Print].

10. Sarkar, A., & Mandal, J. K. (2013). Genetic Algorithm Guided Key Generation in Wireless Communication (GAKG). *International Journal on Cybernetics & Informatics (IJCI), 2*(5), 9-17, DOI: 10.5121/ijci.2013.2502, ISSN 2277 - 548X [Online]; 2320 - 8430 [Print].

11. Sarkar, A., & Mandal, J. K. (2013). Computational Intelligence Based Simulated Annealing Guided Key Generation In Wireless Communication (CISAKG). *International Journal on Information Theory (IJIT), 2*(4), 35-44, DOI: 10.5121/ijit.2014.2403, ISSN 2319 - 7609 [Online]; 2320 - 8465 [Print].

12. Sarkar, A., & Mandal, J. K. (2013). Group Session Key exchange Multilayer Perceptron based Simulated Annealing guided Automata and Comparison based Metamorphosed Encryption in Wireless Communication (GSMLPSA). *International Journal of Wireless & Mobile Networks (IJWMN), 5*(4), 203-222, DOI: 10.5121/ijwmn.2013.5415, ISSN 0975-3834 [Online]; 0975-4679 [Print].

13. Sarkar, A., & Mandal, J. K. (2013). Key Swap Over among Group of Multilayer Perceptrons for Encryption in Wireless Communication (KSOGMLPE), *International Journal of Information Technology, Control and Automation (IJITCA), 3(1),* 85-100, DOI:10.5121/ijitca.2013.3107, ISSN 1839 – 6282.

14. Sarkar, A., & Mandal, J. K. (2012). Secured Wireless Communication using Fuzzy Logic based High Speed Public-Key Cryptography (FLHSPKC), *International Journal of Advanced Computer Science and Applications (IJACSA), 3*(10), 137-145, U.S ISSN : 2156-5570 [Online], U.S ISSN : 2158-107X(Print), 2012 Impact Factor : 1.324.

15. Sarkar, A., & Mandal, J. K. (2012). Key Generation and Certification using Multilayer Perceptron in Wireless Communication (KGCMLP), *International Journal of Security, Privacy and Trust Management (IJSPTM), 1*(5), 27-43, DOI: 10.5121/ijsptm.2012.1503, ISSN 2277 - 5498 [Online]; 2319 - 4103 [Print].

16. Sarkar, A., & Mandal, J. K. (2012). Evolutionary Computation Guided Energy Efficient Key Organization in Wireless Communication (ECEEKO), *International Journal of Information and Network Security (IJINS)*, *2*(1), 352-366, ISSN: 2089-3299.

17. Sarkar, A., & Mandal, J. K. (2012). Energy Efficient Wireless Communication Using Genetic Algorithm Guided Faster Light Weight Digital Signature Algorithm (GADSA), *International Journal of Advanced Smart Sensor Network Systems (IJASSN)*, *2*(3), 9-25, DOI: 10.5121/ijassn.2012.2302, ISSN: 2231 - 4482 [Online]; 2231 - 5225 [Print].

18. Sarkar, A., & Mandal, J. K. (2012). Multilayer Perceptron Guided Key Generation Through Mutation With Recursive Replacement In Wireless Communication (MLPKG), *International Journal on AdHoc Networking Systems (IJANS)*, *2*(3), 11-28, DOI: 10.5121/ijans.2012.2302, ISSN: 2249 - 0175 [Online]; 2249 - 2682 [Print].

19. Sarkar, A., & Mandal, J. K. (2012). Swarm Intelligence based Faster Public-Key Cryptography in Wireless Communication (SIFPKC), *International Journal of Computer Science & Engineering Technology (IJCSET)*, *3*(7), 267-273, ISSN: 2229-3345.

20. Sarkar, A., & Mandal, J. K. (2012). Secured Wireless Communication by High-Speed RSA Using Evolutionary Programming based Optimized Computation (HS-RSA-EP), International *Journal of Advanced Research in Computer Science (IJARCS)*, *3*(4), 161-165, ISSN 0976 – 5697.

21. Sarkar, A., & Mandal, J. K. (2012). Object Oriented Modelling of Idea Using GA Based Efficient Key Generation For E-Governance Security (OOMIG), *International Journal of Distributed and Parallel Systems (IJDPS)*, *3*(2), 171-183, DOI: 10.5121/ijdps.2012.3215, ISSN : 0976 - 9757 [Online] ; 2229 - 3957 [Print].

International Conference

22. Sarkar, A., & Mandal, J. K. (2014). Analysis of Tree parity Machine and Double Hidden Layer Perceptron based Session Key Exchange in Wireless Communication. *Annual Convention and International Conference on Emerging ICT for Bridging Future*, CSI Hyderabad Chapter in Association with JNTU Hyderabad & DRDO, December 12-14 2014, Hyderabad, India: AISC Series Springer. (Accepted)

23. Sarkar, A., Mandal, J. K., & Mondal, P. (2014). Neuro-Key Generation Based on HEBB Network for Wireless Communication. In *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and applications (FICTA 2014),* AISC Series Springer Vol. 328, Book Subtitle Vol.2, pp. 197-205, DOI: 10.1007/978-3-319-12012-6_22, ISBN: 978-3-319-12011-9 [Print], ISBN: 978-3-319-12012-6 [Online], Series ISSN: 2194-5357, November 14-15 2014, Bhubaneswar, Orissa, India, Springer International Publishing.

24. Mandal, J. K., Dutta, D., & Sarkar, A. (2014). Hopfield network based neural key generation for wireless communication. In *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and applications (FICTA 2014),* AISC Series Springer Vol. 328, Book Subtitle Vol.2, pp. 217-224, DOI: 10.1007/978-3-319-12012-6_24, ISBN: 978-3-319-12011-9 [Print], ISBN: 978-3-319-12012-6 [Online], Series ISSN: 2194-5357, November 14-15 2014, Bhubaneswar, Orissa, India, Springer International Publishing.

25. Sengupta, M., Mandal, J. K., Sarkar, A., & Bhattacharyya, T. (2014). KSOFM Network Based Neural Key Generation for Wireless Communication. In *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and applications (FICTA 2014),* AISC Series Springer Vol. 328, Book Subtitle Vol.2, pp. 207-215, DOI: 10.1007/978-3-319-12012-6_23, ISBN: 978-3-319-12011-9 [Print], ISBN: 978-3-319-12012-6 [Online], Series ISSN: 2194-5357, November 14-15 2014, Bhubaneswar, Orissa, India, Springer International Publishing.

26. Sarkar, A., & Mandal, J. K. (2013). Complete Binary Tree Architecture based Triple Layer Perceptron Synchronized Group Session Key Exchange and Authentication in Wireless Communication (CBTLP). In *Proceedings of the 48th Annual Convention of CSI on theme "ICT and Critical Infrastructure*, AISC Series Springer Vol. 249, Book Subtitle Vol.2, pp. 609-615, DOI: 10.1007/978-3-319-03095-1_66, ISBN: 978-3-319-03094-4 [Print], ISBN: 978-3-319-03095-1 [Online], Series ISSN: 2194-5357, December 13-15 2013, Computer Society of India, Visakhapatnam Chapter, Visakhapatnam, India, Springer International Publishing.

27. Sarkar, A., Mandal, J. K., & Patra P. (2013). Double Layer Perceptron Synchronized Computational Intelligence guided Fractal Triangle based Cryptographic Technique for Secured Communication (DLPFT). In *Proceedings of the IEEE 4th International Symposium on Electronic System Design (ISED-2013),* pp. 191-195, ISBN 978-0-7695-5143-2, December 13-15 2013, NTU, Singapore, IEEE.

28. Sarkar, A., & Mandal, J. K. (2013). Computational Intelligence based Triple Layer Perceptron Model Coordinated PSO guided Metamorphosed based Application in Cryptographic Technique for Secured Communication (TLPPSO). In *Proceedings of the First International Conference on Computational Intelligence: Modeling, Techniques and Applications (CIMTA-2013),* Vol.10, pp. 433-442, DOI: 10.1016/j.protcy.2013.12.380, ISSN: 2212-0173, September 27-28 2013, Department of Computer Science & Engineering, University of Kalyani, Kalyani, India, Procedia Technology, Elsevier.

29. Mandal, J. K., Chowdhuri, A., & Sarkar, A. (2013). Time Efficient Optimal Tuning through various Learning Rules in Unify Computing (TEOTLRUC). In *Proceedings of the First International Conference on Computational Intelligence: Modeling, Techniques and Applications (CIMTA-2013),* Vol.10, pp. 474-481, DOI: 10.1016/j.protcy.2013.12.385, ISSN: 2212-0173, September 27-28 2013, Department of Computer Science & Engineering, University of Kalyani, Kalyani, India, Procedia Technology, Elsevier.

30. Mandal, J. K., Chowdhuri, A., & Sarkar, A. (2013). Encryption Technique based on Neural Session Key (ETNSK). In *Proceedings of the Second International Conference on Computing And Systems, (ICCS 2013)*, Department of Computer Science, The University of Burdwan, September 21-22 2013, pp. 29-33, ISBN 978-9-35-134273-1, Burdwan, India: McGraw Hill Education Private Limited.

31. Sarkar, A., & Mandal, J. K. (2013). Secured Wireless Communication Through Simulated Annealing Guided Triangularized Encryption By Multilayer Perceptron Generated Session Key (SATMLP). In *Proceedings of the Third International Conference on Computer Science & Information Technology (CCSIT 2013)*, Computer Science & Information Technology (CS & IT), Bangalore, February 18-20 2013, pp. 217-224, DOI: 10.5121/csit.2013.3624, 2013, ISBN: 978-1-921987-00-7, India: AIRCC.

32. Mandal, J. K., & Sarkar, A. (2011). An Adaptive Genetic Key Based Neural Encryption For Online Wireless Communication (AGKNE). In *Proceedings of the IEEE International Conference on Recent Trends In Information System (RETIS 2011),* Jadavpur University, December 21-23 2011, pp. 62-67, ISBN 978-1-4577-0791-9. Kolkata, India: IEEE.

33. Mandal, J. K., & Sarkar, A. (2011). An Adaptive Neural Network Guided Secret Key based Encryption through Recursive Positional Modulo-2 Substitution for Online Wireless Communication (ANNRPMS). In *Proceedings of the IEEE International Conference on Recent Trends In Information Technology (ICRTIT 2011)*, Madras Institute of Technology, Anna University, Chennai, June 3-5 2011, pp.107-112, ISBN 978-1-4577-0590-8/11, Tamil Nadu, India: IEEE.

34. Mandal, J. K., & Sarkar, A. (2011). An Adaptive Neural Network Guided Random Block Length Based Cryptosystem (ANNRBLC). In *Proceedings of the IEEE International Conference on 2nd International Conference on Wireless Communications, Vehicular Technology, Information Theory And Aerospace & Electronic System Technology (WIRELESS VITAE 2011)*, Special Session: Security Protection Mechanism in Wireless Sensor Networks, Chennai, February 28-March 03 2011, pp. 1-5, ISBN 978-87-92329-61-5, Tamil Nadu, India: IEEE.

35. Mandal, J. K., & Sarkar, A. (2010). Neural Network Guided Secret Key based Encryption through Cascading Chaining of Recursive Positional Substitution of Prime Non-Prime (NNSKECC). In *Proceedings of the First International Conference on Computing And Systems (ICCS 2010)*, Department of Computer Science, The University of Burdwan, November 19-20 2010, pp. 291-297, ISBN 93-80813-01-5, Burdwan, India.

National Conference

36. Mandal, J. K., & Sarkar, A. (2012). Neural Session Key based Traingularized Encryption for Online Wireless Communication (NSKTE), *In Proceedings of the 2nd National Conference on Computing and Systems, (NaCCS 2012)*, Department of Computer Science, The University of Burdwan, Burdwan, India, March 15-16 2012, pp. 172-177, ISBN 978-93-808131-8-9.

37. Mandal, J. K., & Sarkar, A. (2012). Neural Weight Session Key based Encryption for Online Wireless Communication (NWSKE), *In Proceedings of the Research and Higher Education in Computer Science and Information Technology, (RHECSIT- 2012)*, Department of Computer Science, Sammilani Mahavidyalaya, Kolkata, India, February 21-22 2012, pp. 90-95, ISBN 978-81-923820-0-5.

Zonal Seminar

38. Sarkar, A. (2013). Parallel Session Key Exchange and Certification by Fine Tuning of Double Layer Perceptron in Wireless communication (PKECDLP), *In Proceedings of the ICT in Present Wireless Revolution: Challenges and Issues*, The Institution of Electronics and Telecommunication Engineers Kolkata Centre, IETE Kolkata Center, Salt Lake, India, August 30-31 2013, pp. 1-9, ISBN 978-93-5126-699-0.

Publication Indexing Database

The lists of publications are indexed/abstracted in the following databases which are mentioned in the following table serially.

Table
Indexing database of publications

Publication Serial No.	Database
1	*Amazon, Slideshare, docstoc,* takealot, ebay, zopper, infibeam, bol, bokus, gettextbooks, sears, pricecheck, *Google Scholar* etc.
2	*SCOPUS, Thomson Reuters, DBLP Computer Science Bibliography, ERIC - Education Resources Information Center, and ACM Digital Library, CrossRef., Compendex, PsycINFO, INSPEC, Cabell's Directories, Google Scholar etc.*
3	*SCOPUS, DBLP, SciVerse, Engineering Village, Ei Compendex, Summon by Serial Solutions, SCImago, EBSCO, DOAJ, Google Scholar etc.*
4	*Ulrichsweb, DOAJ, Scrib, getCITED, Pubget, .docstoc, pub zone, Open J-Gate, CiteSeerx, Google Scholar, cnki.net, etc.*
5	*EBSCO, Google Scholar, pub zone, CSEB, Scribd, Ulrichsweb, getCITED, Pubget, .docstoc, ProQuest, DOAJ, CiteSeerx, cnki.net, etc.*
6	*Ulrichsweb, , Scrib, getCITED, Pubget, ProQuest, .docstoc, pub zone, CiteSeerx, Google Scholar, EBOSCO, cnki.net, WorldCat, CSEB, etc.*
7	*EBSCO, Google Scholar, pub zone, CSEB, Scribd, Ulrichsweb, getCITED, .docstoc, cnki.net, etc.*
8	*EBSCO, Google Scholar, pub zone, CSEB, Scribd, Ulrichsweb, getCITED, Pubget, .docstoc, ProQuest, etc.*
9	*EBSCO, Google Scholar, pub zone, CSEB, Scribd, Ulrichsweb, getCITED, Pubget, .docstoc, ProQuest, etc.*
10	*Ulrichsweb, DOAJ, Scribd, getCITED, Pubget, .docstoc, pub zone, Open J-Gate, CiteSeerx, Google Scholar, etc.*
11	*EBSCO, Google Scholar, CSEB, Scribd, DOAJ, getCITED, Pubget, CiteSeerx, .docstoc, pub zone, Ulrichsweb, WorldCat, ProQuest, etc.*
12	*The Elektronische Zeitschriftenbibliothek EZB, Genamics JournalSeek, Inspec, Google Scholar, SCIRUS, EBSCO, DOAJ, Scribd, Ulrichsweb, getCITED, Pubget, CiteSeerx, .docstoc, pub zone, PKP (Public Knowledge Project), WorldCat, ProQuest, NASA, etc.*
13	*Google Scholar, ProQuest, EBSCO, .docstoc, getCITED, pub zone, Scribd, CSEB, Ulrichsweb, Pubget, etc.*
14	*DOAJ, Index Copernicus (IC), Google Scholar, Microsoft Academic Search, GetCITED, CiteSeerx, SCIRUS, EBSCOhost, WorldCat, BASE, Ulrichsweb™, Open J-Gate, Cabell's Directory, University Citations: Harvard Library, The University of Melbourne, University of Liverpool, Cornell University, Hochschule RheinMain University of Applied Sciences Wiesbaden Rüsselsheim Geisenheim, Technische Universität Berlin, Universität Hamburg, Queen's University, Technische Universität Darmstadt, Unikassel Versitat, Philipps Universität Marburg, Akademie der Wissenschaften und der Literatur Mainz, Universität Frankfurt Am Main, Biblioteca, Universität Regensburg, Staats- und Universitätsbibliothek Bremen, Technische Hochschule Mittelhssen, etc.*
15	*Ulrichsweb, EBSCO, Google Scholar, CSEB, Scribd, DOAJ, etc.*
16	*SCOPUS, BASE (Bielefeld Academic Search Engine), Cabell's Directory, Cite Seerx , Computer Science Directory, DOAJ, EBSCO Publishing, EI, Electronic Journals Library, ELSEVIER, Google Scholar, Index Copernicus, ISSUU, NewJour,| OJS PKP, Open J-Gate, ProQuest, Science Central , Scirus , , Socolar Open Access, Ulrich's Periodicals Directory, World Wide Science , WorldCat, etc.*
17	*Google Scholar, getCITED, pub zone, .docstoc, Scribd, CSEB, EBSCO, ProQuest, etc.*

Publication Serial No.	Database
18	*Google Scholar, ProQuest, EBSCO, .docstoc, getCITED, pub zone, Scribd, CSEB, etc.*
19	*DOAJ, INDEX COPERNICUS, Google Scholar, Open J-Gate, Cornell University Library, SCIRUS, etc.*
20	*EBOSCO, Genamics, M Library, TU Berlin, Kun Shan University Library, INDEX COPERNICUS, DOAJ, Electronic Journals Library, New Jour, sciencecentral.com, ulrichsweb, Dayang Journal System, Google Scholar etc.*
21	*EBSCO, DOAJ, NASA, Google Scholar, INSPEC and WorldCat, 2011, etc.*
22	*ISI Proceedings, DBLP., SCOPUS, Zentralblatt Math, MetaPress, Springerlink, Google Scholar, etc.*
23	*SCOPUS, ISI Proceedings, DBLP. Ulrich's, EI-Compendex, , Zentralblatt Math, MetaPress, Springerlink, Google Scholar etc.*
24	*SCOPUS, ISI Proceedings, DBLP. Ulrich's, EI-Compendex, , Zentralblatt Math, MetaPress, Springerlink, Google Scholar etc.*
25	*SCOPUS, ISI Proceedings, DBLP. Ulrich's, EI-Compendex, , Zentralblatt Math, MetaPress, Springerlink, Google Scholar etc.*
26	*SCOPUS, DBLP, INSPEC, Google Scholar, ACM, Digital Library, Gale, SCImago etc.,*
27	*SCOPUS,DBLP, Google Scholar, ACM, Digital Library, Gale, SCImago etc.,*
28	*SCOPUS, INSPEC, Google Scholar, Gale, SCImago, ACM, Digital Library etc.,*
29	*SCOPUS, ACM, Digital Library, INSPEC, Google Scholar, Gale, SCImago etc.,*
30	*DBLP, Google Scholar etc.,*
31	*SCOPUS, DBLP, ACM, Digital Library, INSPE , SCImago, Google Scholar etc.,*
32	*SCOPUS, DBLP, Google Scholar, Digital Library, INSPEC, Gale, SCImago etc.,*
33	*SCOPUS, DBLP, INSPEC, , Digital Library, Google Scholar, SCImago etc.,*
34	*SCOPUS, DBLP, Google Scholar INSPEC, ACM,, Digital Library, Gale, SCImago etc.,*
35	*DBLP, Google Scholar etc.,*
36	*DBLP, Google Scholar etc.,*
37	*DBLP, Google Scholar etc.,*
38	*DBLP, Google Scholar etc.,*

List of Papers Presented

International Conference

1. Sarkar, A., Mandal, J. K., & Mondal, P. (2014). Neuro-Key Generation Based on HEBB Network for Wireless Communication. In *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and applications (FICTA 2014),* AISC Series Springer Vol. 328, Book Subtitle Vol.2, pp. 197-205, DOI: 10.1007/978-3-319-12012-6_22, ISBN: 978-3-319-12011-9 [Print], ISBN: 978-3-319-12012-6 [Online], Series ISSN: 2194-5357, November 14-15 2014, Bhubaneswar, Orissa, India, Springer International Publishing.

2. Sarkar, A., & Mandal, J. K. (2013). Computational Intelligence based Triple Layer Perceptron Model Coordinated PSO guided Metamorphosed based Application in Cryptographic Technique for Secured Communication (TLPPSO). In *Proceedings of the First International Conference on Computational Intelligence: Modeling, Techniques and Applications (CIMTA-2013),* Vol.10, pp. 433-442, DOI: 10.1016/j.protcy.2013.12.380, ISSN: 2212-0173, September 27-28 2013, Department of Computer Science & Engineering, University of Kalyani, Kalyani, India, Procedia Technology, Elsevier.

3. Mandal, J. K., Chowdhuri, A., & Sarkar, A. (2013). Time Efficient Optimal Tuning through various Learning Rules in Unify Computing (TEOTLRUC). In *Proceedings of the First International Conference on Computational Intelligence: Modeling, Techniques and Applications (CIMTA-2013)*, Vol.10, pp. 474-481, DOI: 10.1016/j.protcy.2013.12.385, ISSN: 2212-0173, September 27-28 2013, Department of Computer Science & Engineering, University of Kalyani, Kalyani, India, Procedia Technology, Elsevier.

4. Mandal, J. K., Chowdhuri, A., & Sarkar, A. (2013). Encryption Technique based on Neural Session Key (ETNSK). In *Proceedings of the Second International Conference on Computing And Systems, (ICCS 2013)*, Department of Computer Science, The University of Burdwan, September 21-22 2013, pp. 29-33, ISBN 978-9-35-134273-1, Burdwan, India: McGraw Hill Education Private Limited.

5. Sarkar, A., & Mandal, J. K. (2013). Secured Wireless Communication Through Simulated Annealing Guided Triangularized Encryption By Multilayer Perceptron Generated Session Key (SATMLP). In *Proceedings of the Third International Conference on Computer Science & Information Technology (CCSIT 2013),* Computer Science & Information Technology (CS & IT), Bangalore, February 18-20 2013, pp. 217-224, DOI: 10.5121/csit.2013.3624, 2013, ISBN: 978-1-921987-00-7, India: AIRCC.

6. Mandal, J. K., & Sarkar, A. (2011). An Adaptive Genetic Key Based Neural Encryption For Online Wireless Communication (AGKNE). In *Proceedings of the IEEE International Conference on Recent Trends In Information System (RETIS 2011)*, Jadavpur University, December 21-23 2011, pp. 62-67, ISBN 978-1-4577-0791-9. Kolkata, India: IEEE.

7. Mandal, J. K., & Sarkar, A. (2011). An Adaptive Neural Network Guided Secret Key based Encryption through Recursive Positional Modulo-2 Substitution for Online Wireless Communication (ANNRPMS). In *Proceedings of the IEEE International Conference on Recent Trends In Information Technology (ICRTIT 2011)*, Madras Institute of Technology, Anna University, Chennai, June 3-5 2011, pp. 107-112, ISBN 978-1-4577-0590-8/11, Tamil Nadu, India: IEEE.

8. Mandal, J. K., & Sarkar, A. (2011). An Adaptive Neural Network Guided Random Block Length Based Cryptosystem (ANNRBLC). In *Proceedings of the IEEE International Conference on 2nd International Conference on Wireless Communications, Vehicular Technology, Information Theory And Aerospace & Electronic System Technology (WIRELESS VITAE 2011)*, Special Session: Security Protection Mechanism in Wireless Sensor Networks, Chennai, February 28-March 03 2011, pp. 1-5, ISBN 978-87-92329-61-5, Tamil Nadu, India: IEEE.

9. Mandal, J. K., & Sarkar, A. (2010). Neural Network Guided Secret Key based Encryption through Cascading Chaining of Recursive Positional Substitution of Prime Non-Prime (NNSKECC). In *Proceedings of the First International Conference on Computing And Systems (ICCS 2010)*, Department of Computer Science, The University of Burdwan, November 19-20 2010, pp. 291-297, ISBN 93-80813-01-5, Burdwan, India.

National Conference

10. Mandal, J. K., & Sarkar, A. (2012). Neural Session Key based Traingularized Encryption for Online Wireless Communication (NSKTE), In *Proceedings of the 2nd National Conference on Computing and Systems, (NaCCS 2012)*, Department of Computer Science, The University of Burdwan, Burdwan, India, March 15-16 2012, pp. 172-177, ISBN 978-93-808131-8-9.

11. Mandal, J. K., & Sarkar, A. (2012). Neural Weight Session Key based Encryption for Online Wireless Communication (NWSKE), In *Proceedings of the Research and Higher Education in Computer Science and Information Technology, (RHECSIT- 2012)*, Department of Computer Science, Sammilani Mahavidyalaya, Kolkata, India, February 21-22 2012, pp. 90-95, ISBN 978-81-923820-0-5.

Zonal Seminar

12. Sarkar, A. (2013). Parallel Session Key Exchange and Certification by Fine Tuning of Double Layer Perceptron in Wireless communication (PKECDLP), In *Proceedings of the ICT in Present Wireless Revolution: Challenges and Issues*, The Institution of Electronics and Telecommunication Engineers Kolkata Centre, IETE Kolkata Center, Salt Lake, India, August 30-31 2013, pp. 1-9, ISBN 978-93-5126-699-0.

Contents

List of Abbreviations

KSOFM	: Kohonen’s Self Organizing Feature Map
DHLP	: Double Hidden Layer Perceptron
CDHLP	: Chaos based Double Hidden Layer Perceptron
CTHLP	: Chaos based Triple Hidden Layer Perceptron
CGTHLP	: Chaos based Group Triple Hidden Layer Perceptron
SA	: Simulated Annealing
GA	: Genetic Algorithm
ACI	: Ant Colony Intelligence
PSI	: Particle Swarm Intelligence
TPM	: Tree Parity Machine
PPM	: Permutation Parity Machine
MITM	: Man-In-The-Middle
AES	: Advanced Encryption standard
TDES	: Triple Data Encryption standard
RSA	: Rivest Shamir Adleman
RC4	: Rivest Cipher 4
RC5	: Rivest Cipher 5

List of Symbols

$\sum$	: Summation
σ	: Sigma (Hidden Layer Output)
τ	: Perceptron Final Output
$\oplus$	: *Exclusive-OR*
&	: *Bitwise AND*
$\neq$	: Not equal to
$\times$	: Multiplication
$\div$	: Division
$<$	: Less than
$\leq$	: Less than Equal to
$>$	: Grater than
$\geq$	: Grater than equal to
$\cong$	: Equivalent to
$\forall$	: For All
$\in$	: Belongs to
!	: Factorial

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introductory Discussions

Cryptography is the practice and study of techniques for secure communication in the existence of third parties (called adversaries).[1][2] More usually, it is about constructing and analyzing protocols that overcome the influence of adversaries[3][4] and which are associated to a variety of aspects in information security such as data confidentiality, data integrity, authentication, and non-repudiation.[5] Section 1.2 discussed about essence of cryptography that of some other available cryptographic techniques have been presented in section 1.3 of this chapter. Section 1.4 discussed about the soft computing based cryptography. A comprehensive survey of literature has been presented in section 1.5. Learning rules for tuning of perceptrons discussed in section 1.6. Metrics for evaluation of proposed algorithms have been given in section 1.7. Objectives of the study are given in section 1.8. Organization of the thesis is given in section 1.9. Some salient features of the thesis are described in section 1.10.

1.2 Essence of Cryptography

Modern cryptography intersects the disciplines of mathematics, computer science, and electrical engineering. Applications of cryptography include ATM cards, computer passwords, and electronic commerce. Modern cryptography follows a strongly scientific approach, and designs cryptographic algorithms around computational hardness assumptions, making such algorithms hard to break by an adversary. It is theoretically possible to break such a system but infeasible to do so by any practical means. These schemes are therefore computationally secure. There exist secure schemes that provably cannot be broken, an example is the one-time pad, but these schemes are more difficult to implement than the theoretically breakable but computationally secure mechanisms.[6]

Until modern times cryptography almost exclusively as encryption, which is the process of converting ordinary information (called plaintext) into unintelligible gibberish (cipher text).[7] Decryption is the reverse, in other words, moving from the unintelligible cipher text back to plaintext. A cipher (or cypher) is a pair of algorithms that create the encryption and the reversing decryption. The detailed operation of a cipher is controlled both by the algorithm and in each instance by a key. This is a secret parameter (ideally known only to the communicants) for a specific message exchange context. A "cryptosystem" is the ordered list of elements of finite possible plaintexts, finite possible cipher texts, finite possible keys, and the encryption and decryption algorithms which correspond to each key. Keys are important, as ciphers without variable keys can be trivially broken with only the knowledge of the cipher used and are therefore useless (or even counter-productive) for most purposes. Historically, ciphers were often used directly for encryption or decryption without additional procedures such as authentication or integrity checks.[6] To provide the essence of cryptographic techniques section 1.2.1 discussed about the basic idea of encryption/decryption. Concept of cipher is discussed in the section 1.2.2. Section 1.2.3 discussed about the cryptographic key. The concept of key management, key generation, key agreement and key exchange are discussed in section 1.2.4, 1.2.5, 1.2.6, 1.2.7. Finally attack models are presented in section 1.2.8.

1.2.1 Encryption/Decryption

In cryptography, encryption is the process of encoding messages or information in such a way that only authorized parties can read it.[9] Encryption does not of itself prevent interception, but denies the message content to the interceptor.[10] In an encryption scheme, the message or information, referred to as plaintext, is encrypted using an encryption algorithm, generating cipher text that can only be read if decrypted.[10] Symmetric key cryptography involves the usage of the same key for the encryption and decryption. This scheme is suffering from key distribution or key exchange. Since the sender and the receiver will use the same key to lock and unlock, this is called symmetric key operation. Thus the key distribution problem is inherently linked with the symmetric key operation. Also number of keys required as compared to the number of participants in the message exchange is equal about the square of the number of participants, so scalability is an issue. Whereas, asymmetric key cryptography involves the usage of one key for encryption and another, different key for decryption. No other key can decrypt the message-not even the original (i.e. first) key used for encryption. The beauty of this scheme is that every communicating party needs just a key pair for communicating with any number of other communicating parties. One of the two keys is called as public key and other is the private key. In case of asymmetric key cryptography speed of encryption and decryption is very slow and size of resulting encrypted text is more than the original plaintext size. [11]

1.2.2 Cipher

In cryptography, a cipher (or cypher) is an algorithm for performing encryption or decryption, a series of well-defined steps that can be followed as a procedure. An alternative, less common term is *encipherment*. To encipher or encode is to convert information from plaintext into cipher or code. In non-technical usage, a 'cipher' is the same thing as a 'code'; however, the concepts are distinct in cryptography. In classical cryptography, ciphers were distinguished from codes. Most modern ciphers can be categorized in several ways

- By whether they work on blocks of symbols usually of a fixed size (block ciphers), or on a continuous stream of symbols (stream ciphers).
- By whether the same key is used for both encryption and decryption (symmetric key algorithms), or if a different key is used for each (asymmetric key algorithms). If the algorithm is symmetric, the key must be known to the recipient and sender and to no one else. If the algorithm is an asymmetric one, the enciphering key is different from, but closely related to, the deciphering key. If one key cannot be deduced from the other, the asymmetric key algorithm has the public/private key property and one of the keys may be made public without loss of confidentiality. [12]
- Block ciphers or stream ciphers are the two ways in which symmetric ciphers are implemented. A block cipher enciphers input in blocks of plaintext whereas individual characters are the form of input by a stream cipher.
- Block cipher like the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES) have been designated cryptography standards by the US government (though later DES was withdrawn and replaced by AES).[11[13] Despite not being an official standard anymore, DES (especially its still approved and much more secure variety, triple-DES) still holds a firm position. Its application is of wider range, from ATM encryption to e-mail privacy and secure remote access. There are many ciphers that

have been designed and released with variation in quality whereas many have been thoroughly broken like FEAL. [14][15]

- Unlike block cipher, stream cipher creates an arbitrarily long stream of key which is combined bit-by-bit or character-by-character (similar to one-time pad). The output stream, in a stream cipher is created based on hidden internal state which changes as the cipher operates. The secret key is used to set up the internal state. Block ciphers can be used as stream ciphers where RC4 is a widely used stream cipher.[15]

1.2.3 Cryptographic Key

In cryptography, a key is a piece of information (a parameter) that determines the functional output of a cryptographic algorithm or cipher. Without a key, the algorithm would produce no useful result. In encryption, a key specifies the particular transformation of plaintext into cipher text, or vice versa during decryption. Encryption algorithms which use the same key for both encryption and decryption are known as symmetric key algorithms. Asymmetric key algorithms use a pair of keys or keypair, a public key and a private one. Public keys are used for encryption or signature verification; private ones decrypt and sign. The public key cryptography has two different keys but mathematically related to each other.[16] A public key and a private key was proposed by Whitfield Diffie and Martin Hellman in a ground breaking 1976 paper.[17] A public key is related to private key but a public key is constructed in such a way that calculation of one key (private key) is computationally infeasible from the other (the public key). But still both the keys are generated secretly as an interrelated pair. Public key cryptography is described as "the most revolutionary new concept in the field since polyalphabetic substitution emerged in the Renaissance".[18] The public-key is freely distributed in a public-key cryptosystems, while its paired private key must remain secret. In a public-key encryption system, encryption is done by using public key while for decryption private or secret key is used. Being unsuccessful in finding such a system Diffie and Hellman showed that by presenting the Diffie-Hellman key exchange protocol, public-key cryptography was indeed possible a solution that is now widely use in secure communication to allow two parties to secretly agree on shared encryption key.[19]

A widespread academic effort in finding a practical public-key encryption system was initiated due to Diffie and Hellman's publication, as a result in 1978 Ronald Rivest, Adi

Shamir and Len Adleman design the technique which is known as RSA algorithm.[20] Some other examples are Crammer-Shoup cryptosystem, ElGamal encryption and various elliptical curve techniques.[11]

1.2.4 Key Management

Key management is the management of cryptographic keys in a cryptosystem. This includes dealing with the generation, exchange, storage, use, and replacement of keys. Key management concerns keys at the user level, either between users or systems.[21] [22]

1.2.5 Key Generation

Key generation is the process of generating keys for cryptography. A key is used to encrypt and decrypt whatever data is being encrypted/decrypted. Modern cryptographic systems include symmetric-key algorithms (such as DES and AES) and public-key algorithms (such as RSA).[23]

1.2.6 Key Agreement

In cryptography, a key agreement protocol is a protocol whereby two or more parties can agree on a key in such a way that both influence the outcome. If properly done, this precludes undesired third-parties from forcing a key choice on the agreeing parties. Protocols that are useful in practice also do not reveal to any eavesdropping party what key has been agreed upon.[24]

1.2.7 Key Exchange

Key exchange (also known as "key establishment") is any method in cryptography by which cryptographic keys are exchanged between users, allowing use of a cryptographic algorithm.[25] If sender and receiver wish to exchange encrypted messages, each must be equipped to encrypt messages to be sent and decrypt messages received. The nature of the equipping they require depends on the encryption technique they might use. If they use a code, both will require a copy of the same codebook. If they use a cipher, they will need appropriate keys. If the cipher is a symmetric key cipher, both will need a copy of the same key. If an asymmetric key cipher with the public/private key property, both will need the other's public key. Prior to any secured communication, users must set up the details of the cryptography. In some instances this may require exchanging identical keys (in the case of a symmetric key system). In others it may require possessing the other party's public key. While public keys can be openly exchanged (their corresponding private key is kept secret), symmetric keys must be exchanged over a secure communication channel. Formerly, exchange of such a key was extremely troublesome, and was greatly eased by access to secure channels such as a diplomatic bag. Clear text exchange of symmetric keys would enable any interceptor to immediately learn the key, and any encrypted data. The advance of public key cryptography in the 1970s has made the exchange of keys less troublesome. Since the Whitfield Diffie and Martin Hellman published a cryptographic protocol, (Diffie–Hellman key exchange) in 1976, it has become possible to exchange a key over an insecure communications channel, which has substantially reduced the risk of key disclosure during distribution. It allows users to establish 'secure channels' on which to exchange keys, even if an opponent is monitoring that communication channel. However, Diffie-Hellman key exchange did not address the problem of being sure of the actual identity of the person (or 'entity'). In Diffie-Hellman key exchange algorithm two parties, who want to communicate securely, can agree on a symmetric key using this technique. This algorithm can be used for key agreement, but not for encryption or decryption of message. Once both the parties agree on the key to be used, they need to use other symmetric key encryption algorithms for actual encryption or decryption of messages. This Diffie-Hellman key exchange algorithm can fall pray to the Man-In-The-Middle attack, also called as Bucket Bridge Attack. This Man-In-The-Middle attack can work against the Diffie-Hellman key exchange algorithm, causing it to fail. This is plainly because the Man-In-The-Middle makes the actual communicators believe that they are taking to each other, whereas they are actually taking to the Man-In-The-Middle, who is taking to each of them.

1.2.8 Attack Model

- A *chosen-cipher text attack (CCA)* is an attack model for cryptanalysis in which the cryptanalyst gathers information, at least in part, by choosing a cipher text and obtaining its decryption under an unknown key. In the attack, an adversary has a chance to enter one or more known cipher texts into the system and obtain the resulting plaintexts. From these pieces of information the adversary can attempt to recover the hidden secret key used for decryption.[26]

- A *chosen-plaintext attack (CPA)* is an attack model for cryptanalysis which presumes that the attacker has the capability to choose arbitrary plaintexts to be encrypted and obtain the corresponding cipher texts. The goal of the attack is to gain some further information which reduces the security of the encryption scheme. In the worst case, a chosen-plaintext attack could reveal the scheme's secret key. For some chosen-plaintext attacks, only a small part of the plaintext needs to be chosen by the attacker: such attacks are known as plaintext injection attacks.[26]

- In cryptography, a *cipher text-only attack (COA)* or *known cipher text attack* is an attack model for cryptanalysis where the attacker is assumed to have access only to a set of cipher texts. The attack is completely successful if the corresponding plaintexts can be deduced, or even better, the key. The ability to obtain any information at all about the underlying plaintext is still considered a success. For example, if an adversary is sending cipher text continuously to maintain traffic-flow security, it would be very useful to be able to distinguish real messages from nulls. Even making an informed guess of the existence of real messages would facilitate traffic analysis.[26]

- The *known-plaintext attack (KPA)* is an attack model for cryptanalysis where the attacker has samples of both the plaintext (called a crib), and its encrypted version (cipher text). These can be used to reveal further secret information such as secret keys and code books. The term "crib" originated at Bletchley Park, the British World War II decryption operation.[26]

- The *Man-In-The-Middle Attack* (often abbreviated MITM, MitM, MIM, MiM, MITMA) in cryptography and computer security is a form of active eavesdropping in which the attacker makes independent connections with the victims and relays messages between them, making them believe that they are talking directly to each other over a private connection, when in fact the entire conversation is controlled by the attacker. The attacker must be able to intercept all messages going between the two victims and inject new ones, which is straightforward in many circumstances (for example, an attacker within reception range of an unencrypted Wi-Fi wireless access point, can insert himself as a Man-In-The-Middle).[27]

1.3 Cryptographic Algorithm

Out of large variety of cryptographic algorithms, few are discussed in section 1.3.1 to 1.3.4.

1.3.1 Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES) is a symmetric-key block cipher published by the National Institute of Standard and Technology (NIST) as FIPS 197 in the Federal Register in December 2001.[11] AES allows for three different key lengths: 128 bit keys, 192 bit keys and 256 bit keys where encryption consists of ten rounds of processing for 128 bit keys, twelve rounds for 192 bit keys and fourteen rounds for 256 bit keys. In each case, all other rounds are identical except for the last round. There are four steps for each round of processing: One single-byte based substitution, a row-wise permutation, a column-wise mixing and the addition of the round keys. The order of the above four steps is different for encryption and decryption.

1.3.2 Data Encryption Standard (DES)

Data Encryption Standard (DES) is a symmetric-key based block cipher. It was the result of a research project set up by International Business Machines (IBM) Corporation in the late 1960's.[13] DES is based on Feistel block cipher and only operates on 64 bit blocks of data at a time. After an initial permutation, the block is broken into a right half and a left half, each 32 bits long. There are sixteen rounds of identical operations in which the data are combined with the key with key length 56 bits. In each round, the bits of the key are shifted and then 48 bits are selected from the 56 bits of the key. The right half of the data is expanded to 48 bits via an expansion permutation, combined with 48 bits of a shifted and permuted key via an *Exclusive-OR*, sent through eight S-boxes producing 32 new bits and permuted again. After these four operations, the output is combined with the left half via another *Exclusive-OR*. The new right half is generated from the above operations and the old right half becomes the new left half. These operations are repeated for 16 times making 16 rounds of DES. After the sixteenth round, the right and left halves are joined and a final permutation, which is the inverse of the initial permutation, finishes off the DES algorithm.

1.3.3 Triple Data Encryption Standard (Triple DES)

The man-in-the-middle attack on Double DES has made the technique impractical and Double DES is seemed to be inadequate, therefore it paving the way for Triple DES.[13] Triple DES block cipher applies DES cipher thrice to each data block, where the block size is 64 bits. Triple DES uses three DES keys, K1, K2 and K3 (each of 56 bits, excluding parity bits), and the key sizes are $168\ (= 56 \times 3)$, $112\ (= 56 \times 2)$ or 56 bits with respect to keying option 1, 2 or 3 as follows:

Keying Option 1: All of the keys are independent.

Keying Option 2: K1 and K2 are independent and K3 = K1.

Keying Option 3: All of the keys are identical i.e. K1 = K2 = K3.

Keying Option 1 is the strongest with three independent keys with 168 key bits. Keying Option 2 provides less security with 112 key bits but stronger than the simply DES encrypting twice with keys K1 and K2. Keying Option 3, which has backward compatibility with DES, is equivalent to DES with 56 key bits.

The encryption and decryption algorithms of Triple DES with three independent keys are

$$\textit{Cipher Text} = E_{K3}\ (D_{K2}\ (E_{K1}\ (\textit{Plaintext})))$$

$$\textit{Plaintext} = D_{K1}\ (E_{K2}\ (D_{K3}\ (\textit{Cipher Text})))$$

The encryption and decryption algorithms of Triple DES with two independent keys are

$$\textit{Cipher Text} = E_{K1}\ (D_{K2}\ (E_{K1}\ (\textit{Plaintext})))$$

$$\textit{Plaintext} = D_{K1}\ (E_{K2}\ (D_{K1}\ (\textit{Cipher Text})))$$

1.3.4 RSA Algorithm

In 1978, Ron Rivest, Adi Shamir and Leonard Adleman introduced RSA algorithm which is an asymmetric key cryptosystem.[11] RSA involves the use of two keys: a public key, which may be known by anyone and used to encrypt messages and a private key, known only by the recipient and used to decrypt messages. A plaintext P is encrypted to cipher text C by $C = (P^e mod\, n)$ and the ciphertext C is decrypted into plaintext P by $P = (C^d\ mod\, n)$. Since knowing the factors of n, which will give away $\phi(n)$ and therefore d, a cryptanalyst would break the algorithm. The authors of RSA recommended that the length of n be about 200 digits long. However, this length may be varied based on the importance of the speed of encryption versus security.

1.4 Soft Computing based Cryptography

The advances in software technology assign more computational power. New computational environment becomes more distributed, more diverse and more global, the transmission of information is becoming more vulnerable to adversary attacks. Thus making the design of cryptographic schemes that can counter new cryptanalysis techniques is becoming harder. Recently soft computing approaches provide inspiration in solving problems from various fields. Now-a-days works in the application of soft computing inspired computational paradigm in cryptography become famous. The findings show that the research on applications of soft computing based approaches in cryptography is minimal as compared to other fields. Multiple disciplines have started to work together more closely for last few decades to improve the network security for reliable communication. A number of alternative cryptosystems have gained significant attention during these periods. Soft computing is the most promising one among them. Soft computing refers to the science of reasoning, thinking and deduction that recognizes and uses the real world phenomena of grouping, memberships, and classification of various quantities under study. As such, it is an extension of natural heuristics and capable of dealing with complex systems because it does not require strict mathematical definitions and distinctions for the system components. Soft computing differs from conventional (hard) computing in that, unlike hard computing, it is tolerant of imprecision, uncertainty, partial truth, and approximation. In effect, the role model for soft computing is the human mind. Soft computing is a term used in computer science to refer to problems in computer science whose solutions are unpredictable, uncertain and between 0 and 1. Soft computing became a formal area of study in computer science in the early 1990s.[28] Earlier computational approaches could model and precisely analyze only relatively simple systems. More complex systems arising in biology, medicine, the humanities, management sciences, and similar fields often remained intractable to conventional mathematical and analytical methods. That said, it should be pointed out that simplicity and complexity of systems are relative, and many conventional mathematical models have been both challenging and very productive. Soft computing deals with imprecision, uncertainty, partial truth, and approximation to achieve practicability, robustness and low solution cost. As such it forms the basis of a considerable amount of machine learning techniques. Recent trends tend to involve evolutionary and swarm intelligence based algorithms and bio-inspired computation in cryptography. Components of soft computing include:

Evolutionary algorithms[29][30] are adaptive methods, which may be used to solve search and optimization problems, based on the genetic processes of biological organisms. Over many generations, natural populations evolve according to the principles of natural selection and 'survival of the fittest'. By mimicking this process, evolutionary algorithms are able to 'evolve' solutions to real world problems, if they have been suitably encoded. Usually grouped under the term evolutionary algorithms or evolutionary computation[31][32], the domains are genetic algorithms, evolution strategies, evolutionary programming, genetic programming and learning classifier systems. [33] They all share a common conceptual base of simulating the evolution of individual structures via processes of selection, mutation, and reproduction. Cultural algorithms are computational models of cultural evolution. They consist of two basic components, a population space (using evolutionary algorithms), and a belief space. The two components interact by means of a vote-inherit-promote protocol. Likewise the knowledge acquired by the problem solving activities of the population can be stored in the belief space in the form of production rules etc. Cultural algorithms represent a general framework for producing hybrid evolutionary systems that integrate evolutionary search and domain knowledge.

The application of an evolutionary algorithm to the field of cryptography is rather unique. Few works exist on this topic. Using evolutionary algorithms most of the work has been done in the field of cryptanalysis. This nontraditional application is investigated to determine the benefits of applying an evolutionary algorithm to a cryptographic problem, if any. This area is so different from the application areas where evolutionary algorithms are developed. Major works that involves genetic algorithm focuses on cryptanalysis of cryptographic algorithms and design of cryptographic primitives. Most cryptanalytic research using Genetic Algorithm (GA) has been done on classical ciphers. An initial attempt conducted by Spillman et al.[34], whereby GA is exploited to cryptanalysis simple substitution ciphers. Since known cryptanalytic attack for simple substitution ciphers employs frequency distribution of characters in the message, Spillman derived a cost or fitness function based on single-character and diagram frequency distributions in this work. The attempt was fruitful as GA was proven to be highly successful in this cryptanalysis. Spillman suggested the use of trigram frequency distribution and variations on crossover and mutation procedures as future research. Spillman continues the work and illustrated that GA can also be used in the cryptanalysts of public key cryptosystem, the knapsack ciphers. The encryption scheme for knapsack ciphers is based on the NP-complete problem, which is a hard problem.[35] Another initial attempt conducted by Matthews for investigating the use of GA in cryptanalysis of transposition ciphers.[36] In this work the fitness function is based on the message length, frequency distribution of diagrams and trigrams tested for, the number of diagrams and trigrams checked for and the likelihood of occurrence in successful deciphered messages.

Swarm intelligence is aimed at collective behaviour of intelligent agents in decentralized systems. Most of the basic ideas are derived from the real swarms in the nature, which includes particle swarm, ant colonies, bird flocking, honeybees, bacteria and microorganisms etc. Swarm models are population-based and the population is initialized with a population of potential solutions. These individuals are then manipulated (optimized) over many several iterations using several heuristics inspired from the social behaviour of insects in an effort to find the optimal solution. Particle Swarm Optimization (PSO) emulates flocking behavior of birds and herding behavior of animals to solve optimization problems. The PSO was introduced by Kennedy and Eberhart.[37][38] In the PSO domain, there are two main variants: global PSO and local PSO. In the local version of the PSO, each particle’s velocity is adjusted according to its personal best position pbest and the best position lbest achieved so far within its neighborhood. The global PSO learns from the personal best position pbest and the best position gbest achieved so far by the whole population.

Ant Colony Optimization (ACO) algorithms are inspired by the behavior of natural ant colonies, in the sense that they solve their problems by multi agent cooperation using indirect communication through modifications in the environment. This algorithm is a member of the Ant Colony algorithms family, in swarm intelligence methods, and it constitutes some metaheuristic optimizations. ACO was initially proposed by Marco Dorigo in his PhD thesis.[39][40] Ants release a certain amount of pheromone (hormone) while walking, and each ant prefers (probabilistically) to follow a direction, which is rich of pheromone. This simple behavior explains why ants are able to adjust to changes in the environment, such as optimizing shortest path to a food source or a nest. In ACO, ants use information collected during past simulations to direct their search and this information is available and modified through the environment. Bafghi performed a differential cryptanalysis on Serpent using Ant Colony and claimed that it can be used for any block cipher. [41] Ant colony algorithms are multi-agent systems where the behavior of each single agent, the ants, is inspired by the behavior of real ants.

Simulated Annealing (SA) is based on the manner in which liquids freeze or metals recrystalize in the process of annealing. The method was independently described by Kirkpatrick et al.[42] and Černý.[43] The method is an adaptation of the Metropolis-Hastings algorithm, a Monte Carlo method to generate sample states of a thermodynamic system, invented by Rosenbluth and published in a paper by Metropolis et al..[44] In an annealing process, molten metal, initially at high temperature, is slowly cooled so that the system at any time is approximately in thermodynamic equilibrium. If the initial temperature of the system is too low or cooling is done insufficiently slowly the system may become brittle or unstable with forming defects. The initial state of a thermodynamic system is set at energy E and temperature T, holding T constant the initial configuration is perturbed and the change in energy dE is computed. If the change in energy is negative the new configuration is accepted. If the change in energy is positive it is accepted with a probability given by the Boltzmann factor $exp\ -(dE/T)$. This processes is then repeated for few iterations to give good sampling statistics for the current temperature, and then the temperature is decremented and the entire process repeated until a frozen state is achieved at $T = 0$. An extensive research on classical cipher cryptanalysis was investigated by Bagnall[45] and Clark in his Ph.D work[46]. Clark's cryptanalytic attack work covers a variety of classical ciphers that include simple substitution, transposition as well as poly-alphabetic ciphers. Clark proposed new attacks on these ciphers, which utilize SA and the tabu search. Tabu search, created by Glover in 1986[47] and formalized in 1989[48][49], is a metaheuristic search method employing local search methods used for mathematical optimization. Existing attacks which make use of the GA and SA are compared with the new SA and tabu search techniques.

Artificial Neural Network (ANN)[50][51] have been developed as generalizations of mathematical models of biological nervous systems. In a simplified mathematical model of the neuron, the effects of the synapses are represented by weights that modulate the effect of the associated input signals, and the nonlinear characteristic exhibited by neurons is represented by a transfer function, which is usually the sigmoid, Gaussian function etc.[52][53] The neuron impulse is then computed as the weighted sum of the input signals, transformed by the transfer function. The learning capability of an artificial neuron is achieved by adjusting the weights in accordance to the chosen learning algorithm.[54][55]

Neural Cryptography[56][57] is a branch of cryptography dedicated to analyzing the application of stochastic algorithms, especially ANN algorithms, for use in encryption and cryptanalysis. ANNs are well known for their ability to selectively explore the solution space of a given problem. This feature finds a natural niche of application in the field of cryptanalysis. At the same time, ANNs offer a new approach to attack ciphering algorithms based on the principle that any function could be reproduced by an ANN, which is a powerful proven computational tool that can be used to find the inverse-function of any cryptographic algorithm. The ideas of mutual learning, self learning, and stochastic behavior of ANNs and similar algorithms can be used for different aspects of cryptography, like public-key cryptography, solving the key distribution problem using ANN mutual synchronization, hashing or generation of pseudo-random numbers. Another idea is the ability of a ANN to separate space in non-linear pieces using "bias". It gives different probabilities[58][59] of activating or not the ANN. This is very useful in the case of cryptanalysis. Two names are used to design the same domain of researches: Neuro-Cryptography and Neural Cryptography. The most used protocol for key exchange between two parties A and B in the practice is Diffie-Hellman protocol. Neural key exchange, which is based on the synchronization of two Tree Parity Machines (TPM), should be a secure replacement for this method. Synchronizing these two machines is similar to synchronizing two chaotic oscillators in chaos communications.

1.5 Literature Survey

Currently new computational environment becomes more distributed, more diverse and more global; the transmission of information is becoming more vulnerable to adversary attacks. Now-a-days appropriate cryptographic technique in light weight devices having very low processing capabilities or limited computing power in wireless communication is the major challenge. Thus making the design of light weight cryptographic schemes for low processing devices that can counter new cryptanalysis techniques in wireless communication is becoming harder. Therefore, computer network security is a fast moving technology in the field of computer science. Network security using cryptography originally focused on mathematical and algorithmic aspects. As security techniques continue to mature, there is an emerging set of cryptographic techniques always. This advancement of digital communication technology benefitted the field of cryptography. The efficient cryptographic schemes were designed and implemented and also broken subsequently over time.

Metropolis et al. devised an algorithm about Simulated Annealing (SA) method in the year 1953. [44] SA is an adaptation of the Metropolis-Hastings algorithm, a Monte Carlo method to generate sample states of a thermodynamic system, invented by Rosenbluth. Whitfield Diffie and Martin Hellman published a cryptographic protocol, (Diffie–Hellman key exchange) in 1976; to exchange a key over an insecure communications channel, which has substantially reduced the risk of key disclosure during distribution.[17] It allows users to establish 'secure channels' on which to exchange keys, even if an opponent is monitoring that communication channel. However, Diffie–Hellman key exchange did not address the problem of being sure of the actual identity of the person (or 'entity'). So, Diffie–Hellman key exchange is vulnerable to Man-In-The-Middle (MITM) attack. Kirkpatrick et al.[42] in the year 1983 and Černý et al.[43] in the year 1985 independently described Simulated Annealing which is based on the manner in which liquids freeze or metals recrystalize in the process of annealing. Tabu search, proposed by Fred W. Glover[47] in 1986 and formalized in 1989[48] and 1990[49] is a metaheuristic search method employing local search methods used for mathematical optimization. Colorni et al. proposed a scheme of distributed optimization by Ant Colonies in the year 1991.[39] Marco Dorigo in his Ph.D thesis proposed Ant Colony Optimization (ACO) in 1992.[40] This algorithm is a member of the Ant Colony algorithms family, in swarm intelligence methods, and it constitutes some metaheuristic optimizations. Ants release a certain amount of pheromone (hormone) while walking, and each ant prefers (probabilistically) to follow a direction, which is rich of pheromone. In ACO, ants use information collected during past simulations to direct their search and this information is available and modified through the environment. The application of an evolutionary algorithm to the field of cryptography is rather unique. Few works exist on this topic. Using evolutionary algorithms most of the work has been done in the field of cryptanalysis. Major works that involves GA focuses on cryptanalysis of cryptographic algorithms and design of cryptographic primitives. Most cryptanalytic research using GA was done on classical ciphers. An initial attempt was conducted by Spillman et al.[34] in 1993, whereby GA is exploited to cryptanalysis simple substitution ciphers. Since known cryptanalytic attack for simple substitution ciphers employs frequency distribution of characters in the message, Spillman derived a cost or fitness function based on single-character and diagram frequency distributions in the work. This attempt was fruitful as GA was proven to be highly successful in this cryptanalysis. Spillman suggested the use of trigram frequency distribution and variations on crossover and mutation procedures as future research. Spillman continues the work and illustrated that GA can also be used in the cryptanalysts of public key cryptosystem, the knapsack ciphers. The encryption scheme for knapsack ciphers is based on the NP-complete problem, which is a hard problem.[35] Maurer considered the problem of MITM attack for generating a shared secret key S by two parties knowing dependent random variables X and Y, respectively, but not sharing a secret key initially.[60] An enemy who knows the random variable Z, jointly distributed with X and Y according to some probability distribution, can also receive all messages exchanged by the two parties over a public channel. The goal of Maurer research is to develop a protocol is that the enemy obtains at most a negligible amount of information about S. Author shows that such a secret key agreement is possible for a scenario in which all three parties receive the output of a binary symmetric source over independent binary symmetric channels, even when the enemy's channel is superior to the other two channels. But it may not be always possible to receive the output of a binary symmetric source over independent binary symmetric channels in wireless communication. Zadeh et al. in 1994 described the concept of soft computing, Artificial Neural Networks and Fuzzy Logic.[28] Soft Computing became a formal area of study in computer science in the early 1990s. Particle Swarm Optimization (PSO) in the year 1995.[37] Then they analyzed the small worlds and mega-minds effects of neighborhood topology on particle swarm performance.[38] Swarm intelligence is aimed at collective behaviour of intelligent agents in decentralized systems. Most of the basic ideas are derived from the real swarms in the nature, which includes particle swarm, ant colonies, bird flocking, honeybees, bacteria and microorganisms etc. Delgado-Restituto et al. proposed the use of analog integrated circuits for secure communication based on chaos synchronization.[61] This phenomenon is demonstrated through experimental measurements realized on silicon prototypes in a double-metal, single-poly 1.6 μm CMOS technology. The approach is not simple and not suitable for light weight devices having very low processing capabilities in wireless communication. Dourlens presented the application of ANNs in classical cryptography in MSc Thesis named Neuro-Cryptography.[62] Caponetto et al. offered a state controlled cellular neural network-based circuit for secure transmission applications.[63] In this work the basic principles of synchronization between two (or more) chaotic systems are reported concerning the inverse system technique. Fundamentals of this kind of transmission are briefly introduced together with some experimental results. The main problem of this technique is that some of the parameters are predefined in the chaotic systems and still this technique suffers from vulnerability of public channel. Bäck et al. discussed about the evolution strategies, evolutionary programming and genetic algorithms.[29][30] Evolutionary algorithm are adaptive methods, which is used to solve search and optimization problems, based on the genetic processes of biological organisms. An extensive research on classical cipher cryptanalysis using Simulated Annealing investigated by Bagnall[45] and Clark in his Ph.D work[46]. Clark's cryptanalytic attack work covers a variety of classical ciphers that include simple substitution, transposition as well as poly-alphabetic ciphers. Clark proposed new attacks on these ciphers, which utilize Simulated Annealing and the tabu search. Metzler et al. focused on several scenarios of interacting Neural Networks for using competitive perceptrons as decision-making algorithms in a model of a closed market.[64] Neural

Networks which are trained either in an identical or in a competitive way are solved analytically. In the case of identical training each perceptron receives the output of its neighbor. The symmetry of the stationary state as well as the sensitivity to the used training algorithms are investigated. Two competitive perceptrons trained on mutually exclusive learning aims and a perceptron which is trained on the opposite of its own output are examined analytically. Authors of this paper soon discover that dynamics of interacting perceptrons is solved analytically. They have observed that for a directed flow of information the system runs into a state which has a higher symmetry than the topology of the model. A symmetry breaking phase transition is found with increasing learning rate. In addition it is shown that a system of interacting perceptrons which is trained on the history of its minority decisions develops a good strategy for the problem of adaptive competition known as the Bar Problem or Minority Game. Using this competitive perceptrons training approach authors of this paper Kanter and Kinzel came up with revolutionary approach in cryptography called Neural Cryptography.[56] It is a connection between the theory of Neural Networks and cryptography. This turned out to be a new branch of cryptography and became very popular soon. Kanter and Kinzel shows the secure exchange of information by synchronization of two Neural Networks which are trained on their mutual output bits.[57] Using numerical simulations, Kanter and Kinzel shows that two artificial networks being trained by Hebbian learning rule on their mutual outputs develop an antiparallel state of their synaptic weights.[65] From the novel phenomenon Kanter et al. conclude that the synchronized weights are used to construct an ephemeral key exchange protocol for a secure transmission of secret data.[66] It is shown that an opponent who knows the protocol and all details of any transmission of the data has no chance to decrypt the secret message, since tracking the weights is a hard problem compared to synchronization. The complexity of the generation of the secure channel is linear with the size of the network. Two Neural Networks which are trained on their mutual output bits are analyzed using methods of statistical physics. So, interacting Neural Networks and cryptography together. Rosen-Zvi et al. studied the mutual learning process between two parity feed-forward networks with discrete and continuous weights analytically, and they found that the number of steps required to achieve full synchronization between the two networks in the case of discrete weights is finite.[67] The synchronization process is shown to be non-self-averaging and the analytical solution is based on random auxiliary variables. The learning time of an attacker that is trying to imitate one of the networks is examined analytically and is found to be much longer than the synchronization time. An algorithm for an eavesdropper which could break the key was introduced by Klimov et al. by analyzing the scheme and explains why the two parties converge to a common key, and why an attacker using a similar neural network is unlikely to converge to the same key.[68] However, authors shown that this key exchange protocol can be broken in three different ways, and thus it is completely insecure. For this reason Mislovaty et al. investigated the security of Neural Cryptography very minutely.[69] The weights of the networks have integer values between $\pm L$. Authors shown that the synchronization time increases with L^2 while the probability to find a successful attacker decreases exponentially with L. Hence for large L authors find a secure key-exchange protocol which depends neither on number theory nor on injective trapdoor functions used in conventional cryptography. Rosen-Zvi et al. analyzed the mutual learning features in a Tree Parity Machine (TPM) and its application to the cryptography.[70] Mutual learning of a pair of TPMs with continuous and discrete weight vectors is studied analytically. The analysis is based on a mapping procedure that maps the mutual learning in TPMs onto mutual learning in noisy perceptrons. The stationary solution of the mutual learning in the case of continuous TPMs depends on the learning rate where a phase transition from partial to full synchronization is observed. In the discrete case the learning process is based on a finite increment and a full synchronized state is achieved in a finite number of steps. The synchronization of discrete parity machines is introduced in order to construct an ephemeral key exchange protocol. The dynamic learning of a TPM (an attacker) that tries to imitate one of the two machines while the two still update their weight vectors is also analyzed. Now, Kinzel et al. presented a connection between the theory of Neural Networks and cryptography.[71] A new phenomenon, namely synchronization of Neural Networks, is leading to a new method of exchange of secret messages. Kanter et al. presented the mutual synchronization of Neural Networks to analyze the theory of Neural Networks and cryptography.[72] Kinzel et al. shows when Neural Networks are trained on their own output signals they generate disordered time series.[73] This disorder generated by interacting Neural Networks has an application to econophysics and cryptography. When agents competing in a closed market (minority game) are using Neural Networks to make their decisions, the total system relaxes to a state of good performance is an application of econophysics and two partners communicating over a public channel can find a common secret key is an application of cryptography. Mislovaty et al. construct a hybrid network in public channel cryptography by synchronizing of Neural Networks and chaotic maps. In this network the external signal to the chaotic maps is synchronized by the neural nets.[74] This allows a secure generation of secret encryption keys over a public channel. Another initial attempt conducted by Matthews investigated the use of GA in cryptanalysis of transposition ciphers.[36] In this work the fitness function is based on the message length, frequency distribution of diagrams and trigrams tested for, the number of diagrams and trigrams checked for and the likelihood of occurrence in successful deciphered messages. Bafghi performed a differential cryptanalysis on Serpent using Ant Colony and claimed that it can be used for any block cipher.[41] Shacham et al. presented a successful attack strategy in Neural Cryptography.[75] In this attack cooperating attackers are involved for breaking the security of the Neural Cryptography. A successful attack strategy in Neural Cryptography is presented. The neural cryptosystem, based on synchronization of Neural Networks by mutual learning, has been recently shown to be secure under different attack strategies. Mislovaty et al. also analyze the security of Neural Cryptography.[76] The success of the advanced attacker presented by them, called the "majority-flipping attacker," does not decay with the parameters of the model. This attacker's outstanding success is due to its using a group of attackers which cooperate throughout the synchronization process, unlike any other attack strategy known. Ruttor et al. analyze the synchronization of Random Walks with reflecting boundaries.[77] They have shown that reflecting boundary conditions cause two one-dimensional Random Walks to synchronize if a common direction is chosen in each step. The mean synchronization time and its standard deviation are calculated analytically. Both quantities are found to increase proportional to the square of the system size. Additionally, in this method the probability of synchronization in a given step is analyzed, which converges to a geometric distribution for long synchronization times. From this asymptotic behaviour the number of steps required to synchronize an ensemble of independent Random Walk pairs is deduced. They have observed that the synchronization time increases with the logarithm of the ensemble size. To enhance the security Ruttor et al. also proposed a feedback mechanism in Neural Cryptography for increasing the security of the network.[78] Neural Cryptography is based on a competition between attractive and repulsive stochastic forces. A feedback mechanism increases the repulsive forces. Using numerical simulations and an analytic approach, they have calculated the probability of a successful attack for different model parameters. They also derived the scaling laws which show that feedback improves the security of the system. Volkmer et al. thought of authenticated TPM for key exchange purpose.[79] The synchronization of TPMs, has proven to provide a valuable alternative concept for secure symmetric key exchange. Yet, from a cryptographer's point of view, authentication is at least as important as a secure exchange of keys. Adding an authentication via hashing e.g. is straightforward but with no relation to Neural Cryptography. They have presented an alternative, integrating a Zero-Knowledge protocol into the synchronization. A Man-In-The-Middle attack and even all currently known attacks, that are based on using identically structured TPMs and synchronization as well, can so be averted. This in turn has practical consequences on using the trajectory in weight space. Next to authentication, secure key exchange is considered the most critical and complex issue regarding ad-hoc network security. Volkmer et al. also presented a low-cost, (i.e. low hardware-complexity) solution for feasible frequent symmetric key exchange in adhoc networks, based on a Tree Parity Machine Rekeying Architecture.[80] Using this TPM a key exchange can be performed within a few milliseconds, given practical wireless communication channels and their limited bandwidths. Ruttor et al. presented Neural Cryptography with queries.[81] Neural Cryptography is based on synchronization of TPMs by mutual learning. They extend previous key exchange protocols by replacing random inputs with queries depending on the current state of the Neural Networks. The probability of a successful attack shows that queries restore the security against cooperating attackers. The success probability can be reduced without increasing the average synchronization time. Based on synchronization of Neural Networks by mutual learning Klein et al. suggested several models for this cryptographic system, and have been tested for their security under different sophisticated attack strategies.[82] Then they conclude that the most promising models are networks that involve chaos synchronization. Kanter et al. presented a detail analysis of the theory of Neural Networks: learning from examples, time-series and cryptography.[83] These detail analysis actually deals with the storage capacity of the TPM, learning from examples, and time series generation by feed forward networks. Kotlarz et al. proposed a new schedule of S-boxes design in their paper.[84] They presented the most popular S-box design criteria, especially a possibility of application of Boolean bent-functions. Finally, they propose integrating Neural Networks (playing a role of Boolean functions with appropriate properties) in the design process. The necessity of securing the communication between hardware components in embedded systems becomes increasingly important with regard to the secrecy of data and particularly its commercial use. Volkmer et al. suggested Tree Parity Machine Rekeying Architectures and a low-cost (i.e., small logic-area) solution for flexible security levels and short key lifetimes.[85] The basis is an approach for symmetric key exchange using the synchronization of TPMs. So, they proposed a TPM based key establishment IP-Core for ubiquitous computing.[86] Fast successive key generation enables a key exchange within a few milliseconds, given realistic communication channels with a limited bandwidth. Alternative security solutions are considered in science and industry, motivated by the strong restrictions as they are often present in embedded security scenarios especially in a RFID setting. They investigated a low hardware-complexity cryptosystem for lightweight symmetric key exchange and stream cipher based on TPMs.[87] They decided that Tree Parity Machine Rekeying Architectures can be used for embedded security.[88] The speed of a key exchange is basically only limited by the channel capacity as is the stream cipher throughput. This work significantly improves and extends previously published results on Tree Parity Machine Rekeying Architectures. Identity-based public key cryptosystem may perfectly substitute the traditional certificate-based public key system if only the efficiency and security of key issuing are satisfied. Batina et al. proposed a framework and platform to compare stream ciphers not only on their security level but also based on their energy consumption, performance and area cost.[89] They described the basic hardware assumptions, give the area, delay and power consumption values of some existing stream ciphers and give guidelines for the designs of future algorithms. Interactions of neural network has been studied out coming a novel result that the two Neural Networks can synchronize to a stationary weight state with the same initial inputs. Based on this approach Chen et al. proposed a remote user authentication and identity-based key issuing scheme.[90][91] This simple but novel interacting neural network based scheme for secure key agreement purpose, and ID-based private key secure issuing over a complete public channel, which can provide a full dynamic and security remote user authentication over a completely insecure communication channel. Gross et al. proposed a framework for public-channel cryptography using chaotic lasers.[92] Two mutually coupled chaotic diode lasers with individual external feedback, are used to establish chaos synchronization in the low-frequency fluctuations regime. A third laser with identical external feedback but coupled unidirectionally to one of the pair does not synchronize. Both experiments and simulations reveal the existence of a window of parameters for which synchronization by mutual coupling is possible but synchronization by unidirectional coupling is not. Klein et al. proposed a key-exchange protocol that comprises two parties with chaotic dynamics that are mutually coupled and undergo a synchronization process, at the end of which they can use their identical dynamical state as an encryption key.[93] The transferred coupling- signals are based nonlinearly on time-delayed states of the parties, and therefore they conceal the parties' current state and can be transferred over a public channel. JCH Castro et al. shows the application of evolutionary computation in computer security and cryptography.[94] The main objective of the authors is to consider the problem of defining fitness function for the evolutionary algorithms like GA, SA etc. Godhavari et al. uses the concept of neural synchronization by mutual learning to a secret key exchange protocol over a public for encrypting and decrypting the given message using DES algorithm which is simulated and synthesized using VHDL.[95] Klein et al. shows stable isochronal synchronization of mutually coupled chaotic lasers.[96] The dynamics of two mutually coupled chaotic diode lasers are investigated experimentally and numerically by them. By adding self-feedback to each laser, stable isochronal synchronization is established. This stability, which can be achieved for symmetric operation, is essential for constructing an optical public-channel cryptographic system. Ruttor et al. appied the genetic attack on Neural Cryptography.[97] A genetic algorithm, which selects the fittest Neural Networks for attack. The probability of a successful genetic attack is calculated for different model parameters using numerical simulations. Ruttor presented the detail analysis about the Neural Cryptography in Ph.D thesis named Neural Synchronization and Cryptography.[98] Ruttor et al. again analysis the dynamics of Neural Cryptography.[99] In the case of TPMs the dynamics of both bidirectional synchronization and unidirectional learning is driven by attractive and repulsive stochastic forces. They described it by a Random Walk model for the overlap between participating Neural Networks. For that purpose transition probabilities and scaling laws for the step sizes are derived analytically. Both these calculations as well as numerical simulations show that bidirectional interaction leads to full synchronization on average. In contrast, successful learning is only possible by means of fluctuations. Consequently, synchronization is much faster than learning, which is essential for the security of the neural key-exchange protocol. The protection of chip-level microcomputer bus systems in embedded devices is essential to prevent the growing number of hardware hacking attacks. Müehlbach et al. presented an authenticated key exchange and encryption solution in order to ensure chip-level microcomputer bus systems via the Tree Parity Machine Rekeying Architecture (TPMRA).[100] Due to this intention, a scalable TPMRA IP-core is designed and implemented in order to meet variable bus performance requirements. It allows the authentication of the bus participants as well as the encryption of chip-to-chip buses from a single primitive. The solution is transparent and easy applicable to an arbitrary microcomputer bus system for embedded devices on the market. Saballus et al. proposed secure group communication in ad-hoc networks using Tree Parity Machines.[101] This can be divided into key agreement and key distribution. Common group key agreement protocols are based on the Diffie-Hellman key exchange and extend it to groups. Group key distribution protocols are centralized approaches which make use of one or more special key servers. In contrast to these approaches, they present a protocol which makes use of the TPM key exchange between multiple parties. Patra el al. presented a new concept of key agreement, using chaos synchronization based parameter estimation of two chaotic systems.[102] Laskari et al. addresses the issue of cryptography and cryptanalysis through computational intelligence.[103] Arvandi et al. proposed a neural network-based symmetric cipher design methodology to provide high performance data encryption.[104] The proposed approach is a novel attempt to apply the parallel processing capability of Neural Networks for cryptography purposes. A Diffie-Hellman public-key cryptography based on chaotic attractors of Neural Networks is described by Liu et al.[105] There is a one-way function between chaotic attractors and initial states in an Overstoraged Hopfield Neural Networks (OHNN). If the synaptic matrix of OHNN is changed, each attractor and its corresponding domain of initial state attraction will be changed. Then, the neural synaptic matrix as a trap door, and change it with commutative random permutation matrix. A new Diffie-Hellman public-key cryptosystem can be implemented, keeping the random permutation operation of the neural synaptic matrix as the secret key, and the neural synaptic matrix after permutation as public-key. Hen et al.

analyzed and optimized the interacting network neural, then present a cryptography-oriented secure parity model and implement the performance simulations.[106] Li et al. presented a new and effective attack strategy on Neural Cryptography.[107] Their proposal focuses on the authentication which the neural cryptosystem takes little account of. Chen et al. proposed two TPM-based novel OTP solutions.[108] One is a full implementation model including initialization and rekeying, while the other is light-weight and efficient suitable for resource-constrained embedded environment. Dong et al. presented an new authentication method using Neural Cryptography on WiMAX.[109] Dong et al. also presented a new security solution in ubiquitous computing.[110] They explored the challenges for building security and privacy into ubiquitous computing, described their prototype implementation based on Neural Cryptography.[111] Yunpeng et al. proposed the improvement of public key cryptography based on chaotic Neural Networks.[112] By adopting a kind of hybrid-coding and chaotic map, the modified algorithm performs better result on avalanche test. Shouhong et al. proposed password authentication using Hopfield Neural Networks.[113] The conventional verification table approach has significant drawbacks. Neural Networks have been used for password authentication to overcome the shortcomings of traditional approaches. In neural network approaches to password authentication, no verification table is needed; rather, encrypted neural network weights are stored within the system. Tieming et al. proposed the improved secure TPM which can be utilized to synchronize parameters for OTP schemes.[114] Authors introduced the TPM mutual learning scheme and the two TPM-based novel OTP solutions. One is a full implementation model including initialization and rekeying, while the other is light-weight and efficient suitable for resource-constrained embedded environment. Arvandi et al. described an innovative form of cipher design based on the use of recurrent Neural Networks.[115] The proposed cipher has a relatively simple architecture and, by incorporating Neural Networks, it releases the constraint on the length of the secret key. The design of the symmetric cipher is described in detail and its security is analyzed. Dong et al. presented a new service-based computing security model, which is combined with Neural Cryptography.[116] Service-based computing is a new and hot research point for telecommunication and computer scientist. Neural Cryptography is a new way to create shared secret key. The existed system architecture mentions little about security. Synchronization of Neural Networks is an alternative to cryptographic applications such as the realization of symmetric key exchange protocols. Reyes et al. proposed a first view of the so-called Permutation Parity Machine (PPM), an ANN proposed as a binary variant of the TPM.[117][118] The dynamics of the synchronization process by mutual learning between PPMs is analytically studied and the results are compared with those of TPMs. It will turn out that for neural synchronization, PPMs form a viable alternative to TPMs. Pulses of synchronization in chaotic coupled map lattices discussed by Schmitzer et al. in the context of transmission of information.[119] Synchronization and desynchronization propagate along the chain with different velocities which are calculated analytically from the spectrum of convective Lyapunov exponents. Since the front of synchronization travels slower than the front of desynchronization, the maximal possible chain length for which information can be transmitted by modulating the first unit of the chain is bounded. Wallner et al. investigated the implementation of a low hardware complexity cryptosystem for lightweight (authenticated) symmetric key exchange, based on two new Tree Parity Machine Rekeying Architectures (TPMRAs).[120] This work significantly extends and optimizes (number of gates) previously published results on TPMRAs. Lian et al. constructed a hash function based on a three-layer neural network.[121] The three neuron-layers are used to realize data confusion, diffusion and compression respectively, and the multi-block hash mode is presented to support the plaintext with variable length. Allam et al. proposed three new algorithms to enhance the mutual learning process.[122] Ahmad et al. compared between stream cipher and block cipher using RC4 and Hill Cipher.[123] The authors introduced two keys used for encrypting the information transferred during communication by using the Meet in the Middle Attack on triple S-DES algorithm, instead of using Brute force attack. Revankar et al. introduced a query based mutual influence between A and B which is not available to an attacking network E[124]. In this work query incorporated to the case of the Hebbian training rule. Tirdad et al. proposed an application of Hopfield Neural Networks (HNN) as pseudo random number generator.[125] This is done based on a unique property of HNN, i.e., its unpredictable behavior under certain conditions. They compared the main features of ideal random number generators with those of PRNG based on Hopfield Neural Networks. Prabakaran et al. proposed a scheme where TPMs random inputs are replaced with queries for cooperating attackers and effective number of keys.[126][127] The queries depend on the current state of A and B TPMs. Then, TPM's hidden layers of each output vector are compared. That is, the output vector of hidden unit using Hebbian learning rule and dynamic unit using Random Walk learning rule are compared. Among the compared values, the output layer receives one of the best values. Cyclic Cryptography, a different cryptographic system, has been proposed by Chowdhury et al. and its allied characteristics are implemented.[128] A data encryption technique using genetic crossover of robust biometric key and session based password has been introduced by Bhattacharya et al. where the key is obtained by crossing over of the session key generated from the password given by the legitimate user and the biometric key generated from the fingerprint of the same user.[129] A public key cryptosystem based on the system of higher order Diophantine equations has been proposed by Yosh et al..[130] In this system those Diophantine equations are used as public keys for sender and recipient, and both sender and recipient can obtain the shared secret through a trapdoor, while attackers must solve those Diophantine equations without trapdoor. This technique is based on complex mathematics. Jogdand et al. used the existing concept of Neural Cryptography, where both the communicating networks receive an identical input vector, generate an output bit and are trained based on the output bit.[131] The two networks and their weight vectors exhibit a novel phenomenon, where the networks synchronize to a state with identical time-dependent weights. Allam et al. suggested an algorithm that employs and extends the mutual learning process to accommodate the much needed group secure communication.[132] A new key generation mechanism has been introduced and amalgamated by Saeed et al. with the technique termed as “Fauzan-Mustafa Encryption Technique (FMET)”.[133] Karas et al. presented a novel PHY-layer security algorithm whose function is based on Neural Networks.[134] Specifically, they present a full key exchange scheme which includes channel sampling and thresholding and neural network based error reconciliation. Li et al. proposed and analyzed a parallel hash algorithm construction based on chaotic maps with changeable parameters.[135] The two main characteristics of the proposed algorithm are parallel processing mode and message expansion. Lu´ıs et al. presented a successful attack on PPM based Neural Cryptography.[136] Rasool et al. proposed a symmetric key encryption technique which provides security to both the message and the secret key achieving confidentiality and authentication.[137] In this technique, the security level is higher due to the inherent poly-alphabetic nature of the substitution mapping method used here, together with the translation and transposition operations performed in the algorithm. A new distributed key generation technique for threshold cryptography has been introduced by Qian et al. using bivariate symmetric polynomials.[138] The technique is based on some group G^6 which is either a cyclic additive group of prime order q or a cyclic multiplicative group with an element of prime order q. An extensive and careful study has been performed by Gajbhiye et al. on the applications of elliptic curve cryptography (ECC) and on different forms of elliptic curve in various coordinate systems specifying which is most widely used and why, on extended form of elliptic curve i.e. hyper-elliptic curve (HEC) with its pros and cons, on the performance of ECC and HEC based on scalar multiplication and DLP.[139] A cryptographic scheme has been proposed by Vijayakumar et al. which provides first level of security with smaller key size and less computation overhead using DNA Computing technique and the second level of security is provided by using the encryption and decryption algorithms of low computation Elliptic Curve Cryptography (ECC).[140] The novelty of this scheme is advantages of both ECC and DNA computation is exploited in providing a high level of data security. A public key cryptographic technique has been introduced by Som et al. using Genetic algorithm where bit level XOR operation followed by Genetic crossover and mutation during encryption.[141] Das et al. have introduced an integrated symmetric key cryptographic method combining two independent methods modified generalized Vernam cipher method and DJSA method.[142] A hybrid encryption technique has been introduced by Patheja et al. using Tiger algorithm.[143] In Tiger algorithm there is double protection of Data using triple DES and with the help of this algorithm transmission of data will be more secure for exchanging data over short distances from one device to another. The characteristics and performance related issues has been discussed by Kumar et al. for several symmetric block cipher algorithms like MARS, RC6, Serpent, Twofish, Rijndael and asymmetric cryptosystems like RSA, ECC, ECRYPT, HASH, DSAsg.[144]). A different symmetric key based cryptographic algorithm has been developed by Gupta et al. where block based substitution method, logical operations like XOR and shifting operations are used.[145] Based on the concept of Rijndael algorithm, a cryptographic algorithm has been developed by Rayarikar et al..[146] The algorithm uses various invertible, self-invertible and non-invertible components of modern encryption ciphers and key generation same as that of AES. A hybrid security enhancement algorithm has been designed and implemented by Kaul et al. based on AES-DES algorithms using 128 bit key.[147] Enhanced Identity-Base Cryptography (EIBC) has been proposed by Nicanfar et al. which is an efficient key management mechanism that minimizes control packets to reduce the communication overheads.[148] Cryptanalytic attack on DES, which is a known-plaintext attack based on neural networks, has been discussed by Alani.[149] In this work a trained neural network retrieves parts of plaintext from cipher text without retrieving the key used in encryption. A verification strategy in the exhaustive search step of the linear attack has been designed to allow Eve to mount a successful attack in the noisy environment. The most popular and efficient encryption algorithms in smart cards such as RSA, ECC, DES and ECDSA were described and compared between these algorithms by Savari et al. to find out the differences.[150] Banerjee et al. considered the phenomena of chaos synchronization with bidirectional linear feedback coupling. The synchronized system can be used as a cryptosystem, where both the model can be considered as a transceiver. They have proposed an asymmetric cryptographic scheme for ensuring security of data being transmitted in the above manner.[151] Seoane et al. presented an algorithm which implements a probabilistic attack on the key exchange protocol based on PPMs.[152] Instead of imitating the synchronization of the communicating partners, the strategy consists of a Monte Carlo method to sample the space of possible weights during inner rounds and an analytic approach to convey the extracted information from one outer round to the next one. Urbanovich et al. considered the hash function built on ANN.[153] The data about the process of synchronization of an ANN, obtained by experiment, are presented. The fact, that the obtained vector of weight coefficients for the networks after the synchronization is different for each new session, is determined. Santhanalakshmi et al. proposed a genetic approach has been used in the field of Neural Cryptography for synchronizing TPMs by mutual learning process.[154] Here a best fit weight vector is found using a genetic algorithm and then the training process is done for the feed forward network. The proposed approach improves the process of synchronization. Winkler et al. investigated the effect of dynamic adaptive couplings on the cooperative behavior of chaotic networks.[155] The couplings adjust to the activities of its two units by two competing mechanisms: An exponential decrease of the coupling strength is compensated for by an increase due to desynchronized activity. This mechanism prevents the network from reaching a steady state. Numerical simulations of a coupled map lattice show chaotic trajectories of desynchronized units interrupted by pulses of mutually synchronized clusters. Dolecki presented the statistical analysis on TPM synchronization time.[156] The author described the features of architecture and the principles of interaction of two ANNs. Synchronization status of networks allows using the relevant information as a key to encrypt further communications. The design principles of elliptic curve public key cryptography analyzed and the selection method of secure elliptic curve along with its implementation has been discussed by Qing-hai et al. in details.[157] Abdulkader et al. presented cryptography keys using self-organizing maps.[158] Santhanalakshmi et al. proposed a soft computing based approach for generating keys to design a stream cipher for text encryption.[159] Optimal weights for the sender and receiver used for the synchronization on the TPM neural network, are generated using a GA. A hybrid crypto system has been proposed by Gutub et al. which utilizes benefits of both symmetric key and public key cryptographic methods.[160] Symmetric key algorithms (DES and AES) were used in this crypto system to perform data encryption and Public key algorithm (RSA) was used in this crypto system to provide key encryption before key exchange. Combination of both the symmetric-key and public-key algorithms provides greater security and some unique features in that hybrid system. This system is not suitable for light weight devices having very low processing capabilities. A different cryptographic algorithm has been introduced by Shrivastava et al. which provides two phase security to the quantum cryptography system.[161] In this algorithm the presence of the eavesdropper will not affect the security of the system as the secret key bits are modified at both sender and receiver end based on the concept of prime factor. Paramanik et al. worked with the concept of massive parallelism and large information density inherent in DNA molecule are exploited for cryptographic purposes.[162] The main difficulties of DNA cryptography are the requirement of high tech biomolecular laboratory and computational complexity. In this paper, a new parallel cryptography technique is proposed using DNA molecular structure, one-time-pad scheme and DNA hybridization technique which certainly minimizes the time complexity. An implementation of the three-stage quantum communication protocol in free-space has been presented by Mandal et al. where multiple photons can be used for secure communication.[163] Verma et al. proposed enhanced version of RC6 Block cipher algorithm (RC6e - RC6 enhanced version), which is a symmetric encryption algorithm designed for 256 bit plain text block.[164] Yang et al. proposed that how to provide Stream cipher service in JCA, the implementations provide a so-called StreamCipherSpi abstract class for efficiently writing and maintaining any Stream cipher algorithm by developers.[165] Dolecki et al. proposed other schemes to evaluate compatibility of weights' vector.[166] The first one uses Euclidean distance of both weights' vector. The second one is based on frequencies of common TPM's outputs and as such does not rely on the weights' vector. Both approaches to handle secure key exchange protocol facilitate more extended analysis of many technical processes in which a vital role plays an incorporation of a non-standard high-quality method securing any sensitive data. Dolecki et al. uses the phenomenon of Neural Networks synchronization by mutual learning to construct key exchange protocol on an open channel.[167] The method presented permits evaluating the level of synchronization before it terminates. Subsequently, this research enables to assess the synchronizations, which are likely to be considered as long-time synchronizations. Once that occurs, it is preferable to launch synchronization with the new selected weights as there is a high probability that a new synchronization belongs to the short one. By taking an in-depth investigation on the security of Neural Cryptography, Mu et al. proposed a heuristic rule.[168] Aguilar et al. proposed an extended model of the random Neural Networks, whose architecture is multi-feedback.[169] In this case, they suppose different layers where the neurons have communication with the neurons of the neighbor layers. They present its learning algorithm and its possible utilizations; specifically, its use has been tested in an encryption mechanism where each layer is responsible of a part of the encryption or decryption process. It is striking to observe that after the first decade of Neural Cryptography, the TPM network with hidden unit $K = 3$ appears to be the sole network that is suitable for a neural protocol. No convincingly secure neural protocol is well designed by using other network structures despite considerable research efforts. With the goal of overcoming the limitations of a suitable network structure, Lei et al. develop a two-layer tree-connected feed-forward neural network (TTFNN) model for a neural protocol.[170] Three encryption algorithms namely DES, AES and Blowfish were analyzed by Ramesh et al. by considering certain performance metrics such as execution time, memory required for implementation and throughput.[171] A new symmetric key cryptographic method has been proposed by Sircar et al. using Modified generalized Vernam cipher method with feedback along with different block sizes.[172] A different image encryption technique has been presented by Soni et al. based on DNA sequence addition operation.[173] A different symmetric cryptographic technique has been developed which merged both RSA and Diffie-Hellman algorithms and a comparison has been conducted by Mandal et al. between the proposed technique, AES (Rijndael), DES, 3DES, RC2 and Blowfish.[174] Naveen et al. offers two different cryptographic schemes based on DNA binary strands are. In one of the approaches DNA based cryptography itself is used to encrypt and decrypt the message.[175] And in another approach DNA strands are used to generate key for encryption and decryption. Nakun et al. proposed generic framework is named as tree state classification machine (TSCM).[176] Allam et al. aimed to increase the security of the Neural Cryptography by authenticating the communication using preshared secrets.[177] The mutual learning algorithm is modified so that the reflecting boundaries become hidden and only accessible by the two partners. By making use of Artificial Intelligence (AI), Human Intelligence can be simulated by a machine, Neural Networks is one such sub field of AI. ANN consists of neurons and weights assigned to inter neuron connections helps in storing the acquired knowledge. Jhajharia et al. made use of Hebbian learning rule to train the ANN of both sender and receiver machines.[178] They proposesed key generation for PKC by application of ANN using GA. Allam et al. investigated the information leakage through the learning process.[179] This information can be used to reduce the complexity of the genetic attack, a Neural Cryptography known attack strategy. Akhavan et al. proposed a new efficient scheme for parallel hash function based on high-dimensional chaotic map.[180] In the proposed scheme, the confusion as well as the diffusion effect is enhanced significantly by utilizing two nonlinear coupling parameters. Singh et al. proposed Neural Cryptography for secret key exchange and encryption with AES.[181] Adel et al. presented a survey report on cryptography based on Neural Network.[182] Lonkar et al. in the year 2014 worked with cryptography using Neural Networks.[183] They formed the key by Neural Network is in the form of weights and neuronal functions. Apdullah et al. proposed non-linear encryption using relation-building functionality through Neural Network.[184] Mohammed Al-Maitah et al. proposed Neuro Cryptographic protocol based on a three-level Neural Network of the direct propagation.[185] There was evaluated it’s cryptosecurity and analyzed three types of this algorithm attack to show the reality of the hypothesis that Neuro Cryptography is currently one of the most promising post quantum cryptographic systems. Soni et al. described a scheme and claimed that any cryptographic system is used to exchange confidential information securely over the public channel without any leakage of information to the unauthorized users.[186] They proposed that Neural Networks can be used to generate a common secret key because the processes involve in cryptographic system requires large computational power and very complex. Two Neural Networks which are trained on their mutual output bits. The networks synchronize to a state with identical time dependent weights. Secret key exchange over a public channel and this key can be used in implementing any encryption algorithm. Dadhich et al. proposed a scheme for information communication, particularly text, image and video transmission.[187] Improvement of pictorial information for betterment of human perception involves de-blurring, de-noising and safe transmission. These applications extend over several fields such as satellite imaging, medical imaging etc. Specifically they would like to elaborate their research on the significance of computational intelligence as one of the domains which finds application in cryptography and information security, and then the relevance of cryptography is indeed unavoidable. This paper deals with the study of the requirements for strong cryptography and various computational intelligence techniques that find use in cryptography. Finally, they performed detailed comparison between cryptographic methods with computational intelligence and those cryptography techniques without computational intelligence. Singla et al. discussed about efficient random sequence generators which are used in the application areas of cryptographic stream cipher design, statistical sampling and simulation, direct spread spectrum, etc.[188] A cryptographically efficient pseudo-random sequence should have the characteristics of high randomness and encryption effect. The statistical quality of pseudo-random sequences determines the strength of cryptographic system. The generation of pseudo-random sequences with high randomness and encryption effect is a key challenge. A sequence with poor randomness threatens the security of cryptographic system. In this paper, the features and strengths of chaos and Neural Network are combined to design a pseudo-random binary sequence generator for cryptographic applications. The statistical performance of the chaotic neural network based pseudo random sequence generator is examined against the NIST SP800-22 randomness tests and multimedia image encryption. Soni et al. constructed a hash function based on multilayer feed forward network with piecewise linear chaotic map.[189] Chaos has been used in data protection because of the features of initial value sensitivity, random similarity and ergodicity. They have used three neuronal layers to prove confusion, diffusion and compression respectively. This hash function takes input of arbitrary length and generate a fixed length hash value (128 bit, 256 bit or 512 bit). Chakraborty et al. performed a survey on exchange of secret keys over public channels based on neural synchronization using a variety of learning rules offer an appealing alternative to number theory based cryptography algorithms.[190] Though several forms of attacks are possible on this neural protocol e.g. geometric, genetic and majority attacks, they found that deterministic algorithms that synchronize with the end-point networks have high time complexity, while probabilistic and population-based algorithms have demonstrated ability to decode the key during its exchange over the public channels. They also examined the queries, heuristics, erroneous information, group key exchange, synaptic depths, etc, that have been proposed to increase the time complexity of algorithmic interception or decoding of the key during exchange. They conclude that The TPM and its variants, Neural Networks with tree topologies incorporating parity checking of state bits, appear to be one of the most secure and stable models of the end-point networks. Our survey also mentions some noteworthy studies on Neural Networks applied to other necessary aspects of cryptography. They also claimed that discovery of neural architectures with very high synchronization speed, and designing the encoding and entropy of the information exchanged during mutual learning, and design of extremely sensitive chaotic maps for transformation of synchronized states of the networks to chaotic encryption keys, are the primary issues in this field. Adel et al. in the year 2014 proposed a public key cryptography system based on chaotic neural network (CNN) for encrypt and decrypt a digital image.[191] The most traditional public key cryptography is based on number theory which has some drawbacks such as large computational power, complexity, and time consumption. To overcome these drawbacks, a new chaotic Neural Network is introduced by the authors. They used multidimensional chaotic maps as a chaotic sequence for determined the Neural Network weight and basis through five layers of networks and additional layer for public key using Chebyshev chaotic map as a chaotic sequence for basis Neural Network.

Number of cryptographic techniques are proposed each of which has some advantages and disadvantages. There is no algorithm exists as universal solutions. So there is a dearth of searching new techniques as the scenario of computing world is changing continuously with a high rate of gradients.

1.6 Learning Rules for Tuning of Perceptron

Cryptographic session key can be generated through synchronization of two perceptrons one at sender another at receiver. At the perceptron synchronization phase weight vector of both perceptron is updated using perceptron learning rule to tuned the network. If the output bits are different for sender (A) and receiver (B) perceptrons i.e. $\tau^A \neq \tau^B$, nothing get changed. If $\tau^A = \tau^B = \tau$, only the weights of the hidden units with $\sigma_k^{A/B} = \tau^{A/B}$ will be updated. The weight vector of this hidden unit is adjusted using any of the following learning rules:

Anti-Hebbian: Both networks are trained with the opposite of their own output. This is achieved by using the Anti-Hebbian[98] learning rule given in equation 1.1.

$$W_k^{A/B} = W_k^{A/B} - \tau^{A/B} x_k \Theta(\sigma_k \tau^{A/B})(\tau^A \tau^B) \tag{1.1}$$

Hebbian: In the case of the Hebbian[98] learning rule both DHLPs learn from each other. The Hebbian rule given in equation 1.2.

$$W_k^{A/B} = W_k^{A/B} + \tau^{A/B} x_k \Theta(\sigma_k \tau^{A/B})(\tau^A \tau^B) \tag{1.2}$$

Random Walk: The set value of the output is not important for synchronization as long as it is the same for all participating DHLPs. That is why one can use the Random Walk[98] learning rule, too. The Random Walk rule given in equation 1.3.

$$W_k^{A/B} = W_k^{A/B} + x_k \Theta(\sigma_k \tau^{A/B})(\tau^A \tau^B) \tag{1.3}$$

Only weights are changed by these learning rules, which are in hidden units with $\sigma_i = \tau$. By doing so it is impossible to tell which weights are updated without knowing the internal representation $(\sigma_1, \sigma_2, \ldots, \sigma_k)$. This feature is especially needed for the cryptographic application of perceptron synchronization. Of course, the learning rules have to assure that the weights stay in the allowed range between $-L$ and $+L$. If any weight moves outside this region, it is reset to the nearest boundary value $\pm L$. Afterwards the current synchronization step is finished. This process can be repeated until corresponding weights in sender's and receiver's perceptrons have equal values, $W_i^A = W_i^B$. Further applications of the learning rule are unable to destroy this synchronization, because the movements of the weights depend only on the inputs and weights, which are then identical in sender's and receiver's perceptrons.

1.7 Metrics for Evaluation

An indicator conform the evidence that a particular condition exists or certain results have or have not been achieved. It can be either quantitative or qualitative. A metric refers to a unit of measurement that is quantitative. Several kinds of metrics have been used for evaluating the quality of the proposed cryptographic systems. The measures are NIST statistical test, performance test, encryption and decryption time, Avalanche and Strict Avalanche effects, Bit Independence criterion, Chi-Square test, frequency distribution, entropy, floating frequency and autocorrelation which are described in section 1.71 to section 1.7.10 respectively.

1.7.1 NIST Statistical Test

A total of fifteen statistical tests recommended in the NIST test[192] Suite to evaluate randomness of the synchronized session key proposed in different chapters. These tests focused on a variety of different types of non-randomness that could exist in a sequence. Some tests are decomposable into a variety of subtests. The fifteen tests are following:

- *Frequency (Monobits) Test* - The purpose of this test is to determine whether that number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence. The test assesses the closeness of the fraction of ones to ½, that is, the number of ones and zeroes in a sequence should be about the same.
- *Test for Frequency within a Block* - The focus of the test is to find the proportion of zeroes and ones within M-bit blocks. The purpose of this test is to determine whether the frequency of ones is an M-bit block is approximately $\frac{M}{2}$
- *Runs Test* - The focus of this test is the total number of zero and one runs in the entire sequence, where a run is an uninterrupted sequence of identical bits. A run of length k

means that a run consists of exactly k identical bits and is bounded before and after with a bit of the opposite value. The purpose of the runs test is to determine whether the number of runs of ones and zeros of various lengths is as expected for a random sequence. In particular, this test determines whether the oscillation between such substrings is too fast or too slow.

- *Longest Run of Ones in a Block* - The focus of the test is to find the longest run of ones within M-bit blocks. The purpose of this test is to determine whether the length of the longest run of ones within the tested sequence is consistent with the length of the longest run of ones that would be expected in a random sequence. Note that an irregularity in the expected length of the longest run of ones implies that there is also an irregularity in the expected length of the longest run of zeroes. Long runs of zeroes were not evaluated separately due to a concern about statistical independence among the tests.
- *Binary Matrix Rank Test* - The focus of the test is the rank of disjoint sub-matrices of the entire sequence. The purpose of this test is to check for linear dependence among fixed length substrings of the original sequence.
- *Discrete Fourier Transform Test* - The focus of this test is the peak heights in the discrete Fast Fourier Transform. The purpose of this test is to detect periodic features (i.e., repetitive patterns that are near each other) in the tested sequence that would indicate a deviation from the assumption of randomness.
- *Non-overlapping (Aperiodic) Template Matching Test* - The purpose of this test is to reject sequences that exhibit too many occurrences of a given non-periodic (aperiodic) pattern. For this test and for the Overlapping Template Matching test, an m-bit window is used to search for a specific m-bit pattern. If the pattern is not found, the window slides one bit position. For this test, when the pattern is found, the window is reset to the bit after the found pattern, and the search resumes.
- *Overlapping (Periodic) Template Matching Test* - The purpose of this test is to reject sequences that show deviations from the expected number of runs of ones of a given length. Note that when there is a deviation from the expected number of ones of a

given length, there is also a deviation in the runs of zeroes. Runs of zeroes were not evaluated separately due to a concern about statistical independence among the tests. For this test and for the Non-overlapping Template Matching test, an m-bit window is used to search for a specific m-bit pattern. If the pattern is not found, the window slides one bit position. For this test, when the pattern is found, the window again slides one bit, and the search is resumed.

- *Maurer's "Universal Statistical" Test* - The purpose of the test is to detect whether or not the sequence can be significantly compressed without loss of information. An overly compressible sequence is considered to be non-random.
- *Linear Complexity Test* - The focus of this test is to find the length of a generating feedback register. The purpose of this test is to determine whether or not the sequence is complex enough to be considered random. Random sequences are characterized by a longer feedback register. A short feedback register implies non-randomness.
- *Serial Test* - The focus of this test is to obtain the frequency of each and every overlapping m-bit pattern across the entire sequence. The purpose of this test is to determine whether the number of occurrences of the 2^m m-bit overlapping patterns is approximately the same as would be expected for a random sequence. The pattern can overlap.
- *Appoximate Entropy Test* - The focus of this test is to obtain the frequency of each and every overlapping m-bit pattern. The purpose of the test is to compare the frequency of overlapping blocks of two consecutive/adjacent lengths (m and $m + 1$) against the expected result for a random sequence.
- *Cumulative Sums Test* - The focus of this test is the maximal excursion (from zero) of the random walk defined by the cumulative sum of adjusted $(-1, +1)$ digits in the sequence. The purpose of the test is to determine whether the cumulative sum of the partial sequences occurring in the tested sequence is too large or too small relative to the expected behavior of that cumulative sum for random sequences. This cumulative sum may be considered as a random walk. For a random sequence, the random walk

should be near zero. For non-random sequences, the excursions of this random walk away from zero will be too large.

- *Random Excursions Test* - The focus of this test is to find the number of cycles having exactly K visits in a cumulative sum random walk. The cumulative sum random walk is found if partial sums of the $(0,1)$ sequence are adjusted to $(-1,+1)$. A random excursion of a random walk consists of a sequence of n steps of unit length taken at random that begin at and return to the origin. The purpose of this test is to determine if the number of visits to a state within a random walk exceeds what one would expect for a random sequence.

- *Random Excursions Variant Test* - The focus of this test is to find the number of times that a particular state occurs in a cumulative sum random walk. The purpose of this test is to detect deviations from the expected number of occurrences of various states in the random walk.

1.7.2 Performance Analysis

In performance testing performance of all the proposed and existing techniques are compared with each other in terms of average synchronization time for generation of session key and grouped session key of $128/192/256$ bit using fixed weight range and different number of neurons in input and hidden layer, different weight range and fixed number of neurons in input and hidden layer, amount of heap used for generating 128 bit session key, amount of relative time spent in GC used for generating 128 bit session key, amount of thread required for generating 128 bit session key, number of generation vs. average fitness value in SA and GA and key storage comparisons.

1.7.3 Encryption and Decryption Time

All the test programs for the proposed techniques are equipped to calculate and display total encryption time and decryption time at the end of execution. Time taken is the difference between processor clock ticks between the starting and end of the algorithm. All times are measured in milliseconds (ms). The lower processing time means the higher speed which sometimes better for a typical end user. Since the CPU clock ticks are taken as time, there might be a slight variation with actual time. This variation is very insignificant and may be ignored.

1.7.4 Avalanche and Strict Avalanche Effects

In cryptography, the Avalanche Effect (AVAL) is a desirable property of block ciphers. Avalanche effect means that a very small number of bit changes in the plaintext will lead to a very large number of bit changes in the cipher text. In case of high quality block ciphers, a small change in either the key or the plaintext should cause a drastic change in the cipher text. The actual term was first used by Horst Feistel in 1973.[1] More formally, a function $f : \{0,1\}^n \to \{0,1\}^n$ satisfies AVAL if whenever one input bit is changed, on the average half of the output bits change, where i and $j \in (1, 2, 3, \ldots\ldots, n)$ are input and output bits respectively.

The Strict Avalanche Effect (SAE) is a generalized of the avalanche effect. SAE is said to be satisfy if, whenever a single input bit is complemented, each of the output bits changes with a 50% probability. It builds on the combined concept of completeness and avalanche effect. It was first introduced by Webster and Tavares in 1985.[193] A function $f : \{0,1\}^n \to \{0,1\}^n$ satisfies SAE if for all i and $j \in (1, 2, 3, \ldots\ldots, n)$, flipping input bit i changes the output bit j with the probability of exactly one half. In 1990, the notion of strict avalanche criterion was extended by R. Forre. He considered sub-functions obtained from the original function by keeping one or more input bits constant.

1.7.5 Bit Independence Criterion

In 1986, Webster and Tavares introduced another cryptographic property Bit Independence Criterion (BIC) for s-boxes.[194] A function $f : \{0,1\}^n \to \{0,1\}^n$ satisfies BIC if for all $i, j, k \in \{1, 2, 3, \ldots\ldots, n\}$, with $j \neq k$, inverting input bit i causes output bits j and k to change independently. To measure BIC, the correlation coefficient between j'th and k'th components of the output difference string is needed, which is called the Avalanche vector A^{e_i}.

1.7.6 Chi-Square Test

Chi-Square value is calculated from the character frequencies using the equation 1.4 devised by Karl Pearson:[194]

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \tag{1.4}$$

Where,

O_i (Occurred) is the frequency of occurrence of character i in the encrypted message

E_i (Expected) is the frequency of occurrence of character i in the original message

Chi-Square test is used to determine whether the observed sample frequencies differ significantly from the expected frequencies. The higher the Chi-Square values the more deviation from the original message. The large Chi-Square values confirm the heterogeneity of the source file and the encrypted file. Larger Chi-Square value compare to tabulated Chi-Square value ensure the higher degree of heterogeneity.

1.7.7 Frequency Distribution

Frequency distribution analyzes both the original and encrypted files. The occurrence of each character on both the files is measured. Graphs are generated where ASCII value of each character plotted along X-axis and frequency or number of occurrences of characters along Y-axis. The smoother curve in the spectrum of frequency distribution indicates that it is harder for a cryptanalyst to detect the original message bytes.

1.7.8 Entropy

The entropy of a document is an index of its information content. The entropy is measured in bits per character. If a character has a very high probability of occurrence, then its information content is low. For documents which can contain every character of the character set (0 to 255) the entropy lies between 0 bit/char (in a document which consists of only one character) and $\log(256)$ bit/char = 8 bit/char (in a document in which all 256 characters occur equally often).

1.7.9 Floating Frequency

The floating frequency of a document is a characteristic of its local information content at individual points in the document. The floating frequency specifies how many different characters are to be found in any given 64 character long segment of the document. The function considers sequences of text in the active window that are 64 characters long and counts how many different characters are to be found in this "window". The "window" is then shifted one character to the right and the calculation is repeated. This procedure results in a summary of the document in which it is possible to identify the places with high and low information density. A document of length $\mathrm{n} > 64$ bytes has $(\mathrm{n} - 63)$ such index numbers in its characteristics.

1.7.10 Autocorrelation

The purpose of this empirical test of independence is to check correlations between succeeding outcomes of the encryption and/or between the binary sequence s and an alternative version of s that is displaced by t positions. Let t be a number, $1 \leq t \leq (n / 2)$ and fixed.

1.8 Objectives

The objective of modern cryptographic technique is to provide security for the system where unify computing is an essential component and also for light weight devices having very low processing capabilities or limited computing power in wireless communication.

In the present scenario, existing cryptographic technique depend on the exchange of keys through insecure public channel which are used to encrypt and decrypt the information exchange. This is vulnerable in terms of security. Using these key sender and receiver perform reasonably complex mathematical operations on the data stream. This is also takes significant amount of resources. So it is essential to find some cryptographic techniques where instead of transmitting the whole key through insecure public channel, session key can be generated at both sides using mutual synchronization of both parties. By keeping in mind the resource constrains criteria of wireless communication the robust and secure encryption/decryption technique which takes less resources for computations and secure session key which is less complex but provides very high degree of security with respect to existing cryptographic techniques along with energy awareness is very much needed in wireless communication.

So it is essential for modern day users to secure their communication in terms of security as well as energy awareness.

The objectives of this thesis are to

- *enhance the security of the wireless communication system in such a way that the instead of exchanging the whole session key, soft computing based synchronization technique is used to construct a cryptographic key-exchange protocol for generating the identical session key at sender and receiver.*
- *develop and implement cryptographic techniques which are very simple, and easy to implement but provide good security and can be implemented.*
- *compare the proposed techniques with the existing and industrially accepted techniques with respect to parameters like NIST statistical test, performance test, encryption and decryption time, Avalanche and Strict Avalanche effects, Bit Independence criterion, Chi-Square test, frequency distribution, entropy, floating frequency and autocorrelation.*

- *trade-off between security and performance of light weight devices having very low processing capabilities or limited computing power*

1.9 Organization of the Thesis

The thesis consists of eight chapters. Chapter 1 contains the introductory discussion of the problem and solution domain. An introductory interface about cryptography and soft computing based techniques, literature survey, objective and organization of the thesis, learning rules for tuning of perceptrons, metrics for evaluation and salient features of the proposed techniques have been discussed briefly.

Chapter 2 of this thesis deals with Kohonen's Self-Organizing Feature Map Synchronized Cryptographic Technique (KSOMSCT). Security analysis and discussions about the proposed technique has been done.

In Chapter 3, a Double Hidden Layer Perceptron Synchronized Cryptographic Technique (DHLPSCT) has been proposed. Security analysis and discussions about the proposed technique has been done.

In Chapter 4, a Chaos based Double Hidden Layer Perceptron Synchronized Cryptographic Technique (CDHLPSCT) has been proposed. Security analysis and discussions about the proposed technique has been done.

In Chapter 5, a Chaos based Triple Hidden Layer Perceptron Synchronized Cryptographic Technique (CTHLPSCT) has been proposed. Security analysis and discussions about the proposed technique has been done.

In Chapter 6, a Chaos based Grouped Triple Hidden Layer Perceptron Synchronized Cryptographic Technique (CGTHLPSCT) has been proposed. Security analysis and discussions about the proposed technique has been done.

Chapter 7 deals with results and analysis of the proposed techniques. Comparison has been done among proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing Tree Parity Machine (TPM) and Permutation Parity Machine (PPM), RSA, Triple-DES (168 bits), AES (128 bits), RC4 and Vernam Cipher for their relative performances.

A model has been proposed through cascaded implementation of the devised cryptographic techniques of this thesis, in chapter 8. At the end list of references is given.

1.10 Salient Features of the Proposed Techniques

In this thesis, the logic of the proposed soft computing based cryptographic techniques in wireless communication is simple to understand and implementation is easy using any high level programming language. Since keys are session based which varies session to session and key size is variable in length, the security of the proposed techniques is good. The strength of the proposed techniques is the adoption of complexity based on energy and resource available in the wireless communication, infrastructure for computing in a node or mesh in wireless communication. For a wireless network having low energy, the number of cascading stages and iteration be less. Also during the synchronization phase the different structures of the proposed perceptrons can be used depending on the available resources in the wireless communication. So, the proposed techniques are very much suitable for the security of the system where energy and resource is one of the main constraints. All the proposed techniques can handle any sort of input file of any size. There is no alteration of input file size i.e. after encryption file size remains unchanged. The salient features of all the proposed techniques are summarized as follows:

- *Generation of session key through synchronization*
- *No exchange of session key through public channel*
- *High degree of security*
- *Variable in length keys*
- *Independency of file types*
- *Size independency of source file*
- *Offers variable block size*
- *No space overhead*
- *Logics are simple to understand*
- *Less complex*
- *Easy to implement the algorithms*

Chapter 2

Kohonen's Self-Organizing Map Synchronized Cryptographic Technique (KSOMSCT)

2.1 Introduction

In this chapter a novel soft computing assisted cryptographic technique KSOMSCT based on synchronization of two Kohonen's Self-Organizing Feature Map (KSOFM)[195], one at sender and another at receiver has been proposed. In public-key cryptography key generation and key exchange are one of the major issues. Eavesdroppers can reside between sender and receiver and tries to capture all the information transmitting between the parties. So, at the time of key exchange between sender and receiver intruders can perform sniffing, spoofing or phishing operation to tamper the key. Another noticeable problem is that most of the key generation algorithms need large amounts of memory space for generating the session key but now-a-days most of the handheld wireless devices have a criterion of memory and resource constraints.

For the solution of the problem KSOFM based synchronization has been proposed to address this issue by resolving the drawbacks of some of the existing cryptographic approach.[196]

Here, KSOFM based synchronization is performed for tuning both sender and receiver simultaneously. On completion of the tuning phase identical session key generates at the both end using synchronized KSOFM. This synchronized network can be used for transmitting message using any light weight encryption/decryption techniques with the help of identical session key of the synchronized network. To illustrate the cryptographic technique in wireless communication one of the simple and secure encryption/decryption technique has been presented. A plaintext is considered as a stream of binary bits. Fractal triangle based encryption[197] is performed with the help of KSOFM tuned session key to generate the cipher text. The plaintext is regenerated at the destination by performing Fractal triangle based decryption with the help of KSOFM tuned session key.

Section 2.2 presents a description of proposed technique. Section 2.3 deals with the implementation of the proposed cryptographic technique. Section 2.4 discussed the security issue related to the proposed technique. Discussions are presented in section 2.5.

2.2 The Technique

The technique performs the KSOFM based synchronization for generation of secret session key at both ends. This synchronized session key of the tuned network is used for the transmission of secured message through wireless network with the help of any light weight encryption/decryption algorithm. To illustrate the cryptographic technique in wireless communication one of the simple and secure encryption/decryption technique has been proposed, where plaintext (i.e. the input file) is considered as a stream of binary bits, which is encrypted using Fractal triangle based encryption technique. The session key based on KSOFM is used to further encrypt the Fractal triangle encoded text to produce final cipher text. In this technique instead of exchanging the whole session key to the receiver using public channel KSOFM index parameters are exchanged. The technique has an ability to construct the unique secret session key at both ends using exchanged information. For ensuring the randomness in every session, certain parameter values get randomly changed like seed values for generating random inputs and weights, number of iteration to train the map, different mathematical functions (Radial basis, Gaussian, Mexican Hat) for choosing the random points from the KSOFM.

In Fractal triangle based encryption/decryption technique the number of dimensions of Fractal triangle used in the encryption/decryption process key size has been determined. Key for Fractal triangle based encryption is formed from the KSOFM based synchronized session key. The key size may also larger than available number of bits in the synchronized session key. The extra bits require is taken after performing four bits circular right shift operation on the KSOFM based synchronized session key. Finally, a cascaded *Exclusive-OR* operation is performed between Fractal triangle encrypted blocks with the KSOFM based session key to generate final cipher text.

Receiver has same KSOFM synchronized session key as a result of tuning. This session key used to perform first step of the deciphering. In the next step, Fractal triangle based deciphering operation is performed to regenerate the plaintext.

The technique does not produce any storage overhead. This technique needs a minimum amount of storage for storing the key which greatly handles the resource constraints criteria of wireless communication. The implementation on practical scenario is well proven with positive outcome. A comparison of KSOMSCT with existing Tree Parity Machine (TPM) and Permutation Parity Machine (PPM) based key exchange techniques and industry accepted AES, RC4, Vernam Cipher, Triple DES (TDES) and RSA have been done. Details of results along with analysis are given in chapter 7.

In KSOMSCT, synchronization operation on both sender's and receiver's KSOFM system is performed for generating common session key. The Fractal triangle based encryption algorithm takes the plaintext as a binary stream of bits which is encrypted using Fractal triangle based encryption technique. The key size is determined depending on the dimensions of Fractal triangle used in the encryption process. The Fractal dimension is calculated using equation 2.1

$$n = \frac{ln\ s}{ln\,((m-1)^2-1)} + 1 \tag{2.1}$$

Where $m =$ edges numbers and $s\ =$ sub-triangles numbers

If Fractal triangle $dimension\ (n) = 3$ then first four bits of the synchronized session key becomes the encryption key, if $n = 4$ then first thirteen bits are taken from synchronized session key. If encryption key size is greater than available number of bits in the synchronized session key then rest of the required bits can be taken from left to right by performing the four bits circular right shift operation on the synchronized session key. Mandlbrot Set equation is used to form the Fractal triangle, which is given in equation 2.2.

$$Z_{k+1} = Z_{k^2} + C\ (Z_0 = 0) \tag{2.2}$$

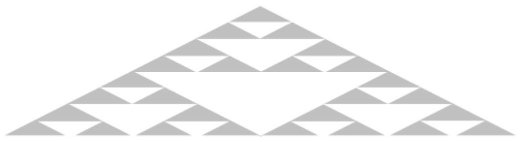


Figure 2.1: The Sierpinski triangle

Fractal triangle has been used to perform encryption technique by placing the source bits (plaintext) into the each vertex of each triangle in sequence and placing the key bits for encryption into the middle of each triangle. Then the encryption operation is performed to generate the Fractal triangle encrypted text. Fractal triangle encoded text is encrypted further using *Exclusive-OR* operation with the session key. The algorithm for the complete process is given in section 2.2.1.

2.2.1 KSOMSCT Algorithm at Sender

Input : Source file/source stream i.e. plaintext

Output : Encrypted file/encrypted stream i.e. cipher text

Method : The process operates on binary stream and generates encrypted bit stream through Kohonen's Self-Organizing Feature Map (KSOFM) and Fractal triangle based encryption.

Step 1. Perform synchronization operation on both sender's and receiver's KSOFM system to generate tuned common session key.

Step 2. Perform Fractal triangle based encryption technique to generate the intermediate cipher text.

Step 3. Perform cascaded Exclusive-OR operations between KSOFM based synchronized session key and intermediate encrypted text generated in step 2 to form the final cipher text.

Step 1 of the algorithm for generating common tuned session key through synchronization of sender's and receiver's KSOFM system is discussed in section 2.2.1.1. Step 2 of the algorithm for performing Fractal triangle based encryption is discussed in 2.2.1.2. Step 3 of the algorithm is discussed in 2.2.1.3.

2.2.1.1 Kohonen Self-Organizing Feature Map (KSOFM) based Synchronization

In this section, a novel Kohonen Self-Organizing Feature Map (KSOFM)[195] based synchronization of both sender and receiver machine has been proposed. The tuned network is used for message communication purpose based on tuned parameter. The technique imparts a simple and secure way of key generation both sender and receiver simultaneously using KSOFM based tuning. Unsupervised competitive learning is used for synchronization. The method uses unsupervised learning to represent input space of the training samples in a discrete $2D$ maps. Neighborhood of each neuron (i.e. the connections of the neuron with adjacent neurons) in the map depends on the dimension of the map. $2D$ regular spacing in a hexagonal or rectangular grid uses to arrange the neurons. Detailed methodology used in KSOFM based synchronization is discussed as follows.

The KSOFM comprises of neurons along with a weight vector for each neuron having a dimension same as the dimension of the input vector. Consider the input vector $X = [x_1, x_2, \ldots, x_n]^T$and weight vector $W = [w_1, w_2, \ldots, w_n]^T$. The process initially, assigns a weight vector to each neuron (point) by arbitrarily choosing a neuron (point) $x \in$ *input space* X. The value of the weight vector is set to a tiny random numbers.

The necessity of unsupervised learning mechanism in KSOFM is to produce similar response from different parts of the network for a certain input patterns. Competitive learning is used in the training period to train the KSOFM. Euclidean distance between each neuron and an arbitrary neuron (point) x get calculated using equation 2.3

$$Distance_k = \sqrt{(x_1 - w_{k1})^2 + (x_2 - w_{k2})^2 + \cdots + (x_n - w_{kn})^2} \tag{2.3}$$

Where,

$k = 1,2, \ldots, P$

P is the neuron number

W_{kj} is the entry of j of the weight of neuron k where $j = 1,2, \ldots, n$

The neuron (point) whose weight vector is most similar to the input is called the Best Matching Unit (BMU). The weights of the BMU and neurons (point) close to it in the KSOFM lattice are adjusted towards the input vector. The magnitude of the change decreases with time and with distance (within the lattice) from the BMU.

The technique uses $2D$ KSOFM with 100 neurons. A learning rate α of 0.1 is used to train the KSOFM and decreasing the spread of the neighborhood function by the rule given in equation 2.4

$$\sigma = \sigma_0 \left(1 - \frac{Iteration\ \_number}{Total\ \_no\ \_of\ \_iteration}\right) \tag{2.4}$$

Here, σ is initial spread. The value of σ decreases from the initial value to the final value (0) constantly. Hence, the neighborhood function influences all neurons of the map in the first time and its influence on far neurons vanishes progressively. Towards the end of the training only the winner neuron will be updated so as to drive neurons towards centers of gravity.

The winner neuron is a neuron which has a minimum distance from x (arbitrary point). Winner neuron gets selected based on the distance factor. The minimum distance $Distance_{Winner}$ fulfills the condition given in equation 2.5.

$$Distance_{Winner} \leq Distance_k \tag{2.5}$$

$$Where\ k = 1,2,\dots,P$$

An updating rule has been applied over the entire map keeping in mind the priority of the winner neuron and its closest neighbors. In existing KSOFM algorithm[198] the general update formula[199] for a neuron with weight vector $W_v(s)$ is given in equation 2.6

$$W_v(s+1) = W_v(s) + \Theta(u,v,s)\,\alpha(s)(D(t) - W_v(s)) \tag{2.6}$$

Where,

S is the old iteration

$S+1$ is the new iteration

t is the index of the target input data vector in the input data set D

$D(t)$ is a target input data vector

v is the index of the node in the map

W_v is the current weight vector of node v

u is the index of the Best Matching Unit (BMU) in the map

$\Theta(u,v,s)$ is a restraint due to distance from BMU, usually called the neighborhood function, and

$\alpha(s)$ is a learning restraint due to iteration progress

The existing generalized KSOFM updating equation 2.6 is expressed using equation 2.7 with the parameters of the proposed technique.

$$w_{k,new} = w_{k,old} + \alpha.Neighbor(winner,k).\left(x - w_{k,old}\right) \tag{2.7}$$

Where, old iteration (S) is denoted by old. New iteration $(S+1)$ is denoted by new, index of the node in the map (v) is denoted by k, current weight vector (W_v) is denoted by $W_{k,}$ learning restraint $\alpha(s)$ is denoted by α, index of the Best Matching Unit (BMU) in the map (u) is denoted by $winner$, neighborhood function $\Theta(u,v,s)$ is denoted by $Neighbor(winner,k)$ with a bell shape centered at the winner neuron. It is a function of the distance between the winner neuron and the neuron k. $Neighbor\,(winner, k) = e^{\frac{|winne\ \ r-k|^2}{\sigma^2}}$ and target input data vector $D(t)$ is denoted by x.

A new arbitrary point $y \in input\ space\ X$ get selected and starting from the step unsupervised training of KSOFM to the step updating the network get perform again. This process is repeated for each input vector for a (usually large) number of cycles λ.

The spreading of the neighborhood function (σ) is important since it controls the convergence of the map. It should be large at the beginning and shrink progressively to reach a small value in order to globally order the neurons over the whole map. The maximum value of $Neighbor(winner, winner) = 1$ corresponds to the winner neuron and value of *Neighbor function* decreases when the distance between neurons k and *winner* increases. Concerning the value of the learning rate σ, it should be small enough to ensure the convergence of the KSOFM.

During synchronization both sender and receiver use the identical KSOFM architecture along with identical parameters in each session. Parameters used in each session are:

- Dimension of the KSOFM ($2D$ or $3D$)
- Number of neurons which specifies the number of different possible session keys
- Dimension of the weight vector specify the length of the key
- Seed value for generating random inputs and weights
- Number of iteration to train the map
- Different mathematical functions as a mask for choosing the random points from the KSOFM (Radial basis, Mexican Hat, Gaussian etc.)
- Different index value for choosing different neurons (key) on the mathematical mask at each session for forming the session key

Parameters that get negotiated at the initial stage of synchronization process between sender and receiver by mutual agreement are completely random. Changing each of the parameters randomly in each session security of proposed technique can be enhanced which in turns decrease the success rate of the attackers.

Both sender's and receiver's KSOFM are starts synchronization by exchanging control frames for negotiation of parameters value. KSOFM based synchronization uses transmission of control frames given in table 2.1 at the time of three way handshaking based TCP connection establishment phase.

Table 2.1
Control frames of KSOFM synchronization

Frame	**Description**
SYN	SYN frame transmitted to the receiver for synchronization in connection establishment phase
ACK_SYN	ACK_SYN frame transmitted to the sender for positive acknowledgement respect to SYN frame
NAK_SYN	NAK_SYN frame transmitted to the sender for negative acknowledgement respect to SYN frame
FIN_SYN	FIN_SYN frame transmitted by either party for closing the connection

The SYN frame is used for establishing the connection to the other side. It carries index information of different initial parameters. The detailed format of SYN frame is given in section 2.2.1.1.1. The proposed ACK_SYN frame is used for providing the positive acknowledgement with respect to the SYN frame. The detailed format of ACK_SYN frame is discussed in section 2.2.1.1.2. The proposed NAK_SYN frame is used for providing the negative acknowledgement with respect to the SYN frame. The detailed format of NAK_SYN frame is discussed in section 2.2.1.1.3. The proposed FIN_SYN frame is used for closing the connection. Either side can generate the request of closing connection through FIN_SYN frame. The detailed format of FIN_SYN frame is discussed in section 2.2.1.1.4.

2.2.1.1.1 Synchronization (SYN) Frame

During synchronization process sender constructs a SYN frame and transmit to the receiver for handshaking in connection establishment phase. SYN usually comprises of several fields these are $Command\ Code, SYN\ ID, DIM\ Index, Weight\ DIM\ Index, Iteration\ Index$, $Mask\ Index, Seed\ Index, Neuron\ DIM\ Index, CRC$. Figure 2.2 shows the complete format of SYN frame.

$Command$ $Code$ 00	$SYN\ ID$	DIM $Index$	$Weight$ DIM $Index$	$Iteration$ $Index$	$Mask$ $Index$	$Seed$ $Index$	$Neuron$ DIM $Index$	CRC $(Cyclic$ $Redundancy$ $Checker)$
2	4	1	2	16	2	4	4	16 $(bits)$

Figure 2.2: Frame format of SYN frame

Table 2.2 shows different $Command\ Code$ against different frames. SYN frame has a unique two bits $Command\ Code$ 00, ACK_SYN has the $Command\ Code$ 01. Whereas NAK_SYN uses 10 as its $Command\ Code$ and finally $Command\ Code$ 11 is for FIN_SYN frame.

Table 2.2
KSOFM control frames and their command codes

Command code	**Frame**
00	SYN
01	ACK_SYN
10	NAK_SYN
11	FIN_SYN

Four bits $SYN\ ID$ is used to identify different SYN frame in different session. One bit is used to specify the dimension of KSOFM using $DIM\ Index$. Table 2.3 gives the $DIM\ Index$ corresponds to the dimension of KSOFM.

Table 2.3
DIM Index corresponds to the dimension of KSOFM

DIM Index	**KSOFM Dimension**
0	$2D$
1	$3D$

Two bits are used to illustrate the $Weight\ DIM\ Index,$ where four different weights are available as shown in table 2.4.

Table 2.4
Weight DIM Index corresponds to the number of weights

Weight DIM Index	**Number of weights**
00	64
01	128
10	192
11	256

Sixteen bits are used to illustrate the $Iteration\ Index$. Two bits are used to illustrate the $Mask\ Index$. Table 2.5 illustrate the different $Mask\ Index$ value corresponds to the different mathematical mask functions.

Table 2.5
Mask Index value corresponds to the different mathematical mask functions

Mask Index	**Mathematical Mask Function**
00	$Mexican\ Hat$
01	$Gaussian$
10	$Radial\ Basis$
11	$Reserved$

Four bits are used to illustrate the $Seed\ Index$ and four bits are used to illustrate the $Neuron\ DIM\ Index$ which is the total number of neurons. Sixteen bits are used in CRC. When the receiver receives the frame SYN, the receiver carries out integrity test. Receiver also performs integrity test after receiving the SYN frame. If the messages are received as sent (with no replication, incorporation, alteration, reordering, or replay) the receiver will execute the synchronization phase.

2.2.1.1.2 Synchronization (ACK_SYN) Acknowledgement Frame

ACK_SYN frame send by the receiver to the sender in respect of ACK frame for positive acknowledgement of the parameters value. This proposed frame comprises of $Command\ Code, SYN\ ID, CRC$. $Command\ Code$ needs two bits. The ACK_SYN frame has the fixed $Command\ Code$ i.e. 10. Four bits are used for representing the $SYN\ ID$ and CRC needs sixteen bits for error checking purpose. Figure 2.3 shows the complete frame format of ACK_SYN frame.

$Command\ Code$ 10	$SYN\ ID$	CRC ($Cyclic\ Redundancy\ \ Checker$)
2	4	16 ($bits$)

Figure 2.3: Acknowledgement of Synchronization (ACK_SYN) frame

2.2.1.1.3 Negative Acknowledgement (NAK_SYN) Frame of Synchronization

NAK_SYN frame send by the receiver to the sender in respect of ACK frame for negative acknowledgement. This frame comprises of three fields, $Command\ Code, SYN\ ID, CRC$. $Command\ Code$ needs two bits. The ACK_SYN frame has the fixed $Command\ Code$ i.e. 11. Four bits is used for representing the $SYN\ ID$ and CRC needs sixteen bits for error checking purpose. Figure 2.4 shows the complete frame format of NAK_SYN frame.

$Command\ Code$ 11	$SYN\ ID$	CRC ($Cyclic\ Redundancy\ \ Checker$)
2	4	16 *(bits)*

Figure 2.4: Negative Acknowledgement of Synchronization (NAK_SYN) frame

2.2.1.1.4 Finish Synchronization (FIN_SYN) Frame

FIN_SYN frame send by the either party for finish the synchronization process. This proposed frame comprises of $Command\ Code, SYN\ ID, Index, CRC. Command\ Code$ needs two bits. The FIN_SYN frame has the fixed $Command\ Code$ i.e. 01. Four bits are used for representing $SYN\ ID$. Two bits are used for providing index value of the neuron (key) on the mathematical mask and CRC needs sixteen bits for error checking purpose. Figure 2.5 shows the complete frame format of FIN_SYN frame.

$Command\ Code$ 01	$SYN\ ID$	$Index$	CRC ($Cyclic\ Redundancy\ \ Checker$)
2	4	4	16 ($bits$)

Figure 2.5: Finish Synchronization (FIN_SYN) frame

The KSOFM synchronization algorithm for generating synchronized session key is discussed in section 2.2.1.1.5. Section 2.2.1.1.6 presents the complexity analysis of the KSOFM synchronization algorithm and KSOFM based session key generation methodology is discussed in section 2.2.1.1.7.

2.2.1.1.5 KSOFM Synchronization

Input : *Assign a weight vector to each neuron by arbitrarily choosing a point of the input space*

Output : *Synchronized KSOFM*

Method : *The process operates on sender's and receiver's Kohonen's Self-Organizing Feature Map (KSOFM) and generate synchronized session key.*

```
Step 1.   Randomize the map's nodes' weight vector.
Step 2.   Select an arbitrary input vector.
Step 3.   Traverse each node in the map.
          Step 3.1  Use the Euclidean distance formula to find the similarity
                    between the input vector and the map's node's weight vector.
          Step 3.2  Track the node that produces the smallest distance (this
                    node is the Best Matching Unit, BMU).
Step 4.   Update the nodes in the neighborhood of the BMU (including the
          BMU itself) by pulling them closer to the input vector using equation
          2.8.
          $Wv(s + 1) = Wv(s) + \Theta(u,v,s)\,\alpha(s)(D(t) - Wv(s))$     (2.8)
          Where,
              $S$ is the current iteration
              $\lambda$ is the iteration limit
              $t$ is the index of the target input data vector in the input data set $D$
              $D(t)$ is a target input data vector
              $v$ is the index of the node in the map
              $w_v$ is the current weight vector of node $v$
              $u$ is the index of the Best Matching Unit (BMU) in the map
```

$\Theta(u, v, s)$ is a restraint due to distance from BMU, usually called the neighborhood function, and

$\alpha(s)$ is a learning restraint due to iteration progress.

Step 5. *Increase s and repeat from step 2.*

2.2.1.1.6 Complexity Analysis

For assigning random weight vector to the map's, nodes needs $O(number\ of\ nodes)$ computations. Selection of any arbitrary point needs unit amount of time. Traversing each node in the map and then using the Euclidean distance formula to find the similarity between the input vector and the map's node's weight vector needs $O(number\ of\ nodes \times operation\ in\ Euclidean\ distance\ calculation)$ computations. Tracking of the node that produces the smallest distance (BMU) needs unit amount of time. Updating the nodes in the neighborhood of the BMU (including the BMU itself) by pulling them closer to the input vector needs $O(number\ of\ updated\ nodes)$ computations. So, each iteration of the algorithm needs $O(number\ of\ nodes\ +\ (number\ of\ nodes\ \times\ operation\ in\ Euclidean\ distance\ calculation)\ +\ number\ of\ updated\ nodes)$ amount of computations. This is the best case situation where a single iteration is needed to synchronize both the KSOFM networks. If the algorithm iterate n number of times then in average and worst case $O(n \times (number\ of\ nodes +\ (number\ of\ nodes \times operation\ in\ Euclidean\ distance\ calculation) +\ number\ of\ updated\ nodes))$ amount of computations is needed. Because each iteration eliminates the nodes which has a far Euclidean distance than the BMU.

2.2.1.1.7 Kohonen Self-Organizing Feature Map (KSOFM) based Session Key Generation

Based on a mutually predetermined iteration steps both sender and receiver stop their iteration due to synchronization at both end. Both sender and receiver have the identical KSOFM as they have started with same initial configuration and proceeds with same mutually agreed parameters. In this situation both sender and receiver uses identical mathematical function as a mask. A general form of the mask is represented by the equation 2.9.

$$f_{winner}\ (x) = a.e^{\frac{|k-w|^2}{\alpha_1^2}} - b.e^{\frac{|k-w|^2}{\alpha_2^2}} \tag{2.9}$$

Where, $a, b, \sigma_1, \sigma_2 \epsilon\ R,\ x$ a neuron in the KSOFM, $winner$ is the winner neuron. A huge number of masks could be generated by changing parameters a, b, σ_1, σ_2. Using the mask incontestably enhances the security of the key. Use of mathematical mask increases the security of the scheme because instead of one single neuron, session key can be constructed using several neurons on the mask. Also changing the mask parameters several session keys can be generated. This provides a significant improvement to the security of the generation of session key. A mask hides all neurons other than the winner. Here, different mask functions like Gauss, Radial basis, Mexican hat functions are used randomly in different sessions. The winner neuron fixes the center of the mask and each neuron around the winner will be weighted and summed to the winner. The result is a different session keys depending on the shape of the mask. From this discussion it can be concluded that initially the process need slightly more amount of time but once the mask get set it takes less amount of time. An adversary could not find the session key because they do not have the map, mathematical function for masking and other mutually pre agreed parameters. During mask association several neurons around the designated neuron get associated to form the session key using the equation 2.10.

$$Session\ Key_{u,q} = \sum_{i,j} w_{ij\,,q} \cdot f_{winner}\ (i,j) \quad q = 1,2, \dots, N \tag{2.10}$$

Where,

$f_{winner}\ (i,j)$is the mask centered at the winner neuron ,

$w_{ij\,,q}$ is the component q of the vector associated to neuron $(i\,,j)$ and

$Session\ Key_{u,q}$ is the component q of the ultimate (final) session key.

In wireless communication, instead of starting from the initial state of the KSOFM key generation procedure a user may use the same trained KSOFM in different session with different users by changing only the parameters value of the mask or mask function used to determine the ultimate session key. This procedure helps to save the resources of wireless communication very efficiently.

Fractal triangle has been used to perform encryption technique by placing the plaintext into the each triangle vertex and place the key bits for encryption into the middle of each triangle and then *Exclusive-OR* operation is performed between central key bit of each triangle and vertex elements of each triangle. On the outcomes of this step *Exclusive-OR* operation is performed again between triangle centered key and vertex elements of big triangle. On the outcome of the previous step *Exclusive-OR* operation between upper triangle's vertex elements are performed with right triangle's vertex elements and finally *Exclusive-OR* operation is performed between upper triangle's vertex elements with left triangle's vertex elements to generate the Fractal triangle encrypted text. The detail steps of Fractal triangle based encryption algorithm are given in section 2.2.1.2.

2.2.1.2 Fractal Triangle based Encryption Algorithm

Input : *Source file/source stream i.e. plaintext*

Output : *Encrypted file/encrypted stream i.e. cipher text*

Method : *The process operates on binary stream and generates encrypted bit stream through Fractal triangle based encryption.*

Step 1. *Perform Exclusive-OR operation between central key bit of each triangle and vertex elements of each triangle. Figure 2.6 shows the red colored vertex elements after performing the Exclusive-OR operations. For example if the Fractal triangle dimension is n= 3 and the four bit key for this encryption is "1110" and first nine bits of the plaintext is "011011110" then figure 2.6 shows the first step of the algorithm.*

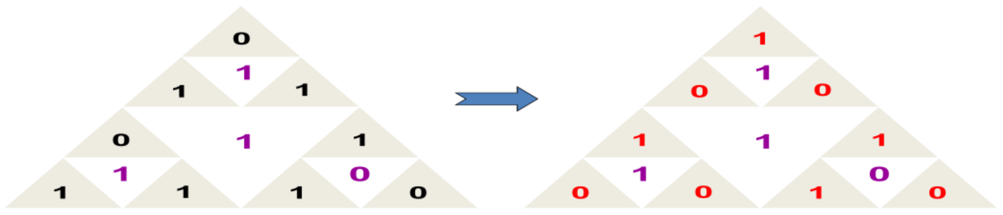


Figure 2.6: *Exclusive-OR* operation between central key bit of each triangle and vertex elements of each triangle

Step 2. *Perform Exclusive-OR operation between triangle centered key and vertex elements of big triangle. Figure 2.7 shows the green colored vertex elements after performing the Exclusive-OR operations.*

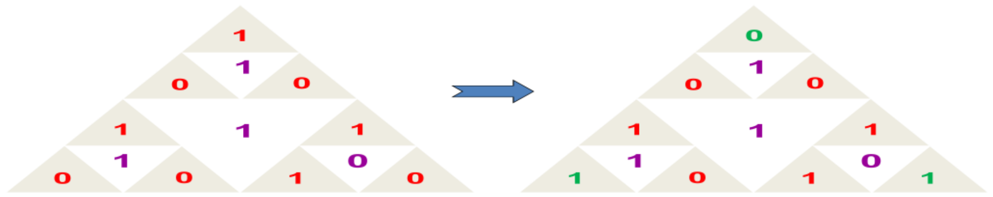


Figure 2.7: *Exclusive-OR* operation between triangle's centered key and vertex elements of big triangle

Step 3. *Perform Exclusive-OR operation between upper triangle's vertex elements with right triangle's vertex elements. Figure 2.8 shows the orange colored right triangle's vertex elements after performing the Exclusive-OR operations.*

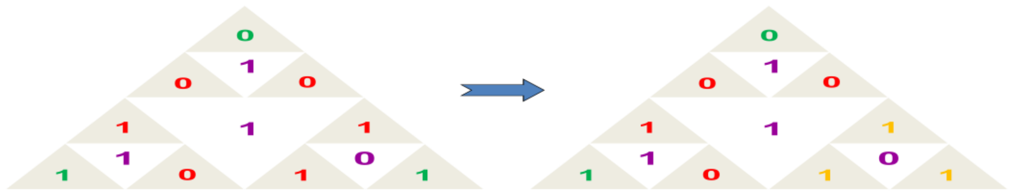


Figure 2.8: *Exclusive-OR* operation between upper triangle's vertex elements with right triangle's vertex elements

Step 4. *Perform Exclusive-OR operation between upper triangle's vertex elements with left triangle's vertex elements. Figure 2.9 shows the orange colored left triangle's vertex elements after performing the Exclusive-OR operations.*

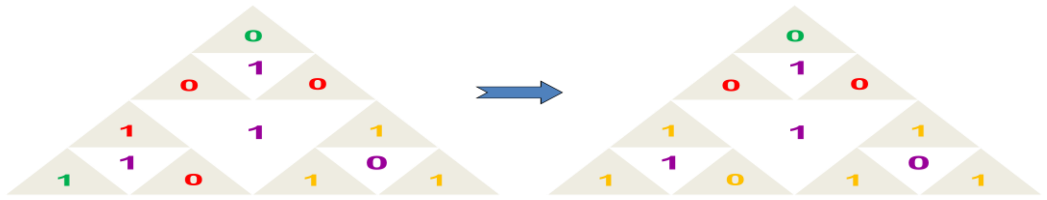


Figure 2.9: *Exclusive-OR* operation between upper triangle's vertex elements with left triangle's vertex elements

Step 5. *Now, Fractal triangle based encrypted text is "000110111" and the representation of the storage structure shown in figure 2.10*

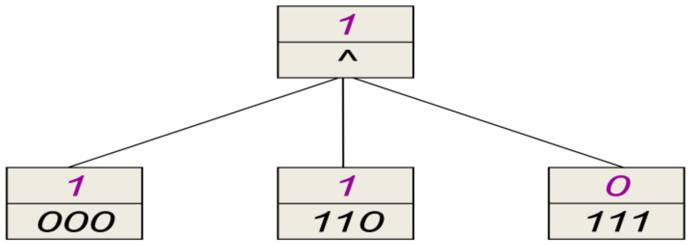


Figure 2.10: Storage structure representation of the encrypted text

2.2.1.3 Session Key based Encryption

At the final step of the technique a cascaded *Exclusive-OR* operation between KSOFM synchronized session key and Fractal triangle encrypted cipher text is done to generate final encoded cipher text and the same is transmitted to the receiver.

In Kohonen's Self-Organizing Map Synchronized Cryptographic Technique (KSOMSCT) decryption algorithm takes the cipher text as a binary stream of bits and perform first level of decryption using KSOFM generated synchronized session key. Finally, Fractal triangle based decryption is performed to regenerate the plaintext, at the receiving end. Section 2.2.2 represents the algorithm of the decryption technique at the receiver end.

2.2.2 KSOMSCT Algorithm at Receiver

Input : Encrypted file/encrypted stream i.e. cipher text

Output : Source file/source stream i.e. plaintext

Method : The process operates on encrypted binary stream and generates decrypted bit stream through Kohonen's Self-Organizing Feature Map (KSOFM) and Fractal triangle based decryption.

Step 1. Perform cascaded Exclusive-OR operation between KSOFM based session key and cipher text.

Step 2. Perform Fractal triangle based decryption on the outcomes of the step 1. Fractal triangle based reverse encryption operation will be *required to decrypt the encrypted text, i.e., to regenerate starting combination i.e. plaintext.*

Step 1 of the algorithm is discussed in section 2.2.2.1. Step 2 of the algorithm for performing Fractal triangle based decryption is discussed in 2.2.2.2.

2.2.2.1 Session Key based Decryption

A cascaded *Exclusive-OR* operation between KSOFM synchronized session key and cipher text get perform to produce session key decrypted text. Outcomes of this operation used as an input of Fractal triangle based decryption algorithm discussed in 2.2.2.2 to regenerate the plaintext.

Fractal triangle has been used to perform decryption technique by placing the plaintext into the each triangle vertex and place the key bits for encryption into the middle of each triangle and *Exclusive-OR* operation is performed between upper triangle's vertex elements with left triangle's vertex elements. On the outcomes of this step *Exclusive-OR* operation is performed again between upper triangle's vertex elements with right triangle's vertex elements. On the outcome of the previous step *Exclusive-OR* operation between triangle centered key and vertex elements of big triangle are performed and finally *Exclusive-OR* operation between central key bit and vertex elements of each triangle are performed to generate the Fractal triangle encrypted text. The detail steps of Fractal triangle based decryption algorithm are given in section 2.2.2.2.

2.2.2.2 Fractal Triangle based Decryption Algorithm

Input : *Fractal triangle encrypted file/ Fractal triangle encrypted stream*

Output : *Source file/source stream i.e. plaintext*

Method : *The process operates on Fractal triangle encrypted bit stream and regenerates the plaintext through Fractal triangle based decryption.*

Step 1. *Perform Exclusive-OR operation between upper triangle's vertex elements with left triangle's vertex elements. Figure 2.11 shows the* *green colored left triangle's vertex elements after performing the Exclusive-OR operations.*

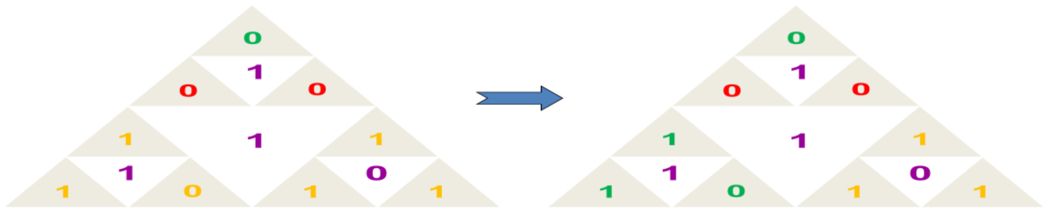


Figure 2.11: *Exclusive-OR* operation between upper triangle's vertex elements with left triangle's vertex elements

Step 2. *Perform Exclusive-OR operation between upper triangle's vertex elements with right triangle's vertex elements. Figure 2.12 shows the orange colored right triangle's vertex elements after performing the Exclusive-OR operations.*

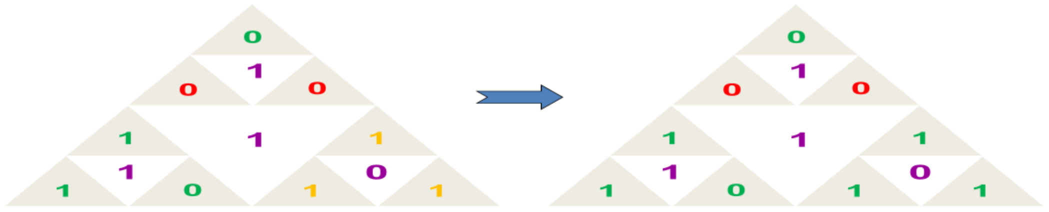


Figure 2.12: *Exclusive-OR* operation between upper triangle's vertex elements with right triangle's vertex elements

Step 3. *Perform Exclusive-OR operation between triangle centered key and vertex elements of big triangle. Figure 2.13 shows the orange colored vertex elements after performing the Exclusive-OR operations.*

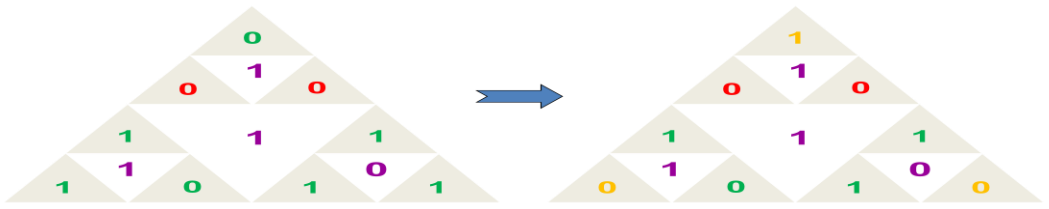


Figure 2.13: *Exclusive-OR* operation between triangle's centered key and vertex elements of big triangle

Step 4. *Perform Exclusive-OR operation between key and vertex elements of each triangle. Figure 2.14 shows the blue colored vertex elements after performing the Exclusive-OR operations.*

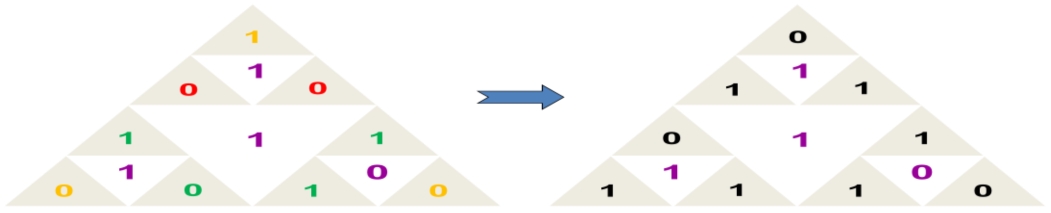


Figure 2.14: *Exclusive-OR* operation between key and vertex elements of each triangle

2.3 Implementation

To perform Fractal triangle based encryption first considers the KSOFM synchronized session key. If the Fractal dimension is three then first four bits form synchronized session key form the encryption/decryption key. For example from the KSOFM synchronized 128 bits, the following session key is generated

1001/1110/1000/1110/0111/0100/0101/0111/1110/0101/01010101/10100011/11011010/
10100011/11010100/10111101/01001101/01101111/10100010/11000011/11101010

Here "/" is used as the separator between successive bytes.

Now, consider the plaintext to be encrypted is "**Technique**", binary representation of the ASCII value of plaintext is

01010100/01100101/01100011/01101000/01101110/01101001/01110001/01110101/
01100101

First four bits of KSOFM synchronized session key i.e. 1001 becomes the key for Fractal triangle encryption of first nine bits of the plaintext i.e. 010101000. For the rest of the plaintext each time nine bits are taken and the next four bits of the synchronized session key becomes the Fractal triangle encrypted key for the particular block. This process will continue until plaintext gets exhausted. The process is given in table 2.6 to table 2.13.

Table 2.6 illustrates the encryption of plaintext block 010101000 using key 1001. After step-4 the encrypted text is 110001000.

Table: 2.6
Fractal triangle encryption of 010101000

Plaintext block : 010101000	**Key:** 1001
Initial Fractal Triangle Value	010101000
After step 1	010101111
After step 2	110111110
After step 3	110111000
After step 4 (Encrypted text)	110001000

Table 2.7 illustrates the encryption of plaintext block 110010101 using key 1110. After step-4 the encrypted text is 101010001.

Table: 2.7
Fractal triangle encryption of 110010101

Plaintext block: 110010101	**Key:** 1110
Initial Fractal Triangle Value	110010101
After step 1	001101101
After step 2	101111100
After step 3	101111001
After step 4 (Encrypted text)	101010001

Table 2.8 illustrates the encryption of plaintext block 100011011 using key 1000. After step-4 the encrypted text is 000001010.

Table: 2.8
Fractal triangle encryption of 100011011

Plaintext block: 100011011	**Key:** 1000
Initial Fractal Triangle Value	100011011
After step 1	100011011
After step 2	000001010
After step 3	000001010
After step 4 (Encrypted text)	000001010

Table 2.9 illustrates the encryption of plaintext block 010000110 using key 1110. After step-4 the encrypted text is 001100110.

Table: 2.9
Fractal triangle encryption of 010000110

Plaintext block: 010000110	**Key:** 1110
Initial Fractal Triangle Value	010000110
After step 1	101111110
After step 2	001101111
After step 3	001101110
After step 4 (Encrypted text)	001100110

Table 2.10 illustrates the encryption of plaintext block 111001101using key 0111. After step-4 the encrypted text is 000110010.

Table: 2.10
Fractal triangle encryption of 111001101

Plaintext block: 111001101 **Key:** 0111	
Initial Fractal Triangle Value	111001101
After step 1	000110010
After step 2	000110010
After step 3	000110010
After step 4 (Encrypted text)	000110010

Table 2.11 illustrates the encryption of plaintext block 001011100 using key 0100. After step-4 the encrypted text is 110101010.

Table: 2.11
Fractal triangle encryption of 001011100

Plaintext block: 001011100 **Key:** 0100	
Initial Fractal Triangle Value	001011100
After step 1	110011100
After step 2	110011100
After step 3	110011010
After step 4 (Encrypted text)	110101010

Table 2.12 illustrates the encryption of plaintext block 010111010 using key 0101. After step-4 the encrypted text is 101010000.

Table: 2.12
Fractal triangle encryption of 010111010

Plaintext block: 010111010 **Key:** 0101	
Initial Fractal Triangle Value	010111010
After step 1	101111101
After step 2	101111101
After step 3	101111000
After step 4 (Encrypted text)	101010000

Table 2.13 illustrates the encryption of plaintext block 101100101using key 1110. After step-4 the encrypted text is 110111010.

Table: 2.13
Fractal triangle encryption of 101100101

Plaintext block: 101100101 **Key:** 1110	
Initial Fractal Triangle Value	101100101
After step 1	010011101
After step 2	110001100
After step 3	110001010
After step 4 (Encrypted text)	110111010

Now, Fractal triangle based encrypted text is
11000100/01010100/01000001/01000110/01100001/10010110/10101010/10100001/
10111010
On performing the *Exclusive-OR* between KSOFM synchronized session key and Fractal triangle encrypted text, final cipher text is generated as follows
01011010/11011010/00110101/00010001/10000100/11000011/00001001/01111011/
00011001.

2.4 Security Analysis

In this chapter a Kohonen's Self-Organizing Map Synchronized Cryptographic Technique (KSOMSCT) has been proposed. The technique generates the synchronized session key by tuning KSOFM of both sender and receiver. Plaintext gets encrypted using Fractal triangle based encryption. Outcomes of this process and final tuned session key get *Exclusive-OR* to produce the final cipher text and same is transmitted to the sender. The Following standard attacks are considered to ensure the robustness of the technique.

- *Cipher text only Attack:* In this type of attack, the attacker has access to a set of cipher text. In cipher text only attack, encryption algorithm and cipher text is known to an attacker. An attacker tries to break the algorithm or in simple words tries to deduce the decryption key or plaintext by observing the cipher text. The KSOMSCT nullifies the success rate of this attack by producing a robust KSOFM and Fractal triangle based encrypted cipher text. The strength of resisting exhaustive key search attack relies on a large key space. Initially, Fractal triangle based encryption used to encrypt the plaintext after that outcome of this passes through KSOFM session key based encryption process. So, cipher text produces by the technique is mathematically difficult to break. Thus a hacker has to try all such key streams to find an appropriate one. Keystream have high degrees of correlation immunity. Thus it is practically difficult to perform a brute-force search in a key-space.

- *Known Plaintext Attack:* The attacker has access to one or more cipher text and some characters in the original data. The objective is to find the secret key. The technique offers better floating frequency of characters. So, known plaintext attack is very difficult in this proposed technique.

- *Chosen Plaintext Attack:* Here, the attacker has liberty to choose a plaintext of his/her choice and get the corresponding cipher text. Since the attacker can choose plaintext of his/her choice, this attack is more powerful. Again the objective of this attack is to find the secret key. This attack is impractical for KSOMSCT because there is no obvious relationship between the individual bits of the sequence in plaintext and cipher text. So, it is not possible to choose a plaintext of his/her choice and get the corresponding cipher text.

- *Chosen Cipher text Only Attack:* The attacker can choose cipher text and get the corresponding plaintext. By selecting some cipher text a cryptanalyst has access to corresponding decrypted plaintext. Chosen cipher text only attack is more applicable to public key cryptosystems. The technique has a good Chi-Square value this confirms good degree of non-homogeneity. So, it will be difficult to regenerate plaintext from the cipher text.

- *Brute Force Attack:* A cryptanalyst tries all possible keys in finite key space one by one and check the corresponding plaintext, if meaningful. The basic objective of a brute force attack is to try all possible combinations of the secret key to recover the plaintext image and or the secret key. On an average, half of all possible keys must be tried to achieve success but brute force attack involves large computation and has a very high complexity. Due to high complexity brute force attack will not be feasible. The technique has a good entropy value near to eight which indicates that brute force attack is very difficult in the proposed technique.

2.5 Discussions

The technique is very simple and easy to implement in various high level language. The test results show that the performance and security provided by the technique is good and comparable to standard technique. The security provided by the KSOMSCT is comparable with other techniques. To enhance the security of the technique, proposed technique offers changes of some parameters randomly in each session. To generate the secret session key index mask is exchanged between sender and receiver. This technique has a unique ability to construct the secret key at both sides using this exchanged information. Since the encryption and decryption times are much lower, so processing speed is very high. Proposed method takes minimum amount of resources which is greatly handle the resource constraints criteria of wireless communication. This method generates a large number of keys which is the same number of neurons in the map. For ensuring the randomness in every session, some of the parameters get change randomly at each session. KSOMSCT outperform than existing TPM, PPM and does not suffers from Brute Force or Man-In-The-Middle (MITM) attack. No platform specific optimizations were done in the actual implementation, thus performance should be similar over varied implementation platform. The whole procedure is randomized, thus resulting in a unique process for a unique session, which makes it harder for a cryptanalyst to find a base to start with. The technique is applicable to ensure security in message transmission in any form and in any size in wireless communication.

Some of the salient features of KSOMSCT are summarized as follows:

a) *Session key generation and exchange – Identical session key can be generate after the tuning of KSOFM in both sender and receiver side. So, no need to transfer the whole session key via vulnerable public channel.*

b) *Degree of security – The technique does not suffers from cipher text only attack, known plaintext attack, chosen plaintext attack, chosen cipher text only attack, brute force attack.*

c) *Variable block size – Encryption algorithm can work with any block length and thus not require padding, which result identical size of files both in original and encrypted file. So, KSOMSCT has no space overhead.*

d) *Variable key – 128/192/256 bit session key with high key space can be used in different sessions. Since the session key is used only once for each transmission, so there is a minimum time stamp which expires automatically at the end of each transmission of information. Thus the cryptanalyst may not be able guess the session key for that particular session.*

e) *Complexity – The technique has the flexibility to adopt the complexity based on infrastructure, resource and energy available for computing in a node or mesh through wireless communication. So, the KSOMSCT may be suitable in wireless communication.*

f) *Non-homogeneity – Measures of central tendency, dispersion and Chi-Square value have been performed between the source and corresponding cipher streams generated using KSOMSCT. All measures indicate that the degree of non-homogeneity of the encrypted stream with respect to the source stream is good.*

g) *Floating frequency – In the technique it is observed that floating frequencies of encrypted characters are indicates the high degree of security for the technique.*

a) *Entropy – The entropy of encrypted characters is near to eight which indicate the high degree of security for the proposed technique.*

h) *Correlation – The cipher stream generated through proposed technique is negligibly correlated with the source stream. Therefore the proposed technique may effectively resist data correlation statistical attack.*

i) *Key sensitivity – The technique generates an entirely different cipher stream with a small change in the key and technique totally fails to decrypt the cipher stream with a slightly different secret session key.*

j) *Security and performance trade-off – The technique may be ideal for trade-off between security and performance of light weight devices having very low processing capabilities or limited computing power in wireless communication.*

Chapter 3

Double Hidden Layer Perceptron Synchronized Cryptographic Technique (DHLPSCT)

3.1 Introduction

In this chapter a novel soft computing assisted cryptographic technique DHLPSCT, based on synchronization of two Double Hidden Layer Perceptron (DHLP)[200], one at sender and another at receiver has been proposed. The KSOMSCT technique proposed in chapter 2 have some drawbacks like sender and receiver both have to be agreed on several predefined parameters which get send via public channel. So, there is an overhead of parameters passing and an associated risk in terms of security. Also initially large numbers of neurons need to form the KSOFM for which significant amount of memory as well as large amount of training cycles is required to train all the neurons in the map. Furthermore, there is no well defined terminating criteria to terminate the training of KSOFM. The existing Tree Parity Machine (TPM) and Permutation Parity Machine (PPM) are also not the best alternative solution which has already analysed in chapter 2. DHLPSCT eliminates all the above mentioned drawbacks of the KSOMSCT in chapter 2, existing TPM and PPM. The technique need less number of synchronization steps than earlier techniques. It also greatly handles the resource constraints criteria of wireless communication and passes fewer parameters than the earlier techniques which in turn significantly reduces the risk and hence increases the security.

Here, DHLP based synchronization is performed for tuning both sender and receiver. On the completion of the tuning phase identical session keys are generated at the both end with the help of synchronized DHLP. This synchronized network can be used for transmitting message using any light weight encryption/decryption technique with the help of session key of the synchronized network. To illustrate the cryptographic technique using DHLP in wireless communication one of the simple and secure encryption/decryption technique has been presented. A plaintext is considered as a stream of binary bits. Genetic operation based Simulated Annealing (SA) guided enciphering technique[201] with the help of DHLP tuned session key is used to generate the cipher text. The plaintext is regenerated from the cipher text using same technique with the help of DHLP tuned session key at the receiver.

Section 3.2 represents a description of proposed technique in detail. Section 3.3 deals with the implementation of the proposed cryptographic technique. Section 3.4 discussed the security issues related to the proposed technique. Discussions are presented in section 3.5.

3.2 The Technique

The technique performs DHLP based synchronization for generation of secret session key at both ends. This synchronized session key of the tuned network is used for the transmission of secured message through wireless network with the help of any light weight encryption/decryption algorithm. To illustrate the cryptographic technique in wireless communication one of the simple and secure encryption/decryption technique has been proposed, where plaintext (i.e. the input file) is considered as a stream of binary bits, which is encrypted using genetic operation based SA generated fittest encryption/decryption keystream. The session key based on DHLP is used to encrypt intermediate output which produces final cipher text. Identical DHLP is used to tune the sender and receiver to generate the secret session key at both ends. Two DHLPs at sender and receiver start with common input vector and completely anonymous random weight vector. In each time DHLPs at both end compute their final output based on input and weight vector, and communicate to each other. If both are be in agreement on the mapping between the present input and the output, their weights are updated according to an appropriate learning rule. In case of discrete weight values this process leads to full synchronization in a finite number of steps. After synchronization weight vector of both DHLPs become identical. This indistinguishable weight vector forms the session key.

Genetic operation based Simulated Annealing guided encryption/decryption process generates the initial population of individuals randomly (i.e. keystream) having population size of 200 individuals. Each individual that represents the candidate keystream is strings of characters 'a' ... 'p'. The letters 'a' ... 'p' represent the numbers $0\ldots15$. Thus, each letter is a sequence of four bits. Fitness values of each keystream in the population are calculated depending on the randomness of the generated keystream, keystream period length and keystream length. Genetic operation based SA guided keystream generation algorithm set the initial temperature to 250 and select a value up to which the algorithm will iterate i.e. the maximum number of generation depend on the resource available at the time of wireless communication. Each generation the process checks whether the current generation number is less or equal to the maximum number of generations, if so, then the operations is repeated. At first the single point crossover is perform in the mating pool with a crossover probability of 0.6 to 0.9 on the keystrem having higher fitness. Then the mutation operation is performed with comparatively lower mutation probabilities i.e. $.001$ to $.01$ to produce new generation having some genetic diversity. Fitness calculation is done again for this newly generated keystream and then checks whether the fitness value of new generation is better than the fitness value of old generation, if so, then the process set the new population as the current population. Otherwise, it computes e as the differences between fitness value of old generation and fitness of new generation after that $Pr = e/temperature$ is computed. Now, if $exp(-pr)$ is grater than an arbitrary random number, then process set current population as an new population, otherwise temperature is updated by multiplying the temperature with an α vale (in this experiment $\alpha = 0.95$). These steps is repeated in several generations until the best fittest keystream is obtained or maximum number of generation is reached whichever is earlier. If the length of the plaintext to be encrypted is grater then the length of generated keystream then triangle edge based key expansion method is used to extend the length of the keystream. Stream of plaintext is encrypted using the SA based keystream/extended keystream. Finally a cascaded *Exclusive-OR* operation is performed between SA encrypted text and the DHLP based session key to generate final cipher text.

Receiver has same DHLP generated synchronized session key as a result of tuning. This session key is used to perform first step of the deciphering. In the next step, SA guided keystream based deciphering operation is performed to regenerate the plaintext.

The DHLPSCT does not cause any storage overhead. This greatly handles the resource constraints criteria of wireless communication. A comparison of DHLPSCT with previously proposed technique in chapter 2, existing Tree Parity Machine (TPM), Permutation Parity Machine (PPM), and industry accepted AES, RC4, Vernam Cipher, Triple DES (TDES) and RSA have been done. Analyses of results are given in chapter 7.

In DHLPSCT, encryption process takes the plaintext as a binary stream of bits which is encrypted using genetic operation based SA generated fittest keystream. SA encoded text is encrypt further through *Exclusive-OR* operation with the session key. The algorithm for the complete process is given in section 3.2.1.

3.2.1 DHLPSCT Algorithm at Sender

Input : Source file/source stream i.e. plaintext

Output : Encrypted file/encrypted stream i.e. cipher text

Method : The process operates on binary stream and generates encrypted bit stream through DHLP guided Simulated Annealing (SA) based encryption technique.

Step 1. Perform tuning of sender's and receiver's DHLP to generate common secret session key.

Step 2. Generates SA based fittest encryption keystream.

Step 3. Perform SA based encryption operation on the plaintext.

Step 4. Perform cascaded Exclusive-OR operation between DHLP based session key and outcomes of step 3.

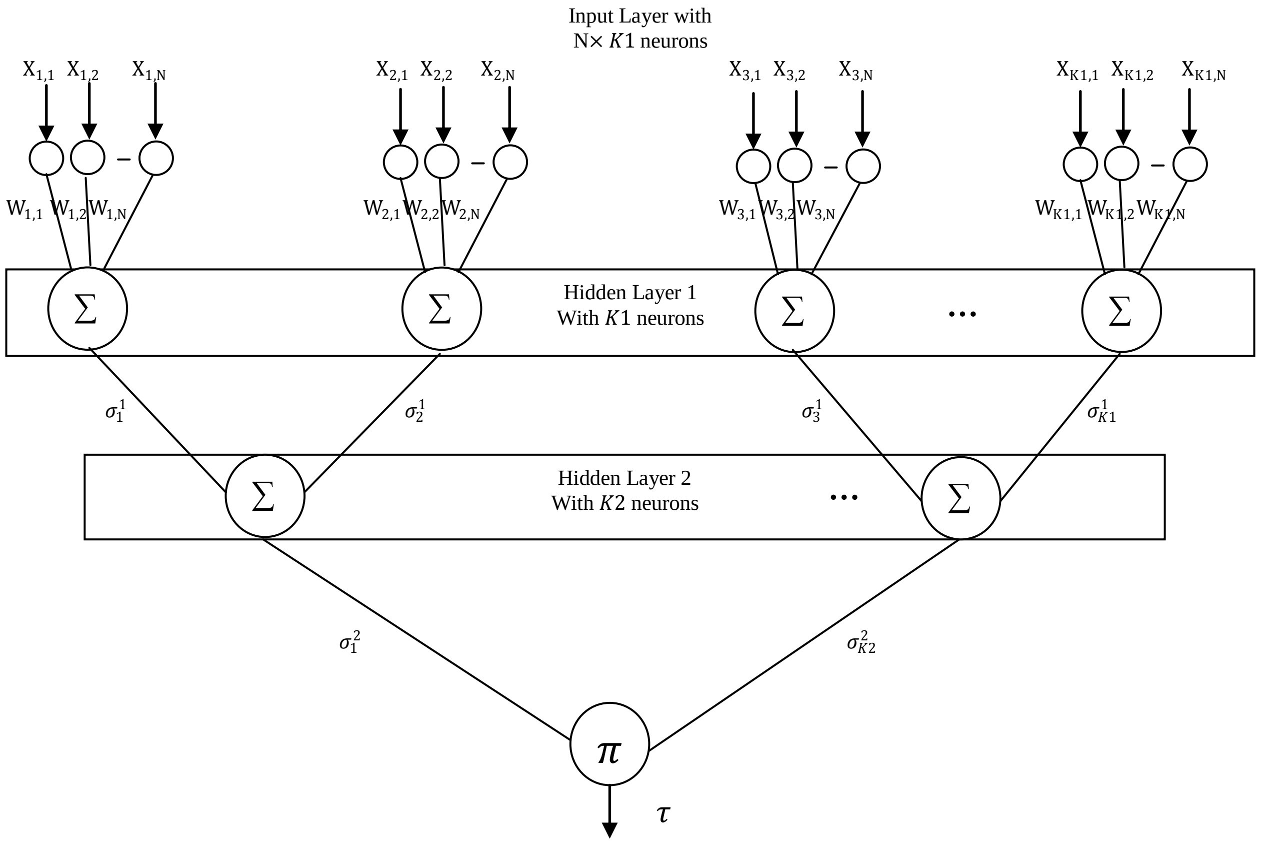
Step 1 of the algorithm generate common session key through synchronization of DHLP at both end. The detailed step is discussed in section 3.2.1.1. Step 2 of the algorithm generates SA based fittest encryption keystream. The detailed description of the process is given in section 3.2.1.2. Algorithm for performing SA based encryption operation (step 3) on the plaintext is discussed in 3.2.1.3. The technique of cascading encryption process (step 4) which takes the intermediate output generated in step 3 is given in details in section 3.2.1.4.

3.2.1.1 Double Hidden Layer Perceptron (DHLP) based Synchronization and Session Key Generation

It is seen that Artificial Neural Networks can synchronize. These mathematical models have been developed to study and simulate the activities of biological neurons at the beginning. But with a very short span of time it was discovered that complex problems in computer science can be solved by using Artificial Neural Networks. Neural synchronization is used to construct a cryptographic key-exchange protocol. Here, the partners are benefitted from mutual interactions, so that a passive attacker is usually prevented. If the synaptic depth (L) is increased, the complexity of a successful attack grows exponentially, but there is only a polynomial increase of the effort needed to generate a key. Using the basic concept of neural synchronization, this chapter offers a novel session key generation technique using DHLP (Double Hidden Layer Perceptron) for transmitting the cipher text session wise using unique session key. Here, both sender and receiver use an identical DHLP. DHLPs at both end start with identical input vector and anonymous random weight vector. In each time both DHLPs compute their output based on inputs and weight vector, and communicate to each other. If both are in agreement, their weights are updated through appropriate learning rules. In case of discrete weight values this process leads to full synchronization in a finite number of steps. On synchronization weight vector of both DHLPs become identical. From this indistinguishable weight vector session key for a particular session is formed. So, as a substitute of transferring the whole session key through public channel DHLP based synchronization process is carried out and outcomes of this is used as a secret session key for that entire session. In DHLP following salient features may be obtained to improve the security of the communication.

- DHLP offers two hidden layers instead of single hidden layer in TPM
- Instead of increasing number of hidden neurons in a single hidden layer, DHLP introduces an additional layer (second hidden layer) which actually increased the structural complexity of the network that in turn helps to make the attacker's life difficult to guessing the internal representation of DHLP
- Weight vector consisting of discrete values are used for faster synchronization
- $SYN, ACK\_SYN, NAK\_SYN, FIN$ frames are used to perform connection establishment and synchronization procedure
- Three different learning rules are used based on the network size for faster synchronization
- The process generates variable length bits length session key where key space is higher
- DHLP enhance the security by increasing the range of the values of weight vector (L)

The figure 3.1 shows a perceptron with two hidden layers. Here $K1 = 4$ and $K2 = 2$. So, the first hidden layer $K1$ has four hidden neurons. The second hidden layer $K2$ has two hidden neurons. The total number of inputs neurons are $N \times K1$, where N is the number of inputs to each hidden neuron in layer 1.



Input Layer with
N× $K1$ neurons
$X_{1,1}$ $X_{1,2}$ $X_{1,N}$
$X_{2,1}$ $X_{2,2}$ $X_{2,N}$
$X_{3,1}$ $X_{3,2}$ $X_{3,N}$
$X_{K1,1}$ $X_{K1,2}$ $X_{K1,N}$
$W_{1,1}$ $W_{1,2}$ $W_{1,N}$
$W_{2,1}$ $W_{2,2}$ $W_{2,N}$
$W_{3,1}$ $W_{3,2}$ $W_{3,N}$
$W_{K1,1}$ $W_{K1,2}$ $W_{K1,N}$
Hidden Layer 1
With $K1$ neurons
σ_1^1 σ_2^1 σ_3^1 σ_{K1}^1
Hidden Layer 2
With $K2$ neurons
σ_1^2 σ_{K2}^2
π
τ

Figure 3.1: A DHLP with two hidden layers

A TPM usually consist of K hidden neurons, $N \times K$ no. of input neurons having binary input vector, $x_{ij} \in \{-1, +1\}$, discrete weights are generated from input to output, are lies between $-L$ and $+L$, $w_{ij} \in \{-L, -L+1, \ldots, +L\}$.Where $i = 1, \ldots, K$ denotes the ith hidden unit of the TPM and $j = 1, \ldots, N$ the elements of the vector and one output neuron. So, there are 2^{K-1} different internal representations $(\sigma_1, \sigma_2, \ldots, \sigma_K)$, which lead to the same output value τ. In DHLP, the parameter K is divided into $K1$ and $K2$. $K1$ numbers of hidden neurons resides in the hidden layer adjacent to the input layer, that of $K2$ represents number of hidden neurons adjacent to the output layer. Now for each $K1$ hidden neurons there are N inputs possible. Hence, the input layer has $N \times K1$ input neurons. The size of the DHLP is represented by $N \times K1 \times K2$.

Total number of weights generated by the DHLP is $(N \times K1 + K1 \times K2)$. Decimal value of each weight is represented in eight bit binary. So, total $((N \times K1 + K1 \times K2) \times 8)$ number of bits are present in a weight (length of a session key). If $N = 1, K1 = 8, K2 = 1$

then $((1 \times 8 + 8 \times 1) \times 8) = 128$ bits weight value may constitute the session key. Consider the synaptic depth i.e. weight limits $L = \pm 127$. So, eight binary bits are needed to represents each weight, where the MSB represents the sign bit and rest of the seven bits represents the magnitude of the weight. The figure 3.2 shows the single path form input neuron to the output neuron.

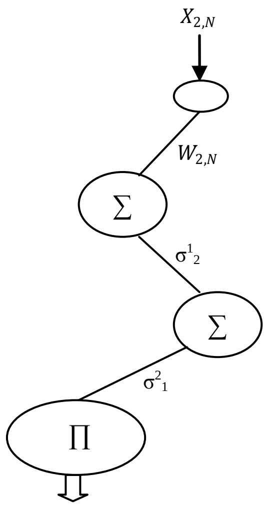


Figure 3.2: Snapshot of a single path of DHLP

Two DHLPs start with identical input vectors generated by sender’s and receiver’s identical secret seed value and completely different random weight vectors. DHLPs compute their final output based on input and weight vector, and communicate to each other. If both are in agreement, their weights are updated using appropriate learning rules. Within finite number of steps both DHLPs get synchronized and as a results weight vector of both DHLPs become identical. These indistinguishable weight vectors forms the session key for a particular session.

In DHLPSCT both DHLPs start synchronization by exchanging control frames. The process involves message integrity and synchronization test. DHLP synchronization uses transmission of control frames at the time of three way handshaking based TCP connection establishment phase, as given in table 3.1.

Table 3.1
Control frames of DHLP synchonization

Frame	Description
SYN	SYN frame transmitted to the receiver for synchronization in connection establishment phase
ACK_SYN	ACK_SYN frame transmitted to the sender for positive acknowledgement respect to SYN frame
NAK_SYN	NAK_SYN frame transmitted to the sender for negative acknowledgement respect to SYN frame
FIN_SYN	FIN_SYN frame transmitted by either party for closing the connection

The SYN frame is used to establish the connection to the other side. It carries index information of different initial parameters. On transmitting the SYN frame, the sender starts a timer and waits for a reply from the receiver. If the receiver does not take any action until a certain time limit and number of attempts exceeded a certain value, the sender restarts the synchronization procedure. When the receiver receives the SYN frame, it should carry out the integrity test. If the messages are received as sent (with no replication, incorporation, alteration, reordering, or replay) the receiver will execute the synchronization check. The sender and receiver have an identical T variable formally store in their respective memory. The sender sends the encrypted T to the receiver. Here the receiver utilizes its $128/192/256$ bits weights to decrypt the encrypted T. If the result is identical to T formerly stored in receiver memory i.e. $(Decrypt_{Receiver\ \_weig\ ht}(Encrypt_{Sender\ \_weig\ ht}(T)) = T$ then the networks are synchronized. This is the best case solution where sender and receiver arbitrarily choose weight vector which are identical. So, networks are synchronized at initial stage. The receiver should send the FIN_SYN frame to alert the sender. But most of the time this best case is may not achievable. If decryption algorithm does not produce predictable result, the receiver should use the secret seed value of sender's to produce the input vector(X) which is identical to sender. With this input vector the receiver will work out its $output\ (\tau^{Sender})$. If the outputs at both ends are different, the receiver should not fine-tune its weights and inform the sender its output. The receiver sends the $NACK\_SYN$ frame to notify the sender, with the same ID value. The proposed NAK_SYN frame in this technique is used for providing the negative acknowledgement in respect to the SYN frame. If receiver's output is equal to sender's output i.e. $(\tau^{Receiver} = \tau^{Sender})$ then receiver update it weights.

At the end of updates, the receiver should report the sender that outputs are equal. The receiver uses the ACK_SYN frame to notify the sender, with the same ID value received from sender. The proposed ACK_SYN frame in this technique is used for providing the positive acknowledgement in respect to the SYN frame. On receipt of ACK_SYN, the sender also updates its weight. If sender receives ACK_SYN it should update its weights. The sender will create new synchronization frame until receive the frame FIN_ACK from receiver. When the sender receives the FIN_ACK frame, it stops the further synchronization. The proposed FIN_SYN frame in this technique is used for closing the connection. At end of synchronization, both networks provide the identical weight vector which acts as a session key identical to both end. Table 3.2 shows the different frames and their corresponding command codes.

Table 3.2
DHLP control frames and their command codes

Frame	**Command**
SYN	0000
FIN_SYN	0001
ACK_SYN	0010
NAK_SYN	0011
$AUTH$	0100
$Reserved$	0101-1111

The identifier (ID) is the function of informing the sender and receiver where the message is a recent message. The variable ID starts with zero and is incremented every time that the sender sends a synchronization frame. The figure 3.3 shows the exchange of frames during DHLP synchronization process.

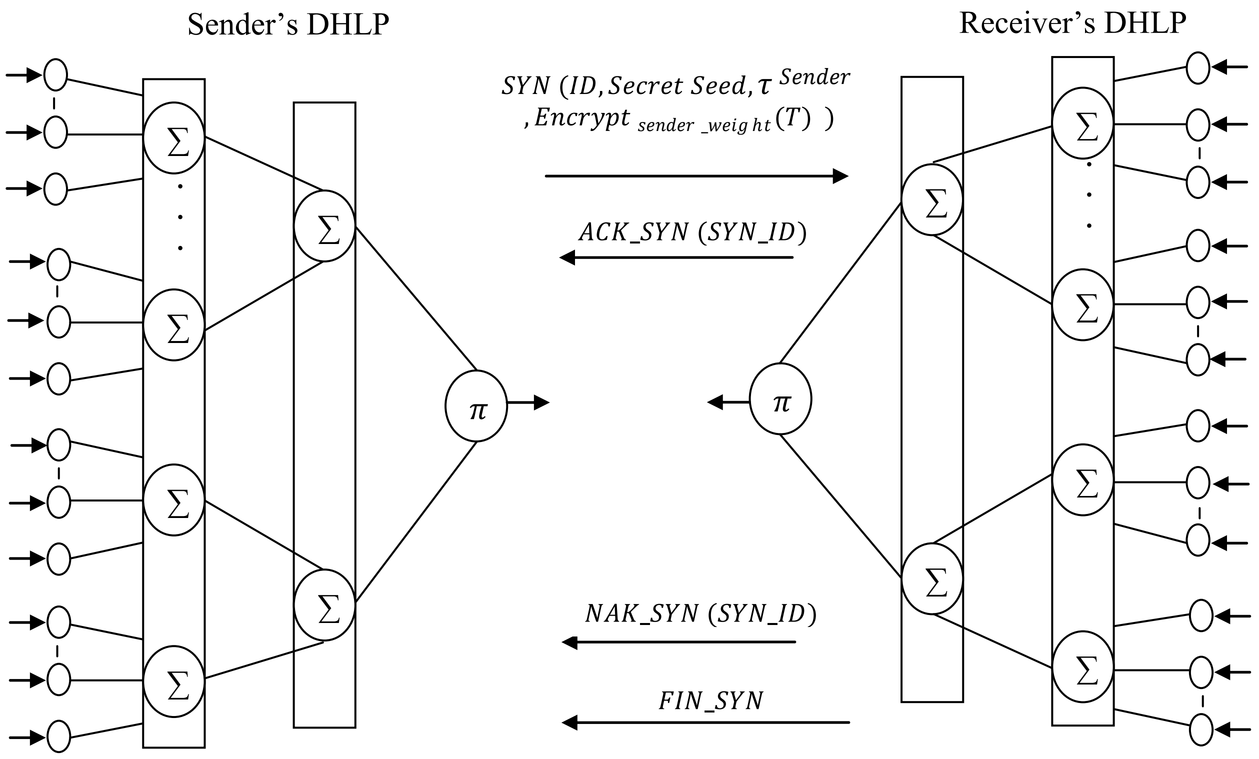


Figure 3.3: Exchange of control frames between sender and receiver during DHLP synchronization

The detailed frame format of SYN frame is discussed in section 3.2.1.1.1. The detailed frame format of ACK_SYN frame is given in section 3.2.1.1.2. The frame format of NAK_SYN frame has been discussed in section 3.2.1.1.3. The frame format of FIN_SYN frame is discussed in section 3.2.1.1.4.

3.2.1.1.1 Synchronization (SYN) Frame

During synchronization process sender constructs a SYN frame and transmit to the receiver for handshaking in connection establishment phase. Sender utilizes its initial 128 weights as key for encryption of T variable (formerly stored in its memory) $Encrypt_{Sender\_weight}(T)$. Sender constructs a SYN frame and transmitted to the receiver for handshaking purpose in connection establishment phase. SYN usually comprises of $Command\ Code, SYN\ ID$, $Secret\ Seed, Sender\ output(\tau^{Sender})$, $Encrypt_{Sender\_weight}(T)$ and CRC. SYN frame has the fixed $Command\ Code$ i.e. 0000. So, $Command\ Code$ needs four bits. $SYN\ ID$, $Secret\ Seed, Sender\ output(\tau^{Sender})$, $Encrypt_{Sender\_weight}(T)$ and CRC needs eight bits, 128 bits, one bits, 128 bits, sixteen bits respectively. When the receiver receive SYN frame, the receiver should carry out integrity test. Receiver performs integrity test on receiving the SYN frame. If the messages are received as sent (with no replication, incorporation, alteration, reordering, or replay) the receiver will execute the synchronization test. In synchronization test receiver utilize its 128 first weights as key for decryption of $Encrypt_{Sender\_weight}(T)$ that was received from the sender. After decryption operation if $(Decrypt_{Receiver\_weight}(Encrypt_{Sender\_weight}(T)) = T$ then networks are synchronized. Figure 3.4 shows the complete frame format of SYN frame.

$Command\ Code$ 0000	$SYN\ ID$	$Secret\ Seed$	τ^{Sender}	$Encrypt_{Sender\_weight}(T)$	CRC ($Cyclic\ Redundancy\ Checker$)
4	8	128	1	128	16 ($bits$)

Figure 3.4: Synchronization (SYN) frame

3.2.1.1.2 Synchronization (ACK_SYN) Acknowledgement Frame

ACK_SYN frame send by the receiver to the sender in respect of ACK frame for positive acknowledgement of the parameters value. The proposed frame comprises of $Command\ Code, SYN\ ID, CRC$. $Command\ Code$ needs four bits. The ACK_SYN frame has the fixed $Command\ Code$ i.e. 0010. Eight bits is used for representing the $SYN\ ID$ and CRC needs sixteen bits for error checking purpose. Now check the condition i.e. If $(Decrypt_{Receiver\_weight}(Encrypt_{Sender\_weight}(T)) \neq T$ then receiver use the $Secret\ Seed$ received from sender to produce the receiver input vector (X) identical to sender input vector (X) and calculates the output $\tau^{Receiver}$. If $(\tau^{Receiver} = \tau^{Sender})$ then receiver should update their weights where $\sigma_k^{Sender/Receiver} = \tau^{Sender/Receiver}$ using learning rule. At end of weight updation of the receiver, it sends ACK_SYN with the same ID to instruct the sender for updating the weights. If sender receives ACK_SYN it should update its weights. Figure 3.5 shows the complete frame format of ACK_SYN frame.

$Command\ \ Code$ 0010	$SYN\ ID$	CRC $(Cyclic\ Redundancy\ \ Checker)$
4	8	$16\ (bits)$

Figure 3.5: Acknowledgement of Synchronization (ACK_SYN) frame

3.2.1.1.3 Negative Acknowledgement (NAK_SYN) Frame of Synchronization

NAK_SYN frame send by the receiver to the sender in respect of ACK frame for negative acknowledgement of the parameters value. The proposed frame comprises of $Command\ Code, SYN\ ID, CRC$. $Command\ Code$ needs four bits. The ACK_SYN frame has the fixed $Command\ Code$ i.e. 0011. Eight bits are used for representing the $SYN\ ID$ and CRC needs sixteen bits for error checking purpose. If $(\tau^{Receiver} \neq \tau^{Sender})$ then the receiver sends the message NAK_SYN to notify the sender. If the receiver's and sender's outputs are different, the receiver should not fine-tune its weights and inform the sender. The receiver sends the message NAK_SYN to notify the sender, with the same ID value. Figure 3.6 shows the complete frame format of NAK_SYN frame.

$Command\ \ Code$ 0011	SYN_ID	CRC $(Cyclic\ Redundancy\ \ Checker)$
4	8	$16\ (bits)$

Figure 3.6: Negative Acknowledgement of Synchronization (NAK_SYN) frame

3.2.1.1.4 Finish Synchronization (FIN_SYN) Frame

FIN_SYN frame send by the either party for finish the synchronization process. This proposed frame comprises of $Command\ Code, SYN\ ID, CRC$. $Command\ Code$ needs four bits. The FIN_SYN frame has the fixed $Command\ Code$ i.e. 0001. Eight bits is used for representing the $SYN\ ID$ and CRC needs sixteen bits for error checking purpose. If $(Decrypt_{Receiver\ \_weig\ ht}(Encrypt_{Sender\ \_weig\ ht}(T)) = T$ then networks are synchronized.

Receiver sends the FIN_SYN frame to the sender. Figure 3.7 shows the complete frame format of FIN_SYN frame.

$Command\ \ Code$ 0001	$SYN\ ID$	CRC $(Cyclic\ Redundancy\ \ Checker)$
4	8	16 $(bits)$

Figure 3.7: Finish Synchronization (FIN_SYN) frame

The DHLP synchronization algorithm for generating synchronized session key is discussed in section 3.2.1.1.5. Section 3.2.1.1.6 presents the complexity analysis of the DHLP synchronization algorithm and DHLP learning is discussed in section 3.2.1.1.7.

3.2.1.1.5 DHLP Synchronization

Input : *Random weights and identical input vector*(X) *for both DHLPs*

Output : *Sender's and receiver's synchronized DHLP along with synchronized session key*

Method : *Sender's and receiver's DHLPs both are be in agreement on the mapping between the present input and the output, their weights are updated according to an appropriate learning rule. After synchronization procedure weight vector of both DHLPs become identical. These indistinguishable weight vector forms the session key for a particular session.*

Step 1. *Initialization of synaptic links (weight values) between input layer and first hidden layer also between first hidden layer and second hidden layer using random weights values. Where,* $W_{ij} \epsilon \{-L, -L+1, \ldots, +L\}$.
Repeat step 2 to step 11 until the full synchronization is achieved.

Step 2. *The input vector*(X) *are generated by the sender using* 128 *bit secret seed value.*

Step 3. *Computes the values of hidden neurons by the weighted sum over the current input values. Each hidden neuron in first hidden layer produces* σ^{1}_{i} *values and each hidden neuron in second hidden layer produces* σ^{2}_{p} *values.*

These can be represented using equation 3.1 and 3.2

$$\sigma^1{}_i = sgn\left(\sum_{i=1}^{K1} \sum_{j=1}^{N} W_{i,j}\, X_{i,j}\right) \qquad (3.1)$$

$$\sigma^2{}_p = sgn\left(\sum_{p=1}^{K2} \sum_{i=1}^{K1} W_{p,i}\ \sigma_i^1\right) \qquad (3.2)$$

$sgn(x)$ *is a function shown in equation 3.3, which returns* $-1, 0$ *or* 1:

$$sgn(x) = \begin{cases} -1 & if\ x < 0 \\ 0 & if\ x = 0 \\ 1 & if\ x > 0 \end{cases} \qquad (3.3)$$

If the weighted sum over its inputs is negative then set $\sigma_i = -1$. *Hence, set* $\sigma_i = +1$, *if the weighted sum over its inputs is positive, or else if weighted sum is zero then it is set to,* $\sigma_i = 0$.

Step 4. *Compute the value of the final output neuron by computing multiplication of all values produced by* $K2$ *number of hidden neurons using the equation 3.4.*

$$\tau = \prod_{p=1}^{K2} \sigma_p^2 \qquad (3.4)$$

Step 5. *Sender utilizes its* 128 *initial weights as key for encryption of* T *variable (formerly stored in its memory)* $Encrypt_{Sender\_weight}(T)$.

Step 6. *Sender constructs a* SYN *frame and transmitted to the receiver for handshaking purpose in connection establishment phase.* SYN *usually comprises of several fields like* $Command\ Code, ID, Secret\ Seed$, *Sender output* $(\tau^{\ Sender}), Encrypt_{Sender\_weight}(T)$ *and* CRC *(Cyclic Redundancy Checker).*

Step 7. *Receiver performs integrity test after receiving the* SYN *frame and then receiver utilize its* 128 *weights as key for decryption of* $Encrypt_{Sender\_weight}(T)$ *that was received from the sender.*

Step 8. *If* $(Decrypt_{Receiver\_weight}(Encrypt_{Sender\_weight}(T)) = T$ *then networks are synchronized. Go to step 12.*

Step 9. *If* $(Decrypt_{Receiver\_weight}(Encrypt_{Sender\_weight}(T)) \neq T$*then receiver use the secret seed received from sender to produce the*

```
            receiver input vector$(X)$ identical to sender input vector $(X)$ and
            calculates the output $\tau^{Receiver}$ using step 3 and step 4.
Step 10.    If $(\tau^{Receiver} = \tau^{Sender})$ then performs the following steps
            Step 10.1   Receiver update their weights where $\sigma_k^{Sender/Receiver} = \tau^{Sender/Receiver}$
                        using any of the learning rules discussed in chapter 1 section 1.8.
            Step 10.2   At end of receiver's weights updation, the receiver sends
                        $ACK\_SYN$ to instruct the sender for updating the weights
                        using step 10.1.
            Step 10.3   Sender transmits
                        $Encrypt_{Sender\_updated\_weight}(T)$ to receiver.
            Step 10.4   Receiver checks
                if $(Decrypt_{Receiver\_updated\_weight}(Encrypt_{Sender\_updated\_weight}(T)) = T$
                        then networks are synchronized. Go to step 12.
            Step 10.5   Perform the following checking
                if $(Decrypt_{Receiver\_updated\_weight}(Encrypt_{Sender\_updated\_weight}(T)) \neq T$
                        then networks are still not synchronized. Go to step 10.1.
Step 11.    If $(\tau^{Receiver} \neq \tau^{Sender})$ then the receiver sends the message
            $NAK\_SYN$ to notify the sender. Go to step2.
Step 12.    Finally, the receiver sends the frame $FIN\_SYN$ to inform the sender to
            finish the synchronization phase.
```

3.2.1.1.6 Complexity Analysis

In DHLP synchronization and session key generation technique initialization of weight vector takes $(N \times K1 + K1 \times K2)$ amount of computations. For example, if $N = 2, K1 = 4$, $K2 = 2$ then total numbers of synaptic links (weights) are $(2 \times 4 + 4 \times 2) = 16$. So, it takes sixteen amount of computations. Computation of the hidden neuron outputs takes $(K1 + K2)$ computations. Where $K1$ and $K2$ are the number of hidden units in first and second layer respectively. Generation of N number of input vector for each $K1$ number of hidden neurons takes $(N \times K1)$ amount of computations. Computation of final output value takes unit amount of computation because it needs only a single operation to compute the value. Encryption of T variable using *Exclusive-OR* operation takes unit amount of computations. Decryption of T variable using *Exclusive-OR* operation also takes unit amount of computation. Check if $(Decrypt_{Receiver\_weight}(Encrypt_{Sender\_weight}(T)) = T$ or not, takes unit amount of computations. If $(Decrypt_{Receiver\_weight}(Encrypt_{Sender\_weight}(T)) \neq T$ then step 3 and 4 iterated again with its respective time complexity. Weight updating procedure takes place where $\sigma_k^{Sender/Receiver} = \tau^{Sender/Receiver}$ using any of the learning rules which takes $(no.of\ \sigma_k^{Sender/Receiver} = \tau^{Sender/Receiver})$ amount of computations.

In best case, sender's and receiver's arbitrarily chosen weight vectors are identical. So, networks are synchronized at initial stage do not needs to update the weight using learning rule. Here, $\big((N \times K1) + (N \times K1 + K1 \times K2) + (K1 + K2)\big)$ amount of computation is needed. So, in the best case the computation complexity can be expressed is in form of O(initialization of input vector + initialization of weight vector + Computation of the hidden neuron outputs).

If the sender's and receiver's arbitrarily chosen weight vector are not identical then in each iteration the weight vectors of the hidden unit which has a value equivalent to the pereceptron output are updated according to the learning rule. This scenario leads to average and worst case situation where I number of iteration to be performed to generate the identical weight vectors at both ends. So, the total computation for the average and worst case is $\big((N \times K1) + (N \times K1 + K1 \times K2) + (N \times K1 + K1 \times K2) + (K1 + K2)\big) +$ $\big(I \times (no.of\ \sigma_k^{Sender/Receiver} = \tau^{Sender/Receiver})\big)$ which is can be expressed as follws O(Time complexity in first iteration+(No. of iteration × No. of weight updation)).

3.2.1.1.7 DHLP Learning Mechanism

If the output bits are different for sender (A) and receiver (B) i.e.$\tau^A \neq \tau^B$, nothing is done. If $\tau^A = \tau^B = \tau$, only the weights of the hidden units with $\sigma_k^{A/B} = \tau^{A/B}$ will be updated using any of the learning rules discussed in chapter 1 section 1.8. For small L values Hebbian takes less synchronization steps than other two learning rules in the range of $2-4-2-5$ $(N-K1-K2-L)$ to $2-4-2-15$ but as the L value increases Hebbian rule takes more steps to synchronize than other two learning rules. Here, Anti-Hebbian rules takes less time than the other two learning rules in the range of $2-4-2-20$ to $2-4-2-30$. Random Walk outperform from $2-4-2-35$ and beyond that. The most vital findings is that if the synaptic depth i.e. weight range (L) is increased, the complexity of a successful attack grows exponentially, but there is only a polynomial increase of the effort needed to generate a key. So, increasing the L value security of the system can be increased.

3.2.1.2 Genetic Function based Simulated Annealing (SA) guided Fittest Keystream Generation

In the DHLPSCT a genetic function based Simulated Annealing (SA) guided approach is used to construct the keystream for encryption/decryption. Instead of this technique any other light weight encryption/decryption technique also used for exchanging message between sender and receiver with the help of DHLP synchronized network.

SA is a randomization technique for solving optimization problems. It is a technique to generate appropriate solutions to a large diversity of combinatorial optimization problems. SA is a good technique for finding near global optimal solutions for complex problems. Generating encryption/decryption keystream of good properties with very minimal resource requirements in wireless communication is always a complex problem. The keystream generators proposed can assist to solve the problem of getting stuck in local optima and to escort towards the global optimal solution.

The keystream (individual) in SA based technique are comprises of character sequence. Here, 'a' ... p' represents the number $0 \ldots 15$. For representing 128 bit long SA based keystream 32 characters are chosen randomly among 'a' ... p' characters, where each character represent a four bits binary number. So, one particular character may appear more than once in the sequence. Table 3.3 represents the different characters along with its decimal and binary representation.

Table 3.3
Character table of SA

Character	**Decimal Value**	**Binary Value**
a	0	0000
b	1	0001
c	2	0010
d	3	0011
e	4	0100
f	5	0101
g	6	0110
h	7	0111
i	8	1000
j	9	1001
k	10	1010
l	11	1011
m	12	1100
n	13	1101
o	14	1110
p	15	1111

Each individual which represents candidate keystream is strings of characters and are represented using binary sequence. These rules should be preserved during the generation of the initial population. The keystream is represented using character sequence 'a' ... 'p'. These letters represent the numbers $0 \ldots 15$. Thus, each letter is a sequence of four bits. The following are examples of the chromosomes having 32 characters i.e. 128 bits:

Chromosome 1: $melapekabrdojenhpgdjlncmaofhjlnc$
Chromosome 2: $ajckehpehgnbmdaofegolplacfbepfhj$
Chromosome 3: $cpdmjalobgejafnbhlicpdamhliejcle$
Chromosome 4: $jlakhfdpnoadflabpmfhnmanlokhgajb$

The fitness value is a measurement of the goodness of the keystream (individual), and it is used to control the application of the operations that modify a population. There are a number of metrics used to analyze keystream, which are keystream randomness, linear complexity and correlation immunity. Therefore, these metrics should be taken in account in designing keystream (individual), and they are in general hard to be achieved. Three factors are considered in the fitness evaluation of the keystream (individual). These are:

a. *Randomness of the generated keystream (individual)*
b. *Keystream (individual) period length*
c. *Keystream (individual) length*

a. *Randomness of the generated keystream (individual)* - The purpose of evaluation of randomness is to determine whether that number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence. The test assesses the closeness of the fraction of ones to ½, that is, the number of ones and zeroes in a sequence should be about the same. The equation 3.5 is used for the evaluation of keystream randomness using the frequency and serial tests, in which, n_w is the frequency of w in the generated binary sequence.

$$f_1 = |n_0 - n_1| + \left|n_{00} - \frac{SZ}{4}\right| + \left|n_{01} - \frac{SZ}{4}\right| + \left|n_{10} - \frac{SZ}{4}\right| + \left|n_{11} - \frac{SZ}{4}\right| \tag{3.5}$$

Fitness f_1 calculates the frequency of the bits. This function is derived from the fact that in the random sequence, *Probability (n_o) = Probability (n_1)* which checks frequency of 0 and 1 in a binary string and *Probability (n_{01}) = Probability (n_{11}) = Probability (n_{10}) = Probability (n_{00})* which checks the probability of occurrence of the pattern $00, 01, 10$ and 11 in a binary string.

b. *Keystream (individual) period length* - The focus of keystream (individual) period length evaluation is to determine the total number of zero and one runs in the entire sequence, where a run is an uninterrupted sequence of identical bits. A run of length k means that a run consists of exactly k identical bits and is bounded before and after with a bit of the opposite value. The purpose of this evaluation is to determine whether the number of runs of ones and zeros of various lengths is as expected for a random sequence. In particular, this test determines whether the oscillation between such substrings is too fast or too slow. $\frac{1}{2^i} \times n_r$ of the runs in the sequence are of length i, where n_r is the number of runs in the sequence. Thus, the following equation 3.6 represents the period length.

$$f_2 = \sum_{i=1}^{M} \left|\left(\frac{1}{2^i} \times n_r\right) - n_i\right| \tag{3.6}$$

Where M is maximum run length, and n_i is the desired number of runs of length i.

c. *Keystream (individual) length* - Another factor is considered in the evaluation of the fitness value which is the size of the candidate keystream (length of the individual).

Thus, the fitness function used to evaluate the chromosome x is given in equation 3.7, where $weight$ is a constant and sz *is* the key stream period length:

$$fitness(x) = \frac{SZ}{1+f_1+f_2} + \frac{weig\,ht}{lengt\,h(x)} \tag{3.7}$$

Crossover operation performs exchange of genetic information. It takes place between randomly selected parent chromosomes. Single point crossover is the most commonly used. In this technique single point crossover is performed with probability 0.6 to 0.9. Figure 3.8 shows the single point crossover operation having chromosome length of eight.

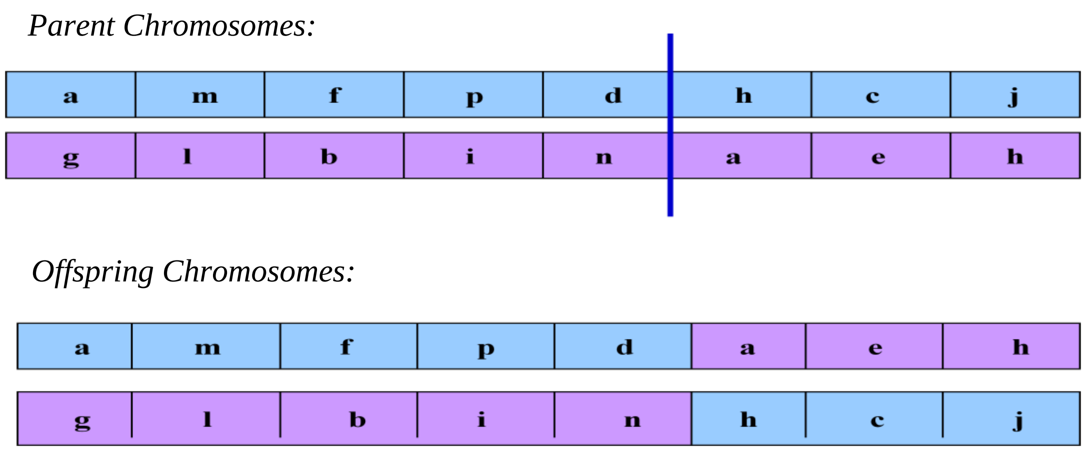


Figure 3.8: Single point Crossover operation

Mutation operation is a random alternation in the genetic structure. It introduces genetic diversity into the population. performs exchange of genetic information. It takes place between randomly selected parent chromosomes. In this technique mutation is performed with probability 0.001 to 0.01. Figure 3.9 shows the mutation operation having chromosome length eight.

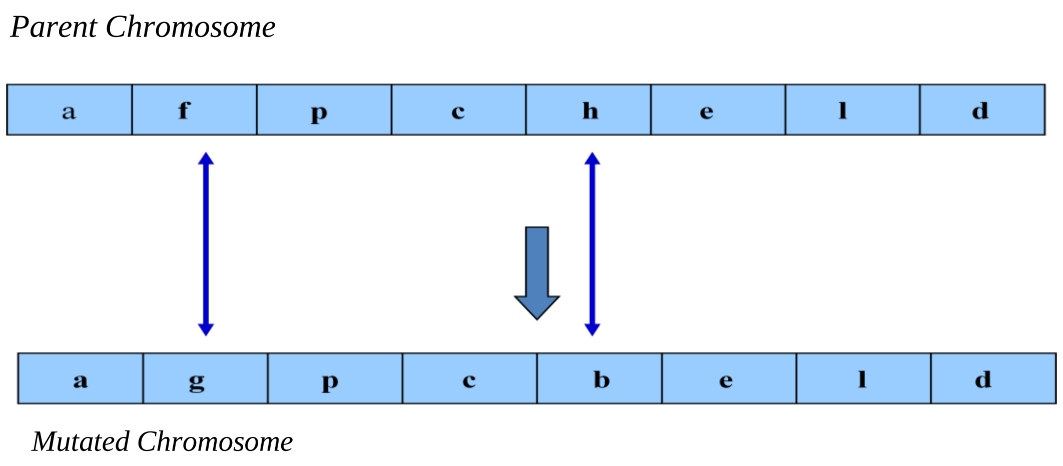


Figure 3.9: Mutation operation

The parameters used in this work have been set based on the experimental results, the parameter value that shows the highest performance has been chosen to be used in the implementation of this algorithm. Thus, the genetic operations used to update the population are single point crossover with probability pc (probability of crossover) = 0.6 to 0.9 and mutation with probability pm (probability of mutation) = .001 to 0.1. The selection strategy, used to select chromosomes for the genetic operations, is the binary tournament selection. The old population is completely replaced by the new population which is generated from the old population by applying the genetic operations. The maximum chromosome length is 256 bits. The run of this algorithm is stopped after a fixed number of iterations depend on resource available in wireless communication. The solution is the best keystream (individual) of the final iteration. The figure 3.10 shows the flowchart of SA based fittest keystream generation and section 3.2.1.2.1 presents the SA based fittest keystream generation algorithm.

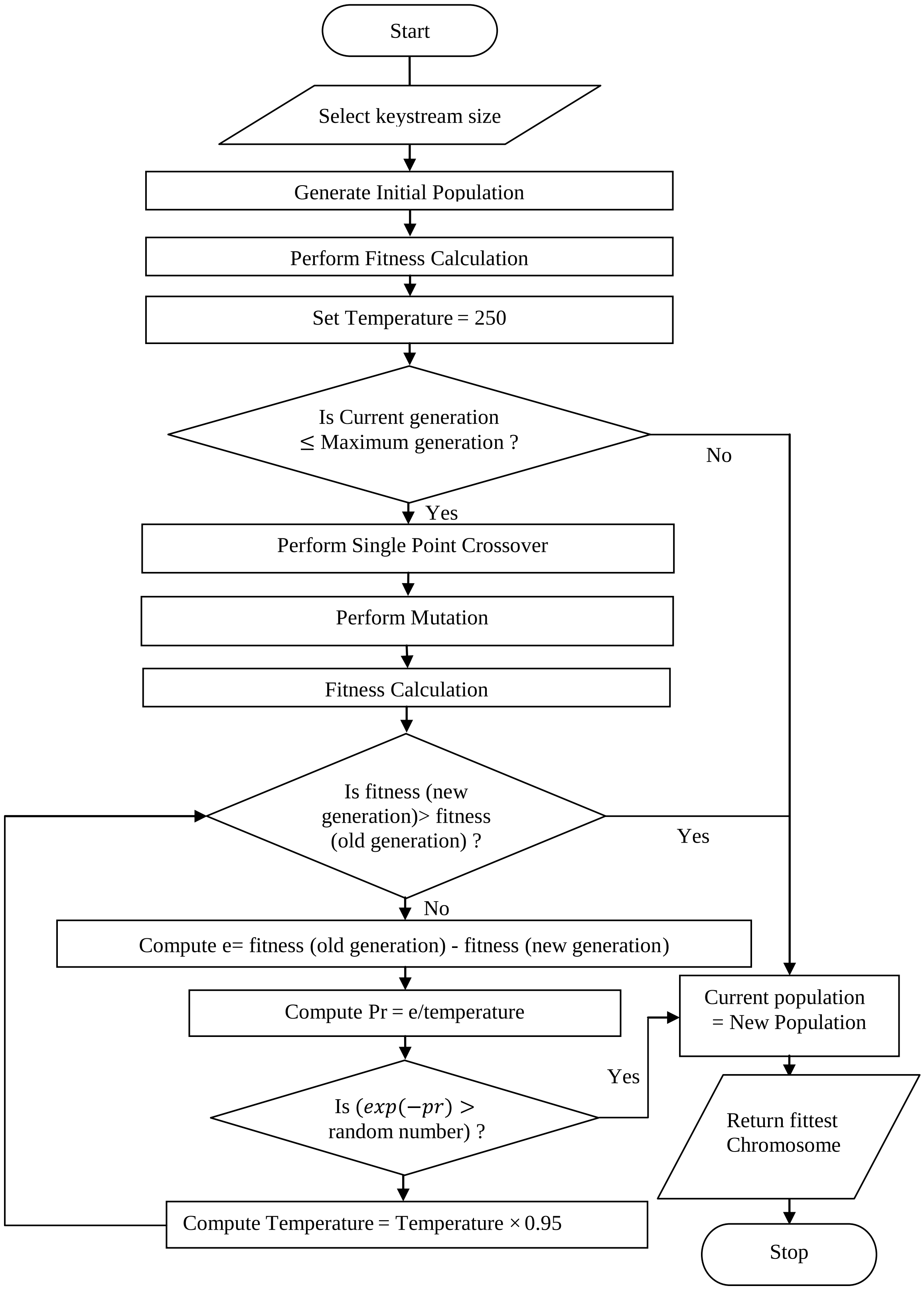


Figure 3.10: Flow chart of Simulated Annealing (SA) based fittest keystream generation

3.2.1.2.1 Simulated Annealing based Fittest Keystream Generation Algorithm

SA based fittest keystream generation algorithm takes length of the keystream and maximum number of iterations as an input. After complete iteration, algorithm generates the fittest keystream as an output. The maximum number of iterations depends on the resource available in wireless communication.

Input : *Length of the keystream, maximum number of iteration*

Output: *Simulated Annealing based best fittest keystream (individual) at the final iteration*

Method: *The process performs Simulated Annealing procedure on set of keystream and finally produces best fittest keystream.*

```
Step 1.   Generate the initial population ($pop$) randomly.
Step 2.   Evaluate the Population.
Step 3.   Set temperature:=$250$.
Step 4.   Perform the following steps until maximum number of generation reach.
    Step 4.1   Generate a new population ($pop1$) by applying crossover and mutation.
    Step 4.2   Evaluate the fitness of the new generated individual of $pop1$.
    Step 4.3   Calculate the averages of fitness values for $pop$ and $pop1$, $av$ and $av1$ respectively.
    Step 4.4   $If$ $(av1 > av)$ then replace the old population by the new one, $pop = pop1$. Else compute $e = av - av1$ and $Pr = e/Temp$. Hence, generate a random number ($rnd$) and check $if$ $(exp(-pr) > rnd)$ then assign $pop = pop1$.
    Step 4.5   Set $Temp = Temp \times 0.95$        (3.8)
Step 5.   Return the best chromosome of the final generation
```

The SA based fittest keystream is used to perform the encryption operation on the plaintext. The detail step of SA based encryption process is given in section 3.2.1.3.

3.2.1.3 Encryption Algorithm

Input : *Source file/source stream i.e. plaintext*

Output : *Encrypted file/encrypted stream i.e. cipher text*

Method : *The process operates on binary stream and generates encrypted bit stream through Simulated Annealing (SA) based encryption.*

Step 1. *Perform Exclusive-OR with Simulated Annealing (SA) generated* $128/192/256$ *bits key and the plaintext to form intermediate cipher text. If the size of the plaintext to be encrypted is larger than* $128/192/256$ *bits then triangle edge extension based keystream expansion strategy is perform to expand the SA based keystream and then expanded keystream is Exclusive-OR with the plaintext for forming the intermediate cipher text.*

Step 2. *Divide the outcomes of step 1 into variable blocks.*

Step 3. *Perform following operation as per equations 3.9 and 3.10 on each block until the source block itself is generated.*

$$s_0^j = s_0^{j-1} \tag{3.9}$$

$$s_i^j = s_{i-1}^2 \oplus s_i^{j-1} \tag{3.10}$$

Step 4. *Consider an intermediate* i^{th} *step during the process of forming the cycle as the encrypted block.*

Step 5. *Merge all the encrypted blocks of step 3.*

The details of triangle edge based keystream expansion in step 1 is discussed in section 3.2.1.3.1. Step 2 of the algorithm is used to divide the outcomes of step 1 in variable blocks. After that in step 3 a pair of *Exclusive-OR* operation is performed on each block for forming the cycle. In step 4, i^{th} step is considered as an encrypted block. Finally, in step 5 all the encrypted blocks of previous step is merged together to generate SA based encrypted text.

3.2.1.3.1 Triangle Edge Extension based Keystream Expansion Technique

If the size of the plaintext to be encrypted is larger than $128/192/256$ bits then triangle edge extension based keystream expansion strategy is performed to expand the keystream. Consider a keystream $K = k_0^0\, k_1^0\, k_2^0\, k_3^0\, k_4^0\, k_5^0 \dots k_{n-2}^0\, k_{n-1}^0$ of size n bits, where $k_i^0 = 0$ or 1 for $0 \le i \le (n-1)$. Now, starting from MSB (k_0^0) and the next-to-MSB (k_1^0), bits are pair-wise *Excusive-OR*, so that the first intermediate sub-keystream K^1 is expressed as $K^1 = k_0^1\, k_1^1\, k_2^1\, k_3^1\, k_4^1\, k_5^1 \dots k_{n-2}^1$ this is consisting of $(n-1)$ bits, where $k_j^1 = k_j^0 \oplus k_{j+1}^0$ for $0 \le j \le n-2$, $\oplus$ stands for the *Exclusive-OR* operation. This first intermediate sub-keystream K^1 is also then pair-wise *Excusive-OR* to generate the second intermediate sub-keystream $K^2 = k_0^2\, k_1^2\, k_2^2\, k_3^2\, k_4^2\, k_5^2 \dots k_{n-3}^2$, of length $(n-2)$. This process continues $(n-1)$ times to ultimately generate $k^{n-1} = k_0^{n-1}$, which is a single bit only. Thus the size of the first intermediate sub-stream is one bit less than the source sub-keystream; the size of each of the intermediate sub-keystreams starting from the second one is one bit less than that of the sub-keystream wherefrom it was generated; and finally the size of the final sub-keystream in the process is one bit less than the final intermediate sub-keystream. In this way intermediate sub-keystream $K^{j+1} = k_0^{j+1}\, k_1^{j+1}\, k_2^{j+1}\, k_3^{j+1} k_4^{j+1} k_5^{j+1} \dots k_{n-(j+2)}^{j+1}$ is generated from the previous intermediate sub-keystream $K^j = k_0^j\, k_1^j\, k_2^j\, k_3^j\, k_4^j\, k_5^j \dots k_{n-(j+1)}^j$. Figure 3.11 shows the keystream expansion triangle with different colors and figure 3.12 represents the left side and right side expanded keystream.

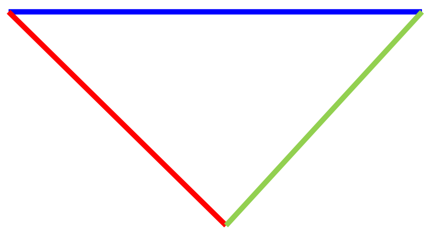


Figure 3.11: Triangle of different color sides, blue side represents the original key, red and green side represents the left and right side extended key

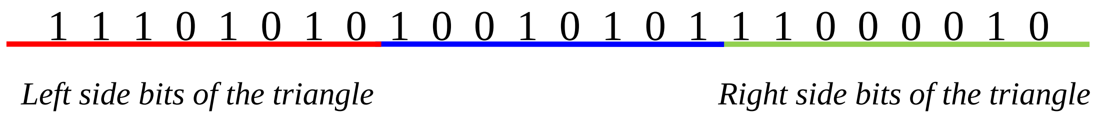

1 1 1 0 1 0 1 0 1 0 0 1 0 1 0 1 1 1 0 0 0 0 1 0
Left side bits of the triangle
Right side bits of the triangle

Figure 3.12: Expanded keystream

Bits of the left side of the triangle (i.e. 11101010) is generated at the front of the original key and bits of the right side of the triangle (i.e.11000010) is attached at the end. As per keystream expansion strategy the new expanded keystream will be three times longer than original one.

3.2.1.4 Session Key based Encryption

During final step of the technique a cascaded *Exclusive-OR* operation between DHLP synchronized session key and SA encrypted cipher text is performed to generate final encoded cipher text.

The decryption algorithm takes the cipher text as a binary stream of bits and perform first level of operation using DHLP generated synchronized session key to produce intermediate decrypted text. Finally, SA generated fittest keystream based decryption is performed on the intermediate decrypted text to regenerate the plaintext. The algorithm for the complete process is given in section 3.2.2.

3.2.2 DHLPSCT Algorithm at Receiver

Input : Encrypted file/encrypted stream i.e. cipher text

Output : Source file/source stream i.e. plaintext

Method : The process operates on encrypted binary stream and generates decrypted bit stream through DHLP guided Simulated Annealing (SA) based decryption operations.

Step 1. Perform cascaded Exclusive-OR operation between DHLP based session key and cipher text.

Step 2. Perform Simulated Annealing (SA) based decryption on the outcomes of the step 1 to regenerate starting combination i.e. plaintext.

Step 1 of the algorithm is discussed in section 3.2.2.1. Step 2 of the algorithm for performing SA based decryption is discussed in 3.2.2.2.

3.2.2.1 Session Key based Decryption

Initially cascaded *Exclusive-OR* operation between DHLP synchronized session key and cipher text is performed to produce session key decrypted text. Outcomes of this operation used as an input of decryption algorithm discussed in 3.2.2.2 to regenerate the plaintext.

In the decryption process the SA based cipher text is divided into blocks. A pair of *Exclusive-OR* operation based cycle decryption is performed on each block. After that all blocks are merged together. The SA generated keystream is use to *Exclusive-OR* with the merged blocks to regenerate the plaintext. The detail steps of decryption process is given in section 3.2.2.2.

3.2.2.2 Decryption Algorithm

Input : SA Encrypted file/ SA encrypted stream

Output : Source file/source stream i.e. plaintext

Method : The process operates on SA encrypted bit stream and regenerates the plaintext through SA based decryption.

Step 1. Divide the SA encrypted text into different blocks.

Step 2. Perform operation given in equation given in 3.11 and 3.12 upto $(P - i)$ steps on each block if the total number of iterations required to complete the cycle is P and the i^{th} step is considered to be the encrypted block.

$$s_0^j = s_0^{j-1} \tag{3.11}$$

$$s_i^j = s_{i-1}^2 \oplus s_i^{j-1} \tag{3.12}$$

Step 3. Merge outcomes of step 2.

Step 4. Check if the length of the SA based keystream is less than the length of outcomes of step 3 then perform triangle edge based keystream expansion method to enhance the length of the keystream. Otherwise, select the 128 *bit fittest keystream for decryption.*

Step 5. *Finally, perform Exclusive-OR operation between outcomes of step 3 and SA generated fittest encryption keystream of same length to produce the plaintext.*

3.3 Implementation

Consider Initial population size as 200 and randomly generated each keystream having 128 bits. The population gets evaluated with the help of fitness function using generations through a fitness technique which consist of number of statistical tests to examine whether the pseudorandom number sequences are sufficiently random or not.

On receipt of fittest generation the SA based keystream generation algorithm let generate the best fittest keystream having length of 128 bits. Let the binary form of 128 bits SA based keystream is
11111101/10101110/00011111/11011010/11010010/10000010/10101101/01100110/
01001111/11101001/00001110/11110101/01010010
Here "/" is used as the separator between successive bytes.

Consider the plaintext to be encrypted is "**SA Encryption**", binary representation of the ASCII value of plaintext is
01000001/01010011/00100000/01000101/01101110/01100011/01110010/01111001/011100
00/01110100/01101001/01101111/01101110

So, the plaintext size is 104 bits. As plaintext size is less than the size of the 128 bit SA based keystream, no need to perform keystream expansion operation.

On performing *Exclusive-OR* operation between plaintext and SA based keystream the intermediate cipher text is
10111100/11111101/00111111/10011111/10111100/11100001/11011111/00011111/
00111111/10011101/01100111/10011010/00111100

Perform *Exclusive-OR* based cycle formation operation on intermediate cipher text by dividing into seven segments having variable size like 16, 32, 8, 16, 16, 8, 8 bits respectively.

Following are the different segments constructed from S (intermediate encrypted text):

S_1 = 1011110011111101 (16 bits)

S_2 = 00111111100111111011110011100001 (32 bits)

S_3 = 11011111 (8 bits)

S_4 = 0001111100111111 (16 bits)

S_5 = 1001110101100111 (16 bits)

S_6 = 10011010 (8 bits)

S_7 =00111100 (8 bits)

Cycle formation operation is now performed on S_1, S_2, S_3, S_4, S_5, S_6, S_7 segments respectively. For each of the segments, an arbitrary intermediate stream segment is considered as the encrypted stream segment.

The formation of cycles for segments S_1 (1011110011111101) is shown below. After 16 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (1001001101010001) after iteration-6 considered as an encrypted segment for the segment S_1.

1011110011111101 → 1101011101010110^1→1001101001100100^2→1110110001000111^3→ 1011011110000101^4→1101101011111001^5→1001001101010001^6→1110001001100001^7→ 1011110001000001^8→1101011110000001^9→1001101011111110^{10}→1110110010101011^{11} →1011011100110010^{12}→1101101000100011^{13}→1001001111000010^{14}→ 1110001010000011^{15}→ 1011110011111101^{16}

The formation of cycles for segments S_2 (00111111100111111011110011100001) is shown below. After 32 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (00110000010010000010101110001010) after iteration-22 considered as an encrypted segment for the segment S_2.

00111111100111111011110011100001→$00101010111010101101011101000001^1$→ $00110011010011001001101001111110^2$→$00100010011101110001001110101011^3$→ $00111100010110100001110100110010^4$→$00101000011011000001011000100011^5$→ $00110000010010000001101111000010^6$→$00100000011100000001001010000011^7$→ $00111111101000000001110011111101^8$→$00101010110000000001011101010110^9$→

00110011011111111110010110011011^{10}→00100010010101010100011011101101^{11}→
00111100011001100111101101001001^{12}→00101000010001000101001001110001^{13}→
00110000011110000110001110100001^{14}→00100000001010000010000101100001^{15}→
00111111100111111000001101111110^{16}→00101010111010101111110110101011^{17}→
00110011010011001010100100110010^{18}→00100010011101110011000111011100^{19}→
00111100010110100010000101101000^{20}→00101000011011000011111001001111^{21}→
00110000001001000001010111000101O^{22}→0010000000111000000011001011110011^{23}→
001111111010000000010001101011101^{24}→001010101100000000011110110010110^{25}→
001100110111111111101011011100100^{26}→00100010010101010110010010111000^{27}→
00111100011001100100011100101111^{28}→00101000010001000111101000110101^{29}→
00110000011110000101001111011001^{30}→001000000010100000110001010010001^{31}→
00111111100111111011110011100001^{32}

The formation of cycles for segments S_3 (11011111) is shown below. After 8 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (11010010) after iteration-4 considered as an encrypted segment for the segment S_3.
11011111→10010101^{1}→11100110^{2}→10111011^{3}→11010010^{4}→10011100^{5}→11101000^{6}→
10110000^{7}→11011111^{8}

The formation of cycles for segments S_4 (0001111100111111) is shown below. After 16 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment I_{47} (0001000010110000) after iteration-7 considered as an encrypted segment for the segment S_4.
0001111100111111→0001010111010101^{1}→0001100101100110^{2}→0001000110111011^{3}→
0001111011010010^{4}→0001010010011100^{5}→0001100011101000^{6}→0001000010110000^{7}→
0001111100100000^{8}→0001010111000000^{9}→0001100101111111^{10}→0001000110101010^{11}
→0001111011001100^{12}→0001010010001000^{13}→0001100011110000^{14}→
0001000010100000^{15}→0001111100111111^{16}

The formation of cycles for segments S_5 (1001110101100111) is shown below. After 16 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (1011101010000100) after iteration-6 considered as an encrypted segment for the segment S_5.

1001110101100111→1110100110111010^{1}→1011000100101100^{2}→1101111000110111^{3}→ 1001010000100101^{4}→1110011111000110^{5}→1011101010000100^{6}→101001100000111^{7}→ 1001110111111010^{8}→1110100101010011^{9}→1011000110011101^{10}→1101111011101001^{11} →1001010010110001^{12}→1110011100100001^{13}→1011101000111110^{14}→ 1101001111010100^{15}→1001110101100111^{16}

The formation of cycles for segments S_6 (10011010) is shown below. After 8 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (11100010) after iteration-5 considered as an encrypted segment for the segment S_6.

10011010→11101100^{1}→10110111^{2}→11011010^{3}→10010011^{4}→11100010^{5}→10111100^{6}→ 11010111^{7}→10011010^{8}

The formation of cycles for segments S_7 (00111100) is shown below. After 8 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (00100000) after iteration-3 considered as an encrypted segment for the segment S_7.

00111100→00101000^{1}→00110000^{2}→ 00100000^{3}→ 00111111^{4}→00101010^{5}→00110011^{6}→ 00100010^{7}→00111100^{8}

On completion of the cycle formation technique on each segment as indicated above, seven intermediate blocks one for each segment are considered as the encrypted segments. On merging the above seven encrypted segments following SA based encrypted text is generated.

10010011/01010001/00110000/01001000/00101011/10001010/11010010/00010000/101100 00/10111010/10000100/11100010/00100000

Consider the Double Hidden Layer Perceptron (DHLP) synchronized 128 bits session key is 00110010/11101001/10111000/11000101/00011001/01000111/00010000/01010100/110011 00/00011010/01101111/00100101/01000111/11001110/10101100/00101011

Following is the session key encrypted final cipher text produce on performing *Exclusive-OR* operation between SA based encrypted text and DHLP synchronized session key.

10100001/10111000/10001000/10001101/00110010/11001101/11000010/01000100/01111100/10100000/11101011/11000111/01100111.

3.4 Security Analysis

In DHLPSCT, sender (A) and receiver (B) do not share a common secret but use their indistinguishable weights as a secret session key. The fundamental conception of DHLP based key exchange protocol focuses mostly on two key attributes of DHLP. Firstly, two nodes coupled over a public channel will synchronize even though each individual network exhibits disorganized behavior. Secondly, an outside network, even if identical to the two communicating networks, will find it exceptionally difficult to synchronize with those parties, are communicating over a public network. An attacker (E) who knows all the particulars of the algorithm and records through this channel finds it thorny to synchronize with the parties, and hence to calculate the common secret key. Synchronization by mutual learning (A and B) is much quicker than learning by listening (E). Usual cryptographic systems, improve the safety of the protocol by increasing of the key length. In the case of DHLP, security is improved by increasing the synaptic depth L of the DHLP. The communication of DHLPs has been discussed as a substitute concept for secure symmetric key exchange. For the key exchange protocol eavesdropping attacks can all be made arbitrarily costly and thus can be defeated by simply increasing the parameter synaptic depth (L) of the DHLP i.e. weight range. The security increases proportional to L^2 while the probability of a successful attack decreases exponentially with L. The approach is thus regarded computationally secure with respect to these attacks for sufficiently large L. For a brute force attack using $K1$ hidden neurons in layer 1, $K2$ hidden neurons in layer 2, $K1 \times N$ input neurons and boundary of weights L, gives $(2L+1)^{(K1\times N+K1\times K2)}$ possibilities. For example, the DHLP configuration $K1 = 3, K2 = 3, L = 3$ and the value of $N = 100$ gives $(2 \times 3 + 1)^{(3\times 100+3\times 3)}$ key possibilities, making the attack unfeasible. E could start from all of the $(2L+1)^{(K1\times N+K1\times K2)}$ initial weight vector and calculate the ones which are consistent with the input/output sequence. It has been shown, that all of these initial states move towards the same final weight vector, the key is unique. This is not true for simple perceptron the most unbeaten cryptanalysis has two supplementary ingredients first; a group of attacker is used. Second, E makes extra training steps when A and B are quiet. So increasing synaptic depth L of the DHLP we can make our DHLP safe. The main difference between the partners and the attacker in DHLP is that A and B are able to influence each other by communicating their output bits and while E can only listen to these messages. Of course, A and B use their advantage to select suitable input vector for adjusting the weights. This finally leads to different synchronization times for partners and attackers. Bidirectional interaction of the partners confirm that the security of DHLP key generation. Both A and B uses a secret seed for generating identical input vector. Whereas attacker does not know this secret seed state. By increasing synaptic depth (weight range) average synchronize time will be increased polynomial time. But success probability of attacker will be drop exponentially Synchronization by mutual learning is much faster than learning by adapting to example generated by other network. As E can't influence A and B at the time they stop transmit due to synchronization. Only one weight get changed where, $\sigma_i = \tau$. So, difficult to find weight for attacker to know the actual weight without knowing internal representation it has to guess. It is important to note, though, that all of the existing attacks refer to a non-authenticated key exchange, in which a MITM-attack on the symmetric principle is possible as well. Generally, for mutual authentication the two parties engage in a conversation to increase their confidence that it is a specific other party with whom they communicate. Additionally exchanging a new secret (session) key leads to authenticated key exchange. Assume that all communication among interacting parties is under the adversary's control. In particular, the adversary can read the bit packages produced by the parties, provide her own bit packages to them, modify bit packages before they reach their destination, and delay bit packages as well as replay them. The scheme cannot be reduced to number-theoretic hardness assumptions. Yet, a proof that the authentication is sound is given, as well as a proof of its security with regard to eavesdropping-attacks that also use DHLPs. The structure of the network, the involved computations producing the output $\tau^{A/B(t)}$. The different initial preliminary keys $W_{ij}^{A,B}(t_0)$ of the two parties are the secret information. If they were public, the resulting final keys could simply be calculated (by an adversary), because all further computations are completely deterministic. An implicit solution to include authentication into the DHLP key exchange protocol bases on the simple but strong fact, that two interacting parties A and B which have different input vector $x^{\mathrm{A}}(\mathrm{t}) \neq x^{\mathrm{B}}(\mathrm{t})$; $x^{\mathrm{A}}(\mathrm{t}), x^{\mathrm{B}}(\mathrm{t}) \in \{0,1\}^{K1\times N}$ cannot become synchronous. Equation 3.13 shows two DHLPs A and B are synchronous at iteration t_{s} when all their weights are identical.

$$W_{ij}^{A}(t_s) = W_{ij}^{B}(t_s) \qquad \forall\, i, j \tag{3.13}$$

Equation 3.14 shows two corresponding hidden units of two DHLPs A and B are synchronous at iteration t_{s} when all their weights (components) are identical:

$$W_{ij}^{A}(t_s) = W_{ij}^{B}(t_s) \quad \forall j\ (i\ fixed) \tag{3.14}$$

Once a summation unit is synchronous (i.e. it is in an identical state with the other party) it remains synchronous for all subsequent iterations. Two corresponding summation units σ_i^A and $\sigma_i^B$of two DHLPs A and B that have identical internal states $W_{ij}^{A}(t) = W_{ij}^{B}(t)$ $\forall j\ (i\ fixed)$ at an arbitrary iteration t_s (that are synchronous at an arbitrary iteration t_s) remain synchronous for all $t > t_s$ when having the same inputs and applying the same learning rule and bounding operation. Consider the subsequent iteration at time $t_s + 1$. Formally, two cases can be distinguished, first is no adaptation at iteration $t_s + 1$. If $\tau^A(t_s+1) \neq \tau^B(t_s+1)$, no adaptation is performed and in this trivial case $W_{ij}^{A}(t_s+1) = W_{ij}^{A}(t) = W_{ij}^{B}(t) = W_{ij}^{A}(t_s+1)\ , \forall j\ (i\ fixed)$ i.e. the summation units of both DHLPs remains synchronous. Second is adaptation at iteration $t_s + 1$.

If $\tau^A(t_s+1) = \tau^B(t_s+1) = \sigma_i^A(t_s+1) = \sigma_i^B(t_s+1)$, an adaptation is performed and each component j of the weight vector of hidden unit i of both parties will be changed according to the same learning rule given in equation 3.15 for the parties A and B.

$$W_{ij}^{A/B}(t_s+1) = W_{ij}^{A/B}(t_s) + \ \tau^{A/B}(t_s+1) X_{ij}^{A/B}(t_s+1)\ , \forall j\ (i\ fixed) \tag{3.15}$$

The adaptation is performed in the same direction given in equation 3.16 and 3.17.

$$\tau^A(t_s+1) = \ \tau^B(t_s+1) \tag{3.16}$$

$$X_{ij}^{A}(t_s+1) = X_{ij}^{B}(t_s+1), \forall i, j \tag{3.17}$$

Thus $W_{ij}^{A}(t_s+1) = W_{ij}^{B}(t_s+1)$,$\forall j\ (i\ fixed)$– the summation units remain synchronous. Proposed mechanism has been compared with existing key exchange method like

Diffie-Hellman Key exchange. In this existing method attackers can reside middle of sender and receiver and tries to capture all the information transmitting from both parties Intruders can act as sender and receiver simultaneously and try to steal secret session key at the time of exchanging key via public channel.

In DHLPSCT, problem of Man-In-The-Middle (MITM) attack of Diffie-Hellman Key exchange has been resolved by DHLP based session key generation technique where both sender and receiver uses an identical DHLP. In each session sender's and receiver's both DHLPs are start synchronization by exchanging some control frames. During synchronization process message integrity test and synchronization test has been carried out. Synchronized identical weight vector forms the session key on synchronization for a particular session. So, in this technique session key instead of transferring through public channel DHLP based synchronization process is carried out and outcomes of this used as a secret session key for that entire session. That actually helps to get rid of famous Man-In-The-Middle attack. The following standard attacks are considered to ensure the robustness of the DHLPSCT.

- *Cipher text only Attack:* The DHLPSCT nullifies the success rate of this attack by producing a completely random SA based encryption/decryption keystream. The strength of resisting exhaustive key search attack relies on a large key space. Initially, SA based large keystream is used to encrypt the plaintext after that, outcomes of this passes through *Exclusive-OR* based cycle formation based encryption process and DHLP generated session key based encryption. So, cipher text produces by this technique is mathematically difficult to break. Thus a hacker has to try all such keystreams to find an appropriate one. This method makes it difficult for the hacker to find out the keystream used for encryption. The technique helps to generate long period of random keystreams along with no obvious relationship between the individual bits of the sequence. Also the generated keystreams are of large linear complex. Finally keystream have high degrees of correlation immunity. Thus it is practically difficult to perform a brute-force search in a key-space.

- *Known Plaintext Attack:* DHLPSCT offers better floating frequency of characters and in SA based encryption technique cycle formation operation also enhance the security of the technique. So, known plaintext attack is difficult in this technique.
- *Chosen Plaintext Attack:* The technique passes the frequency (monobit) test, runs test, binary matrix rank test and in each session a fresh DHLP based session key is used for encryption which confirms that chosen plaintext attack is very difficult in this technique.
- *Chosen Cipher text Only Attack:* The technique passes the discrete Fourier transform test, approximate entropy test, overlapping (periodic) template matching test which confirms that chosen plaintext attack is difficult in this technique.
- *Brute Force Attack:* In DHLPSCT, security is improved by increasing the synaptic depth L of the DHLP. The security increases proportional to L^2 while the probability of a successful attack decreases exponentially with L. The approach is thus regarded computationally secure with respect to these attacks for sufficiently large L. For a brute force attack using $K1$ hidden neurons in layer 1, $K2$ hidden neurons in layer 2, $K1 \times N$ input neurons and boundary of weights L, gives $(2L+1)^{(K1 \times N + K1 \times K2)}$ possibilities. For example, a DHLP with the configuration $K1 = 3$, $K2 = 3$, $L = 3$ and $N = 100$ produces $(2 \times 3 + 1)^{(3 \times 100 + 3 \times 3)}$ amount of session key which makes the attack difficult.

3.5 Discussions

DHLPSCT is simple and easy to implement in various high level language. The test results show that the performance and security provided by the DHLPSCT is good and comparable to standard technique. The security provided by the technique is comparable with other techniques. To enhance the security of the technique, DHLPSCT offers changes of some parameters randomly in each session. To generate the secret session key secret seed get exchanged between sender and receiver. This technique has a unique ability to construct the secret key at both sides using this DHLP synchronization. Since the encryption and decryption times are much lower, so processing speed is very high. The method takes minimum amount of resources which is greatly handle the resource constraints criteria of wireless communication. DHLPSCT outperform than existing TPM, PPM and method proposed in chapter 2. No platform specific optimizations were done in the actual implementation, thus performance should be similar over varied implementation platform. The whole procedure is randomized, thus resulting in a unique process for a unique session, which makes it harder for a cryptanalyst to find a base to start with. This technique is applicable to ensure security in message transmission in any form and in any size in wireless communication.

Some of the salient features of DHLPSCT are summarized as follows:

a) *Session key generation and exchange – Identical session key can be generate after the tuning of DHLP in both sender and receiver side. So, no need to transfer the whole session key via vulnerable public channel.*

b) *Degree of security – The technique does not suffers from cipher text only attack, known plaintext attack, chosen plaintext attack, chosen cipher text only attack, brute force attack and attacks during DHLP synchronization process.*

c) *Variable block size – Encryption algorithm can work with any block length and thus not require padding, which result identical size of files both in original and encrypted file. So, DHLPSCT has no space overhead.*

d) *Variable key –* $128/192/256$ *bits DHLP based session key and* $128/192/256$ *bits SA based encrypted keystream with high key space can be used in different sessions. Since the session key is used only once for each transmission, so there is a minimum time stamp which expires automatically at the end of each transmission of information. Thus the cryptanalyst may not be able guess the session key for that particular session.*

e) *Complexity – The technique has the flexibility to adopt the complexity based on infrastructure, resource and energy available for computing in a node or mesh through wireless communication. So, the DHLPSCT may be suitable in wireless communication.*

f) *Non-homogeneity – Measures of central tendency, dispersion and Chi-Square value have been performed between the source and corresponding cipher streams generated using DHLPSCT. Measures indicate that the degree of non-homogeneity of* *the encrypted stream with respect to the source stream is good. This technique has a better Chi-Square value than technique proposed in chapter 2.*

g) *Floating frequency – In the DHLPSCT it is observed that floating frequencies of encrypted characters are indicates the high degree of security for the technique. This technique has a better floating frequency than technique proposed in chapter 2.*

h) *Entropy – The entropy of encrypted characters is near to eight which indicate the high degree of security of technique. This technique also has a better entropy value than technique proposed in chapter 2.*

i) *Correlation – The cipher stream generated through proposed technique is negligibly correlated with the source stream. Therefore the DHLPSCT may effectively resist data correlation statistical attack.*

j) *Key sensitivity – The technique generates an entirely different cipher stream with a small change in the key and technique totally fails to decrypt the cipher stream with a slightly different secret session key.*

k) *Security and performance trade-off – The technique may be ideal for trade-off between security and performance of light weight devices having very low processing capabilities or limited computing power in wireless communication.*

Chapter 4

Chaos based Double Hidden Layer Perceptron Synchronized Cryptographic Technique (CDHLPSCT)

4.1 Introduction

In this chapter a novel soft computing based cryptographic technique CDHLPSCT, on synchronization of Chaos based two Double Hidden Layer Perceptron (CDHLP), has been proposed. The DHLPSCT technique proposed in chapter 3 have some drawbacks like secret seed values used in the generation of identical input vector has to be transmitted to the other party via public channel in the SYN frame in each iteration. This significantly increases the synchronization time. Also for ensuring the security this parameters should not be transmitted via public channel. Furthermore, till now all the synchronization techniques devised in previous chapters concentrated only in session key generation mechanism by tuning the sender and receiver. But the process of generating session keys does not guarantee the information security. Because, any attacker can also synchronize with an authorized device, because the protocol is a public knowledge. Proposed CDHLPSCT of this chapter eliminates all the above stated drawbacks of the DHLPSCT in chapter 3, KSOFMSCT in chapter 2 and existing TPM and PPM.

Here, Chaos based Double Hidden Layer Perceptron (CDHLP) synchronization mechanism has been introduced between sender and receiver where instead of transmitting the secret seed values used in the generation of identical input vector in each iteration, Chaos is use to generate identical random seed value for generating common input vector at the both ends.[202] This improves the security of the technique. Also to ensure that only entities authorized have access to information, authentication service has been introduced in this chapter. The function of the authentication service is to ensure the recipient that the message is from the source that it claims. CDHLPSCT performs secret keys authentication where both entities must have a common secret code. This newly introduced technique significantly reduces the risk and increases the security.

On the completion of the tuning phase identical session keys is generated at the both end with the help of synchronized CDHLP. This synchronized network can be used for transmitting message using any light weight encryption/decryption technique with the help of session key of the synchronized network. To illustrate the cryptographic technique using CDHLP in wireless communication one of the simple and secure encryption/decryption technique has been presented. A plaintext is considered as a stream of binary bits. Genetic Algorithm (GA) guided enciphering technique[203] with the help of CDHLP tuned session key is used to generate the cipher text. The plaintext is regenerated from the cipher text using same technique with the help of CDHLP tuned session key at the receiver.

Section 4.2 represents a description of proposed technique in detail. Section 4.3 deals with the implementation of the proposed cryptographic technique. Section 4.4 discussed the security issues related to the proposed technique. Discussions are presented in section 4.5.

4.2 The Technique

The technique performs the CDHLP based synchronization for generation of secret session key at both ends. This synchronized session key of the tuned network is used for the transmission of secured message through wireless network with the help of any light weight encryption/decryption algorithm. To illustrate the cryptographic technique in wireless communication one of the simple and secure encryption/decryption technique has been proposed, where plaintext (i.e. the input file) is considered as a stream of binary bits, which is encrypted using GA generated fittest encryption/decryption keystream. The session key based on CDHLP is used to encrypt intermediate output which produces final cipher text. Identical CDHLPs are used to tune the sender and receiver to generate the secret session key at both ends. Chaos helps to generate identical secret seed values (z) at the both end using chaos synchronization. This identical seed value is used to generate identical input vector. For generating common seed value (z) in sender and receiver side two chaotic system synchronized with each other by exchanging parameters (σ, b, r, x_1, y_2 and z_2) between sender and receiver. Some of the parameter which takes major roles to form the common seed value (z) does not get transmitted via public channel, sender keeps these parameters secret. Two CDHLPs at sender and receiver start with common seed value (z) based identical input vector and completely anonymous random weight vector. In each time both CDHLPs at both end compute their final output based on input and weight vector, and communicate to each other. If both are be in agreement on the mapping between the present input and the output, their weights are updated according to an appropriate learning rule. After synchronization weight vector of both CDHLPs become identical. This identical weight vector forms the session key.

CDHLPSCT performs secret session keys authentication where both sender and receiver must have a common secret code. Here, two secret codes are used, called RSC (Receiver Secret Code) and SSC (Sender Secret Code).

In GA based keystream generation technique LFSR (Linear Feedback Shift register) are used to generate the initial population of keystream (i.e. chromosomes). Each chromosome that represents candidate keystream is strings of characters 'a' ... 'p' along with LFSR function (SR), bitwise OR, bitwise AND, bitwise *Exclusive-OR* and these are represented using prefix notation.The letters 'a' ... 'p' represent the numbers $0\ldots15$. Thus, each letter is a sequence of four bits. The number of these letters must be even, because half of them are used for the initial state of the LFSR, and the second half for the feedback function of LFSR. GA based keystream generation algorithm first initialize the keystream (chromosome) size (maximum 300 characters for each chromosome), select a value up to which the algorithm will iterate i.e. the maximum number of generation depend on the resource available at the time of wireless communication and generate initial population of keystream randomly having size of 200 keystream. Then for each keystream in the population fitness is evaluated. Fitness values of each keystream in the population are calculated depending on the randomness of the generated keystream, keystream period length and keystream length. The process checks whether the current generation number is less or equal to the maximum number of generations, if so, then the process performs uniform crossover and dynamically adjust the crossover probability and then mutation operation is performed through dynamically adjusted mutation probability. Newly generated chromosomes through genetic operations like uniform crossover and mutation formed a new population. So, fitness value of each chromosome in this newly formed population is evaluated. Fitness calculation is performed again for the newly generated keystream and then checks whether the fitness value of new generation is higher than the fitness value of old generation, if so, then the process set the new population as the current population. The process again check whether the current generation number is less or equal to the maximum number of generations, if the condition is still satisfied then operation is repeated until the best fittest keystream is found or maximum number of generation is reached whichever is earlier. If the length of the plaintext to be encrypted is grater then the length of GA based keystream then square edge based keystream expansion method is used to extend the length of the keystream. Stream of plaintext is then encrypted using the GA based keystream/expanded keystream. Finally a cascaded *Exclusive-OR* operation is performed between GA encrypted text and the CDHLP based session key to generate final cipher text.

Receiver has same CDHLP synchronized session key. This session key is used to perform first step of the deciphering technique. In the next step, GA guided keystream based deciphering operation is performed to regenerate the plaintext.

The CDHLPSCT does not cause any storage overhead. This greatly handles the resource constraints criteria of wireless communication. A comparison of proposed technique with previously proposed technique in chapter 3, chapter 2, existing Tree Parity Machine (TPM), Permutation Parity Machine (PPM), and industry accepted AES, RC4, Vernam Cipher, Triple DES (TDES) and RSA have been done. Analyses of results are given in chapter 7.

In CDHLPSCT, encryption algorithm takes the plaintext as a binary stream of bits which is encrypted using GA generated fittest keystream. Chaos based DHLP synchronized session key is used to further encrypt the GA encoded text to produce final cipher text. The algorithm for the complete process is given in section 4.2.1.

4.2.1 CDHLPSCT Algorithm at Sender

Input : Source file/source stream i.e. plaintext

Output : Encrypted file/encrypted stream i.e. cipher text

Method : The process operates on binary stream and generates encrypted bit stream through CDHLP guided Genetic Algorithm (GA) based encryption operations.

Step 1. Perform tuning of sender's and receiver's CDHLP to generate common secret session key.

Step 2. Generates GA based fittest encryption keystream.

Step 3. Perform GA based encryption operation on the plaintext.

Step 4. Perform cascaded Exclusive-OR operation between CDHLP based session key and outcomes of step 3.

Step 1 of the algorithm generate common session key through synchronization of CDHLP at both end. The detailed step is discussed in section 4.2.1.1. Step 2 of the algorithm generates GA based fittest encryption keystream. The detailed description of the process is given in section 4.2.1.2. Algorithm for performing GA based encryption operation (step 3) on the plaintext is discussed in 4.2.1.3. The technique of cascading encryption process (step 4) which takes the intermediate output generated in step 3 is given in details in section 4.2.1.4.

4.2.1.1 Chaos based Double Hidden Layer Perceptron (CDHLP) Synchronization and Session Key Generation

A novel Chaos based scheme is introduce to generate identical seed values at the both sender and receiver end at the initial phase. There are several authentication methods, differentiated mainly by the use of secret-keys or public-keys. CDHLPSCT performs secret keys authentication where both entities must have a common secret code. Here, two secret codes are used, called RSC (Receiver Secret Code) and SSC (Sender Secret Code) which are known to both parties. This codes are used to examine whether the authenticate parties are involved in synchronization or not. The CDHLP synchronization technique use the already discussed architecture and parameters of DHLP synchronization technique of chapter 3 with some additional parameters for chaos synchronization.

Common seed has been generated by synchronization of two chaotic systems using Pecora and Caroll (PC) method[204]. In this technique tuning parameters (σ, b, r, x_1, y_2 and z_2) are being exchanged between sender and receiver for synchronization purpose. Some of the parameter which takes major roles to form the seed does not get transmitted via public channel, sender keeps these parameters secret. This way of handling parameter passing mechanism prevents any kind of attacks during exchange of parameters like sniffing, spoofing, phishing, or Man-In-The-Middle (MITM) attack.In this technique PC method is applied on the Edward Lorenz chaotic system[205] to describe three equations 4.1, 4.2 and 4.3 to form two secure sub systems i.e. initiator (sender) and responder (receiver).

$$\dot{x} = \sigma(x - y) \tag{4.1}$$

$$\dot{y} = rx - y - xz \tag{4.2}$$

$$\dot{z} = xy - bz \tag{4.3}$$

The main objective of this technique is coordination of two chaotic systems. This is refers to a method where two (or more) chaotic systems (either identical or non identical) regulate a given property of their motion to a similar performance owing to a pairing or to a forcing (periodical or noisy). Proposed technique use the PC method to assume a dynamical system characterized by the state space equation 4.4.

$$\dot{\bar{x}} = f(\bar{x}) \tag{4.4}$$

Where $\dot{x} = (x_1, x_2, \dots, x_n)$ the system vector and f is an arbitrary mapping. Further system is decomposed into two following sub system represented by equation 4.5 and 4.6.

$$\left.\begin{aligned}\dot{\bar{u}} &= f(\bar{u}, \bar{v})\\ \dot{\bar{v}} &= g(\bar{u}, \bar{v})\end{aligned}\right\} driver \tag{4.5}$$

$$\dot{\bar{w}} = h(\bar{u}, \bar{w})\} \, response \tag{4.6}$$

Driver signal $\bar{u}(t)$ is drives the response system. Using Lyapunov exponents of the response system along with consideration that the action of the driver is negative, Chaotic synchronization can be possible between these driver and response system. From the following equations two secure sub systems i.e. initiator and responder respectively can be defined by applying the PC method on the Lorenz system. The initiator (sender) ($\dot{x}_1$, $\dot{z}_1$), can be defined by equations 4.7 and 4.8.

$$\dot{x}_1 = \sigma(x_1 - y) \tag{4.7}$$

$$\dot{z}_1 = x_1 y - b z_1 \tag{4.8}$$

The responder (receiver) ($\dot{y}_2$, $\dot{z}_2$) can be defined by equations 4.9 and 4.10.

$$\dot{y}_2 = rx - y_2 - x z_2 \tag{4.9}$$

$$\dot{z}_2 = x y_2 - b z_2 \tag{4.10}$$

From the above two equations 4.9 and 4.10 it can be observed that the Lyapunov exponents of the system are both negative. The sender and receiver response subsystems are driven by the signal $y(t)$ and $x(t)$. When t trends to infinity value of $|z_2 - z_1|$ trends to zero. After synchronization of both the system a common value of both the system is obtained.

Figure 4.1 shows that in this technique at first sender initialize the value of σ and b, after that value of b is send to the receiver. Receiver initializes the value of r and send to the sender. Sender initially generates random value for the point x_1 and $z_{1.}$ Sender sends x_1 to receiver. Receiver initially generates random value for the point y_2, z_2 and sends to the sender. So, receiver receives the value of b and x_1 from the sender and sender receives y_2, z_2, from the receiver.

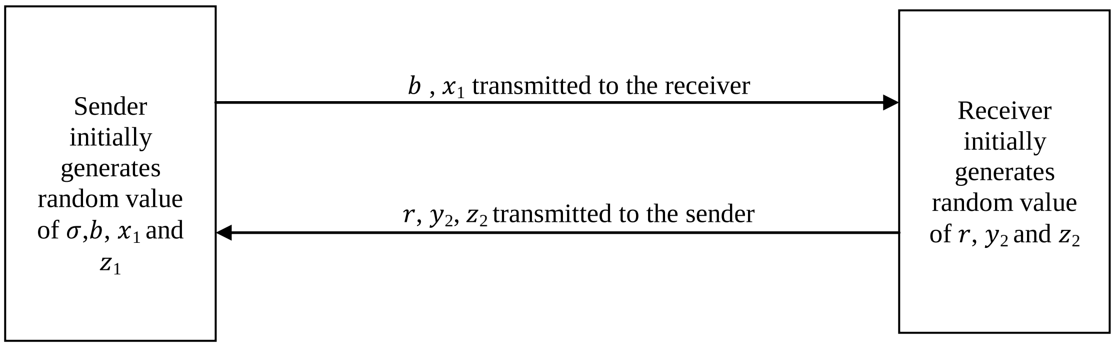

Sender initially generates random value of σ,b, x_1 and z_1
b , x_1 transmitted to the receiver
r, y_2, z_2 transmitted to the sender
Receiver initially generates random value of r, y_2 and z_2

Figure 4.1: Exchange of values between sender and receiver at the initial state

Receiver calculates the new value of $\dot{y_2}$and $\dot{z_2}$with the help of r and b and x_1 (received from sender) using the equation 4.11 and 4.12 and returns the value of $\dot{y_2}$and $\dot{z_2}$to the sender. In the equation 4.11the value of $x = x_1$ (current value received form sender).

$$\dot{y}_2 = rx - y_2 - xz_2 \tag{4.11}$$

$$\dot{z}_2 = xy_2 - bz_2 \tag{4.12}$$

Sender calculates the new value of $\dot{x_1}$and $\dot{z_1}$with the help of receives value y_2 from the receiver and own generated values σ and b using the equation 4.13 and 4.14 and sends the value of new $\dot{x_1}$ to the receiver and so on. In equation 4.13 and 4.14 $y = y_2$(current value received from receiver)

$$\dot{x}_1 = \sigma(x_1 - y) \tag{4.13}$$

$$\dot{z}_1 = x_1y - bz_1 \tag{4.14}$$

The figure 4.2 shows the exchange of updated parameters.

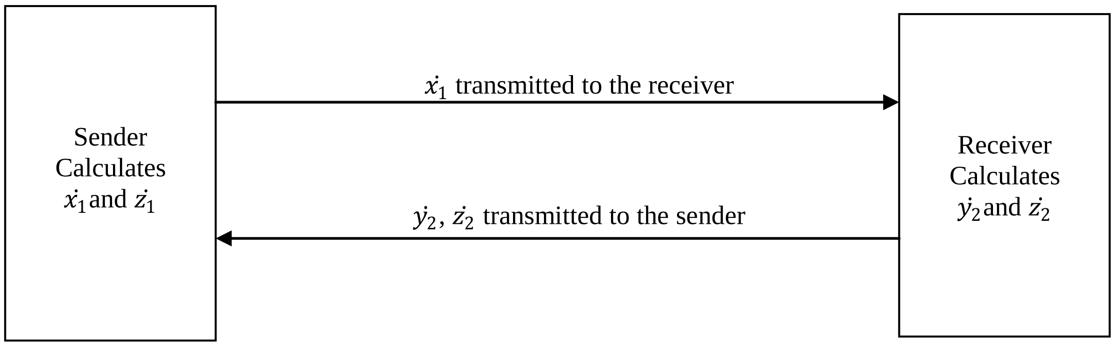

Sender Calculates $\dot{x_1}$and $\dot{z_1}$
$\dot{x_1}$ transmitted to the receiver
$\dot{y_2}$, $\dot{z_2}$ transmitted to the sender
Receiver Calculates $\dot{y_2}$and $\dot{z_2}$

Figure 4.2: Exchange of updated values of the parameters $\dot{x_1}$, $\dot{y_2}$and $\dot{z_2}$

After predefined amount of exchange of parameters sender generates a nonce which is a random number. This nonce gets encrypted using a symmetric cipher with z_1 as the key and sends the results of the encryption using equation 4.15.

$$En\_Nonce = Encrypt_{z_1}(Nonce) \tag{4.15}$$

The receiver receives En_Nonce from sender. Then decrypts En_Nonce using z_2 as the key and performs a defined function $f(\,)$ on it using equations 4.16and 4.17 respectively.

$$De\_Nonce = Decrypt_{z_2}(En\_Nonce) \tag{4.16}$$

$$Fn\_Nonce = f(De\_Nonce) \tag{4.17}$$

The receiver encrypts the result of the previous step using z_2 as the key and sends the result to the sender using equation 4.18.

$$En\_Fn\_Nonce = Encrypt_{z_2}(Fn\_Nonce) \tag{4.18}$$

Figure 4.3 shows the exchange of En_Nonce and EN_Fn_Nonce.

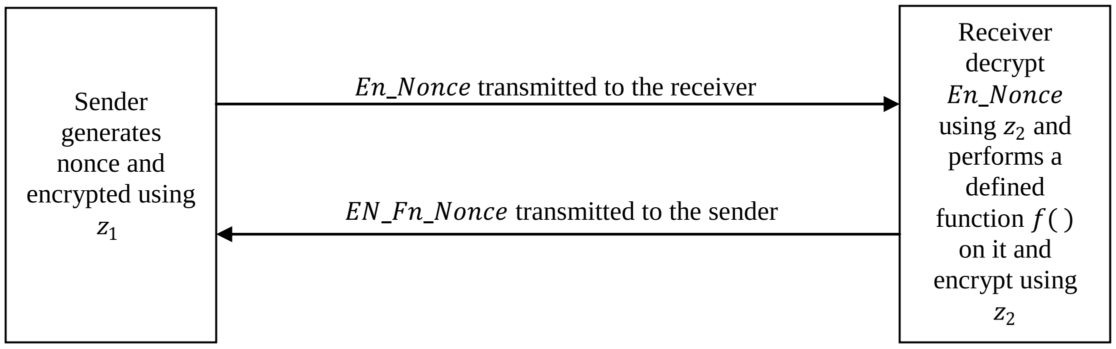

Sender generates nonce and encrypted using z_1
En_Nonce transmitted to the receiver
EN_Fn_Nonce transmitted to the sender
Receiver decrypt En_Nonce using z_2 and performs a defined function $f(\,)$ on it and encrypt using z_2

Figure 4.3: Exchange of En_Nonce and EN_Fn_Nonce

Sender receives the message En_Fn_Nonce from receiver and tries to decrypts this message using z_1 as the key and also performs the inverse of the pre-defined function $f(\,)$ and checks if the original nonce is obtained or not using equation 4.19.

$$Nonce = f^{-1}\left(Decrypt_{z_1}(En\_Fn\_Nonce)\right) \tag{4.19}$$

If the original 'Nonce' is generated it can be concluded that both chaotic system has the same value of z i.e. $z_1 = z_2$ on which they get synchronized. Then z_1 is used as a secret seed for generation of identical input vector for sender and receiver. Otherwise if original Nonce not get obtained then again some predefined amount of message get exchanged between sender and receiver for chaos synchronization.

Two CDHLPs start with chaos synchronized secret seed value generated identical input vector and anonymous random weight vector. CDHLPs compute their final output based on input and weight vector, and communicate to each other. If both are in agreement, their weights are updated using appropriate learning rules. Within finite number of steps both CDHLPs is synchronized and as a results weight vector of both CDHLPs become identical. This indistinguishable weight vector forms the session key for a particular session.

However, only the process of generating keys does not guarantee the information security. Therefore, any attacker can also synchronize with an authorized device, because the protocol is a public knowledge. Thus, to ensure that only entities authorized have access to information is necessary authentication service. The function of the authentication service is to ensure the recipient that the message is from the source that it claims. There are several authentication methods, differentiated mainly by the use of secret-keys or public-keys. Unlike encryption algorithms, in public-key authentication the user A send message encrypted with A's private-key. The recipient of the message uses the public-key to verify the message, thus ensuring that only the owner of the private-key could have encrypted the message. On secret keys authentication both entities must have a common secret code. In this proposed approach two secret codes are used, called RSC ($Receiver\ Secret\ Code$) and SSC ($Sender\ Secret\ Code$), as shown in the figure 4.4.

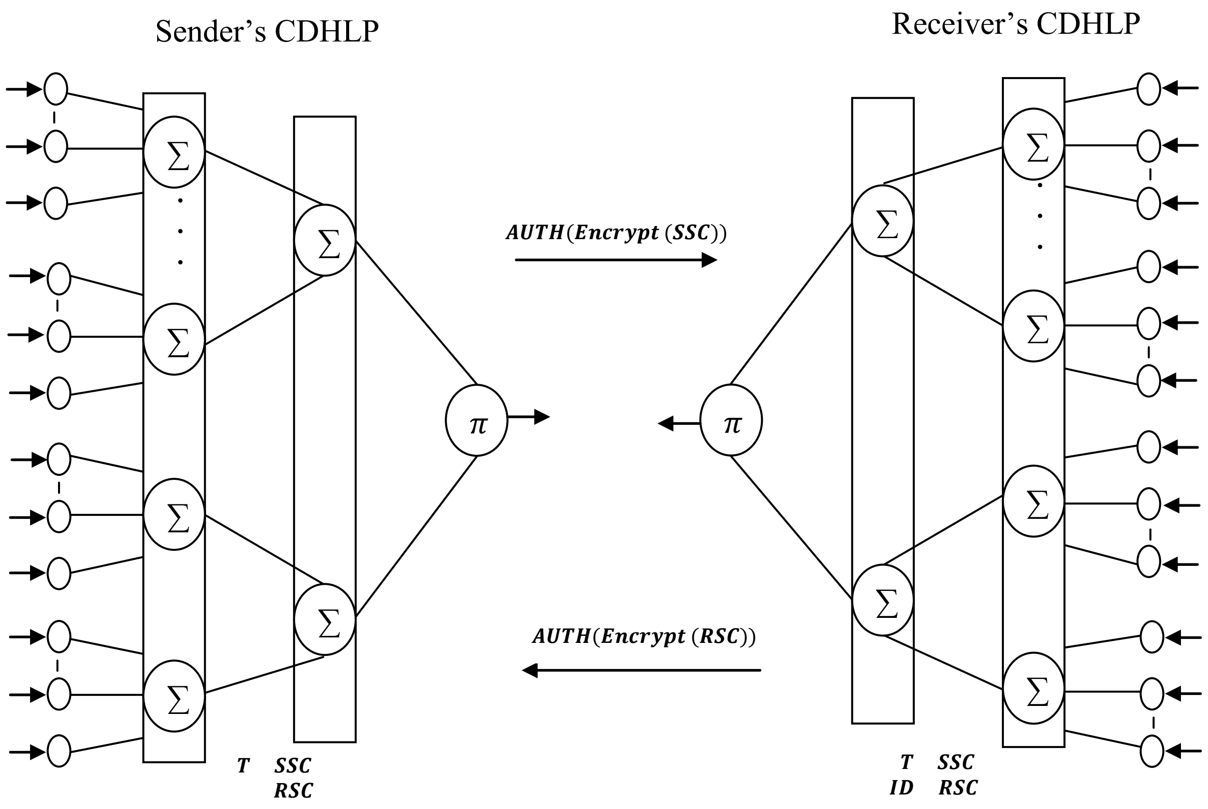


Figure 4.4: Exchange of authentication frame during session key certification phase

In the proposed technique CDHLPs start synchronization by exchanging some control frames. The process involves message integrity and synchronization test. Proposed CDHLP synchronization uses transmission of control frames at the time of three way handshaking based TCP connection establishment phase, as given in table 4.1.

Table 4.1
Control frames of CDHLP synchronization

Frame	**Description**
SYN	SYN frame transmitted to the receiver for synchronization in connection establishment phase
ACK_SYN	ACK_SYN frame transmitted to the sender for positive acknowledgement respect to SYN frame
NAK_SYN	NAK_SYN frame transmitted to the sender for negative acknowledgement respect to SYN frame
FIN_SYN	FIN_SYN frame transmitted by either party for closing the connection

The SYN frame is used to establish the connection to the other side. It carries index information of different initial parameters. On transmitting the SYN frame, the sender starts a timer and waits for a reply from the receiver. If the receiver does not take any action until a certain time limit and number of attempts exceeded a certain value, the sender restarts the synchronization procedure. When the receiver receives the SYN frame, it carry out the integrity test. If the messages are received as sent (with no replication, incorporation, alteration, reordering, or replay) the receiver will execute the synchronization check. The sender and receiver have an identical T variable formally store in their respective memory. The sender sends the encrypted T to the receiver. Here the receiver utilizes its $128/192/256$ bits weights to decrypt of the encrypted T. If the result is identical to T formerly stored in receiver memory i.e. $(Decrypt_{Receiver\_weig\,ht}(Encrypt_{Sender\_weig\,ht}(T)) = T$ then the networks are synchronized. This is the best case solution where sender and receiver arbitrarily choose weight vector which are identical. So, networks are synchronized at initial stage. The receiver should send the frame FIN_SYN to alert the sender. But most of the time this best case is may not achievable. If decryption algorithm does not produce predictable result, the receiver should use the secret seed of senders to produce the input vector (X) which is identical to sender. With this input vector the receiver will work out its $output$ $(\tau^{Receiver})$. If the receiver's and sender's outputs are different, the receiver should not fine-tune its weights and inform the sender its output. The receiver sends the $NACK\_SYN$ frame to notify the sender, with the same ID value. The proposed NAK_SYN frame in this technique is used for providing the negative acknowledgement in respect to the SYN frame. If receiver's output is equal to sender's output i.e. $(\tau^{Receiver} = \tau^{Sender})$ then receiver update it weights. At the end of weights update, the receiver should report the sender that outputs are equal. The receiver uses the ACK_SYN frame to notify the sender, with the same ID value received from sender. The proposed ACK_SYN frame in this technique is used for providing the positive acknowledgement in respect to the SYN frame. On receipt of ACK_SYN, the sender also updates its weight. If sender receives ACK_SYN it should update its weights. The sender will create new synchronization frame until receive the FIN_ACK frame from receiver. When the sender receives the FIN_ACK frame, it stops the further synchronization. The proposed FIN_SYN frame in this technique is used for closing the connection. At end of synchronization, both networks provide the identical weight vector which acts as a session key identical to both ends. The figure 4.5 shows the exchange of frames during CDHLP synchronization process.

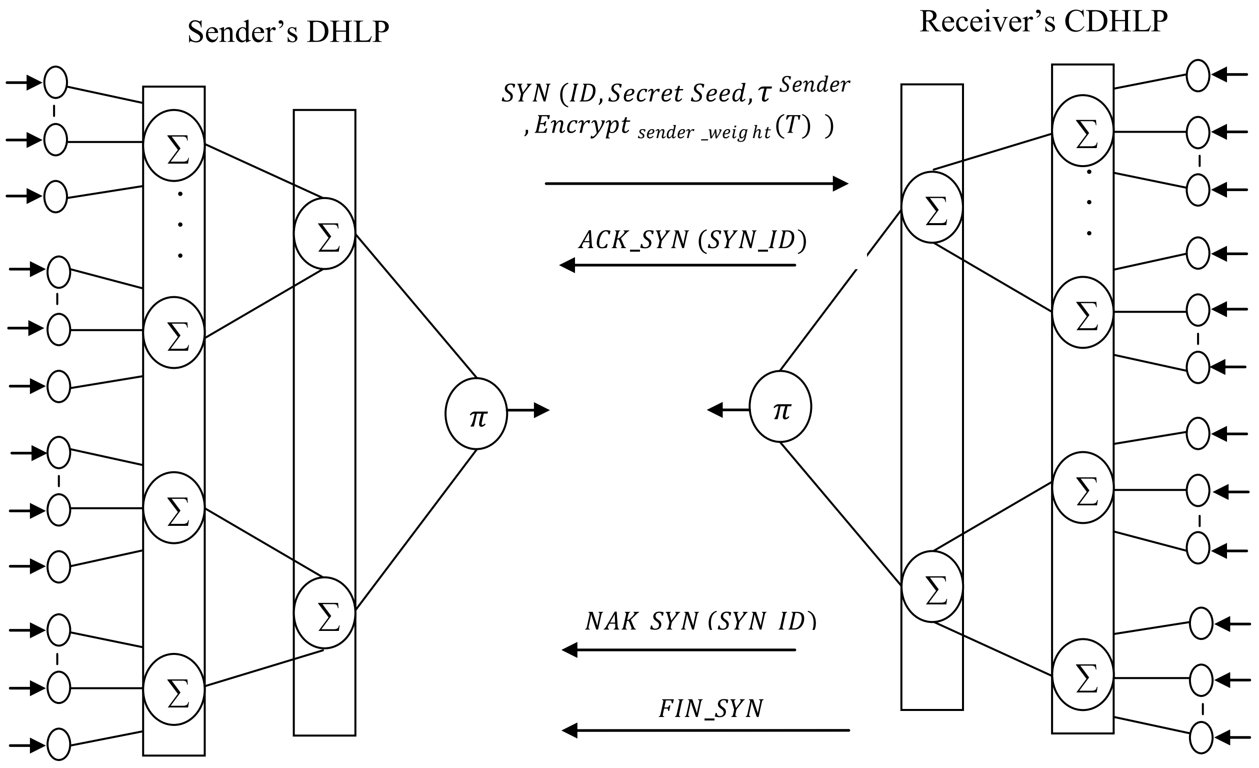


Figure 4.5: Exchange of control frames between sender and receiver during CDHLP Synchronization

Table 4.2 shows the different frames and their corresponding Command Codes

Table 4.2
CDHLP control frames and their command codes

Frame	Command
SYN	0000
FIN_SYN	0001
ACK_SYN	0010
NAK_SYN	0011
$AUTH$	0100
$Reserved$	0101-1111

The identifier (ID) is the function of informing the sender and receiver where the message is a recent message. The variable ID starts with zero and is incremented every time that the sender sends a synchronization frame. The detailed frame format of SYN frame is discussed in section 4.2.1.1.1. The detailed frame format of ACK_SYN frame is given in section 4.2.1.1.2. The frame format of NAK_SYN frame has been discussed in section 4.2.1.1.3. The frame format of FIN_SYN frame is discussed in section 4.2.1.1.4.

4.2.1.1.1 Synchronization (SYN) Frame

During synchronization process sender constructs a SYN frame and transmit to the receiver for handshaking in connection establishment phase. Sender utilizes its initial 128 weights as key for encryption of T variable (formerly stored in its memory) $Encrypt_{Sender\_weight}(T)$. Sender constructs a SYN frame and transmitted to the receiver for handshaking purpose in connection establishment phase. SYN usually comprises of $Command\ Code,\ SYN\ ID,$ $Sender\ output(\tau^{Sender}),\ Encrypt_{Sender\_weight}(T)$ and $CRC.SYN$ frame has the fixed $Command\ Code$ i.e. 0000. So, five different fields like $Command\ code$ needs four bits. $SYN\ ID,\ Sender\ output(\tau^{Sender}),\ Encrypt_{Sender\_weight}(T)$ and CRC needs eight bits, one bits, 128 bits, sixteen bits respectively. When the receiver receive SYN frame, the receiver should carry out integrity test. Receiver performs Integrity test on receiving the SYN frame. If the messages are received as sent (with no replication, incorporation, alteration, reordering, or replay) the receiver will execute the synchronization test. In synchronization test receiver utilize its 128 first weights as key for decryption of $Encrypt_{Sender\_weight}(T)$ that is received from the sender. This received value is decrypted in the receiver end. After decryption operation if $(Decrypt_{Receiver\_weight}(Encrypt_{Sender\_weight}(T)) = T$ then networks are synchronized. Figure 4.6 shows the complete frame format of SYN frame.

$Command\ Code$ 0000	$SYN\ ID$	τ^{Sender}	$Encrypt_{Sender\_weight}(T)$	CRC $(Cyclic\ Redundancy\ Checker)$
4	8	1	128	$16\ (bits)$

Figure 4.6: Synchronization (SYN) frame

4.2.1.1.2 Synchronization (ACK_SYN) Acknowledgement Frame

ACK_SYN frame send by the receiver to the sender in respect of ACK frame for positive acknowledgement of the parameters value. This proposed frame comprises of $Command\ code, SYN\ ID, CRC$. $Command\ Code$ needs four bits. The ACK_SYN frame has the fixed $Command\ code$ i.e. 0010. Eight bits is used for representing the $SYN\ ID$ and CRC needs sixteen bits for error checking purpose. Now check the condition i.e. If $(Decrypt_{Receiver\_weight}(Encrypt_{Sender\_weight}(T)) \neq T$ then receiver use the $Secret\ Seed$ received from sender to produce the receiver input vector (X) identical to sender input vector(X) and calculates the output $\tau^{Receiver}$. If $(\tau^{Receiver} = \tau^{Sender})$ then receiver should update their weights where $\sigma_k^{Sender/Receiver} = \tau^{Sender/Receiver}$ using learning rule. At end of weight updation of the receiver, then it sends ACK_SYN with the same ID to instruct the sender for updating the weights. If sender receives ACK_SYN it should update its weights. Figure 4.7 shows the complete frame format of ACK_SYN frame.

$Command\ Code$ 0010	$SYN\ ID$	CRC $(Cyclic\ Redundancy\ Checker)$
4	8	16 $(bits)$

Figure 4.7: Acknowledgement of Synchronization (ACK_SYN) frame

4.2.1.1.3 Negative Acknowledgement (NAK_SYN) Frame of Synchronization

NAK_SYN frame send by the receiver to the sender in respect of ACK frame for negative acknowledgement of the parameters value. This proposed frame comprises of $Command\ Code, SYN\ ID, CRC$. $Command\ Code$ needs four bits. The ACK_SYN frame has the fixed $Command\ Code$ i.e. 0011. Eight bits are used for representing the $SYN\ ID$ and CRC needs sixteen bits for error checking purpose. If $(\tau^{Receiver} \neq \tau^{Sender})$ then the receiver sends the message NAK_SYN to notify the sender. If the receiver and sender outputs are different, the receiver should not fine-tune its weights and inform the sender. The receiver sends the message NAK_SYN to notify the sender, with the same ID value. Figure 4.8 shows the complete frame format of NAK_SYN frame.

$Command\ Code$ 0011	SYN_ID	CRC $(Cyclic\ Redundancy\ \ Checker)$
4	8	16 $(bits)$

Figure 4.8: Negative Acknowledgement of Synchronization (NAK_SYN) frame

4.2.1.1.4 Finish Synchronization (FIN_SYN) Frame

FIN_SYN frame send by the either party for finish the synchronization process. This proposed frame comprises of $Command\ Code, SYN\ ID, CRC$. $Command\ Code$ needs four bits. The FIN_SYN frame has the fixed $Command\ Code$ i.e. 0001. Eight bits is used for representing the $SYN\ ID$ and CRC needs sixteen bits for error checking purpose. If $(Decrypt_{Receiver\ \_weig\ ht}(Encrypt_{Sender\ \_weig\ ht}(T)) = T$ then networks are synchronized. Receiver sends the FIN_SYN frame to the sender. Figure 4.9 shows the complete frame format of FIN_SYN frame.

$Command\ Code$ 0001	$SYN\ ID$	CRC $(Cyclic\ Redundancy\ \ Checker)$
4	8	16 $(bits)$

Figure 4.9: Finish Synchronization (FIN_SYN) frame

The CDHLP synchronization algorithm for generating synchronized session key is discussed in section 4.2.1.1.5. Section 4.2.1.1.6 presents the computational complexity of the CDHLP synchronization algorithm and CDHLP learning is discussed in section 4.2.1.1.7.

4.2.1.1 CDHLP Synchronization

Sender and receiver initially start Chaos synchronization between them to construct a common seed value at both ends. This Chaos synchronized identical seed value is used to generate the identical input vector at sender and receiver. Two CDHLPs start with Chaos synchronized common seed value generated identical input vector and anonymous random weight vector. In each time both CDHLPs compute their final output based on input and weight vector, and communicate to each other. If both are be in agreement on the mapping between the present input and the output, their weights are updated according to an appropriate learning rule. In the case of discrete weight values this process leads to full synchronization in a finite number of steps. After synchronization procedure weight vector of both CDHLPs become identical. These indistinguishable weight vector forms the session key for a particular session.

Input : *Tuning parameters* (σ, b, r, x_1, y_2 *and* z_2), *random weights*

Output : *Sender's and receiver's synchronized CDHLP along with synchronized session key*

Method : *Sender's and receiver's CDHLPs both are be in agreement on the mapping between the present input and the output, their weights are updated according to an appropriate learning rule. After synchronization procedure weight vector of both CDHLPs become identical. These indistinguishable weight vector forms the session key for a particular session.*

Step 1. *Sender initializes the value of* σ *and* b, *after that value of* b *is send to the receiver.*

Step 2. *Receiver initializes the value of* r.

Step 3. *Sender generates the point* x_1 *and* z_1.

Step 4. *Receiver generates the point* y_2*and* z_2.

Step 5. *Sender sends* x_1 *to receiver and receiver sends* y_2 *and* z_2 *to sender.*

Step 6. *Receiver calculates the new value of* $\dot{y}_2$ *and* $\dot{z}_2$*with the help of* r *and* b *using the equations 4.20 and 4.21 then returns the value of* $\dot{y}_2$ *and* $\dot{z}_2$ *to the sender.*

$$\dot{y}_2 = rx - y_2 - xz_2 \tag{4.20}$$

$$\dot{z}_2 = xy_2 - bz_2 \tag{4.21}$$

Step 7. *Sender calculates the value of* $\dot{x}_1$ *and* $\dot{z}_1$ *with the help of* y_2, σ *and* b *using equations 4.22 and 4.23 then sends the value of* $\dot{x}_1$ *to the receiver and so on.*

$$\dot{x}_1 = \sigma(x_1 - y_2) \tag{4.22}$$

$$\dot{z}_1 = x_1y_2 - bz_1 \tag{4.23}$$

Step 8. *Sender generates a nonce. This nonce gets encrypted using a symmetric cipher with* z_1 *as the key and sends the results of the encryption using equation 4.24.*

$$En\_Nonce = Encrypt_{z_1}(Nonce) \tag{4.24}$$

Step 9. *The receiver decrypts* En_Nonce *using* z_2 *as the key, performs a defined function on it using equation 4.25 and 4.26.*

$$De\_Nonce = Decrypt_{z_2}(En\_Nonce) \tag{4.25}$$

$$Fn\_Nonce = f(De\_Nonce) \tag{4.26}$$

Step 10. *The receiver encrypts the result of the previous step using* z_2 *as the key and sends the result to the sender illustrated in equation 4.27.*

$$En\_Fn\_Nonce = Encrypt_{z_2}(Fn\_Nonce) \tag{4.27}$$

Step 11. *The sender decrypts this message using* z_1 *as the key, performs the inverse of the pre-defined function and checks if the original nonce is obtained as shown in equation 4.28.*

$$Nonce = f^{-1}\left(Decrypt_{z_1}(En\_Fn\_Nonce)\right) \tag{4.28}$$

Step 12. *If synchronization is not achieved, the process is repeated from step 5.*

Step 13. *If synchronization is achieved i.e.* $z_1 = z_2$ *then* z_1 *is used as a seed for a pseudo random number generator to generate identical input vector*(X) *at both end.*

Step 14. *Initialization of synaptic links between input layer and first hidden layer and between first hidden layer and second hidden layer using random weights values. Where,* $W_{ij}\,\epsilon\{-L, -L+1, \dots, +L\}$.

Repeat step 15 to step 24 until the full synchronization is achieved,

Step 15. *The input vector*(X) *is generated both end using Chaos synchronized seed value.*

Step 16. *Computes the values of hidden neurons by the weighted sum over the current input values. Each hidden neurons in first hidden layer produces* $\sigma^1{}_i$ *values and each hidden neuron in second hidden layer produces* $\sigma^2{}_p$ *values.*

These can be represented using equation 4.29 and 4.30.

$$\sigma^1{}_i = sgn\left(\sum_{i=1}^{K1} \sum_{j=1}^{N} W_{i,j} X_{i,j}\right) \tag{4.29}$$

$$\sigma^2{}_p = sgn\left(\sum_{p=1}^{K2} \sum_{i=1}^{K1} W_{p,i} \sigma_i^1\right) \tag{4.30}$$

$sgn(x)$ *is a function represents in equation 4.31, which returns the value* $-1, 0$ *or* 1:

$$sgn(x) = \begin{cases} -1 & if\ x < 0 \\ 0 & if\ x = 0 \\ 1 & if\ x > 0 \end{cases} \tag{4.31}$$

If the weighted sum over its inputs is negative then set $\sigma_i = -1$. *Hence, set* $\sigma_i = +1$, *if the weighted sum over its inputs is positive, or else if weighted sum is zero then it is set to,* $\sigma_i = 0$.

Step 17. *Compute the value of the final output neuron by computing multiplication of all values produced by* $K2$ *no. hidden neurons using the equation 4.32.*

$$\tau = \prod_{p=1}^{K2} \sigma_p^2 \tag{4.32}$$

Step 18. *Sender utilizes its* 128 *first weights as key for encryption of* T *variable (formerly stored in its memory)* $Encrypt_{Sender\_weight}(T)$.

Step 19. *Sender constructs a* SYN *frame and transmitted to the receiver for handshaking purpose in connection establishment phase.* SYN *usually comprises of several fields* $Command\ code, ID, Secret\ Seed$, *Sender output*$(\tau^{Sender})$,$Encrypt_{Sender\_weight}(T)$ *and* CRC *(Cyclic Redundancy Checker).*

Step 20. *Receiver performs Integrity test after receiving the SYN frame and then Receiver utilize its* 128 *first weights as key for decryption of* $Encrypt_{Sender\_weight}(T)$ *that was received from the sender.*

Step 21. *If* $(Decrypt_{Receiver\_weight}(Encrypt_{Sender\_weight}(T)) = T$ *then networks are synchronized. Go to step 25.*

Step 22. *If* $(Decrypt_{Receiver\_weight}(Encrypt_{Sender\_weight}(T)) \neq T$ *then receiver use the chaos synchronized secret seed to produce the receiver input vector*(X) *identical to sender input vector*(X) *and calculates the output* $\tau^{Receiver}$ *using step 16 and step 17.*

Step 23. *If* $(\tau^{Receiver} = \tau^{Sender})$ *then performs the following steps*

Step 23.1 *Receiver update their weights where* $\sigma_k^{Sender/Receiver} = \tau^{Sender/Receiver}$ *using any of the learning rules discussed in chapter 1 section 1.8.*

Step 23.2 *At end of receiver's weights updation, the receiver sends* ACK_SYN *to instruct the sender for updating the weights using step 23.1.*

Step 23.3 *Sender transmits* $Encrypt_{Sender\_updated\_weight}(T)$ *to receiver.*

Step 23.4 *Receivers checks* $if\,(Decrypt_{Receiver\_updated\_weight}(Encrypt_{Sender\_updated\_weight}(T)) = T$ *then networks are synchronized. Go to step 25.*

Step 23.5 *Perform the following checking* $if\,(Decrypt_{Receiver\_updated\_weight}(Encrypt_{Sender\_updated\_weight}(T)) \neq T$ *then networks are still not synchronized. Go to step 23.1.*

Step 24. *If* $(\tau^{Sender} \neq \tau^{Receiver})$ *then the receiver sends the message* NAK_SYN *to notify the sender. Go to step 15.*

Step 25. *Finally, the receiver sends the frame* FIN_SYN *to inform the sender to finish the synchronization phase.*

4.2.1.2 Complexity Analysis

In CDHLP synchronization algorithm initialization of the value of σ and b takes unit amount of computation at sender. Receiver initialization of the value of r also takes unit amount of computation. Generation of the point x_1 and z_1 needs unit amount of computation. Generation of the point y_2 and z_2 requires unit amount of computation. Receiver calculates the new value of y_2 and z_2 with the help of r and b. This step also takes unit amount of computation. Sender calculates the value of x_1 and z_1 with the help of y_2, σ and b. This step also takes unit amount of computation. Sender generates a nonce having a random value. This nonce is encrypted using a symmetric cipher with z_1 as the key and sends the results of the encryption. This step needs $(nonce\ length)$ amount of computation. The receiver decrypts En_Nonce using z_2 as the key. It also takes $(nonce\ length)$ amount of computation. The receiver encrypts the result of the previous step using z_2 as the key. It takes $(message\ length)$ amount of computation. The sender decrypts this message using z_1 as the key, performs the inverse of the pre-defined function and checks if the original nonce is or not. It takes $(encrypted\ message\ length)$ amount of computation. Initialization of weight vector takes $(N \times K1 + K1 \times K2)$ amount of computations. For example, if $N = 2, K1 = 4, K2 = 2$ then total numbers of synaptic links (weights) are $(2 \times 4 + 4 \times 2) = 16$. So, it takes 16 amount of computations. Generation of N number of input vector for each $K1$ number of hidden neurons takes $(N \times K1)$ amount of computations. Computation of the hidden neuron outputs takes $(K1 + K2)$ amount of computations. Where $K1$ and $K2$ are the number of hidden units in first and second layer respectively. Computation of final output value takes unit amount of computation because it needs only a single operation to compute the value. Encryption of T using *Exclusive-OR* operation also takes unit amount of computation. Decryption of T using *Exclusive-OR* operation also takes unit amount of computation. Checking $if\ \ (Decrypt_{Receiver\ \_weig\ ht}(Encrypt_{Sender\ \_weig\ ht}(T)) = T$ or not takes unit amount of computation. Weight updating procedure takes place where $\sigma_k^{Sender\ /Receiver} = \tau^{Sender\ /Receiver}$ using any of the learning rules which takes $O\left(no.\,of\ \sigma_k^{Sender\ /Receiver} = \tau^{Sender\ /Receiver}\right)$ amount of computations.

In best case of DHLP synchronization algorithm, sender's and receiver's arbitrarily chosen weight vectors are identical. So, networks are synchronized at initial stage do not needs to update the weight using learning rule. Here, $\big(nonce\ length + message\ length + encrypted\ message\ length + (N \times K1) + (N \times K1 + K1 \times K2) + (K1 + K2)\big)$ amount of computation is needed in best case which is in form of $\mathrm{O(Generation\ of\ common\ seed\ value + initialization\ of\ input\ vector + initialization\ of\ weight\ vector + Computation\ of\ the\ hidden\ neuron\ outputs)}$.

If the sender's and receiver's arbitrarily chosen weight vector are not identical then in each iteration the weight vectors of the hidden unit which has a value equivalent to the pereceptron output are updated according to the learning rule. This scenario leads to average and worst case situation where I number of iteration to be performed to generate the identical weight vectors at both ends. So, the total computation for the average and worst case is $\big(nonce\ length + message\ length + encrypted\ message\ length + (N \times K1) + (N \times K1 + K1 \times K2) + (K1 + K2)\big) + \big(I \times (no.of\ \sigma_k^{Sender\ /Receiver} = \tau^{Sender\ /Receiver}\ )\big)$ This is can be expressed as $\mathrm{O}(\text{Time complexity in first iteration} + (\text{No. of iteration} \times \text{No. of weight updation}))$.

4.2.1.3 CDHLP Learning Mechanism

If the output bits are different for sender (A) and receiver (B) i.e. $\tau^A \neq \tau^B$, nothing get changed. If $\tau^A = \tau^B = \tau$, only the weights of the hidden units with $\sigma_k^{A/B} = \tau^{A/B}$ will be updated. The weight vector of this hidden unit is adjusted using any of the learning rules discussed in chapter 1 section 1.8. For small L values Hebbian takes less synchronization steps than other two learning rules in the range of $2-4-2-5$ $(N-K1-K2-L)$ to $2-4-2-15$ but as the L value increases Hebbian rule takes more steps to synchronize than other two learning rules. Here, Anti-Hebbian rules takes less time than the other two learning rules in the range of $2-4-2-20$ to $2-4-2-30$. Random Walk outperform from $2-4-2-35$ and beyond that. The most vital findings is that if the synaptic depth i.e. weight range (L) is increased, the complexity of a successful attack grows exponentially, but there is only a polynomial increase of the effort needed to generate a key. So, increasing the L value security of the system can be increased.

4.2.1.2 Genetic Algorithm (GA) guided Fittest Keystream Generation

A Genetic Algorithm (GA) guided approach is used to construct the keystream for encryption/decryption. Instead of this technique any other light weight encryption/decryption technique also may use for exchanging message between sender and receiver.

Ordinary version of GA suffers from many troubles such as getting stuck in a local minimum and parameters dependence. In the proposed encryption/decryption keystream generation self acclimatize GA approach some useful improvements have been proposed to enhance the performance of the simple GA, by dynamically adjusts selected control parameters, such as population size and genetic operation rates, during the course of evolving a problem solution. That is because, one of the main problems related to GA is to find the optimal control parameter values that it uses, when a poor parameter setting is made for an evolutionary computation algorithm, the performance of the algorithm will be seriously degraded. Thus, different values may be necessary during the course of a run. A widely practiced approach to identify a good set of parameters for a problem is through experimentation. For these reasons, proposed technique offers the most appropriate exploration and exploitation behavior. Following sub sections discussed about methodology used in self acclimatize GA based encryption/decryption keystream generation.

The LFSR (Linear Feedback Shift Register) based generator is used to generate the chromosomes (solution) in self acclimatize GA. The operator used in this work is presented in table 4.3.

Table 4.3
Operator's format and their meaning

Operator	**Format**	**Meaning**
|	$\|ab$	$Bitwise\ OR$
&	$\&ab$	$Bitwise\ AND$
^	$\hat{}ab$	$Bitwise\ Exclusive - OR$
X		$Character\ sequence\ from\ `a' \ldots\ p'\ represents\ the\ number$ 0..15
SR	SRx	$Shift\ Register\ is\ represents\ as\ SR\ and\ x\ denotes\ the\ feedback\ polynomial$

Each chromosome that represents candidate keystream generators is strings of characters which are represented using prefix notation. These syntactic rules should be preserved during the generation of the initial population. The initial states and feedback functions of the shift registers are represented as strings of the letters $`a' \ldots 'p'$. These letters represent the numbers $0 \ldots 15$. Thus, each letter is a sequence of four bits. So, from this (16×8)=128 bit keystream get generated. The length of a LFSR is determined by the number of letters which are initially generated randomly. The number of these letters must be even, because half of them for the initial state, and the second half for the feedback function. For example, if the number of these letters is eight, then four letters are used for the feedback function, thus, the length of LFSR is 16 bits (4×4). Furthermore, the first zeros of the feedback function are ignored.

For example, consider the LFSR "$SR\,meip$" So, binary representation of the LFSR "SR meip" will be 1100 0100 1000 1111. Here 1100 0100 (me) is used for the initial state and rest half i.e. 1000 1111 (ip) is used for the feedback function. Representation of 'i' is number $(8)_{10} = (1000)_2$ then the first three zeros are ignored. Now, here length of 1111 (p) is four bits and length of 1000 (i) is 1bit after ignoring first three zeros from LSB. So, the length of this LFSR will be five bits $(4 + 1) = 5$. The following are examples of the chromosomes:

Chromosome 1: $SRahij$

Chromosome 2: $\wedge SRmchpSRncoe$

Chromosome 3: $SRahjflpdmobenka$

Chromosome 4: $|\&SRaj \wedge SRfh\&|SRglnc \wedge SRbacfSRpoSRln$

The fitness value is a measurement of the goodness of the keystream (individual), and it is used to control the application of the operations that modify a population. There are a number of metrics used to analyze keystream, which are keystream randomness, linear complexity and correlation immunity. Therefore, these metrics should be taken in account in designing keystream (individual), and they are in general hard to be achieved. Three factors are considered in the fitness evaluation of the keystream (individual) which are:

a. *Randomness of the generated keystream (individual)*
b. *Keystream (individual) period length*
c. *Keystream (individual) length*

a. *Randomness of the generated keystream (individual)* - The purpose of evaluation of randomness is to determine whether that number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence. The test assesses the closeness of the fraction of ones to ½, that is, the number of ones and zeroes in a sequence should be about the same. The equation 4.33 is used for the evaluation of keystream randomness using the frequency and serial tests, in which, n_w is the frequency of w in the generated binary sequence.

$$f_1 = |n_0 - n_1| + \left|n_{00} - \frac{SZ}{4}\right| + \left|n_{01} - \frac{SZ}{4}\right| + \left|n_{10} - \frac{SZ}{4}\right| + \left|n_{11} - \frac{SZ}{4}\right| \qquad (4.33)$$

Fitness f_1 calculates the frequency of the bits. This function is derived from the fact that in the random sequence, *Probability (n_o) = Probability (n_1)* which checks frequency of 0 and 1 in a binary string and *Probability (n_{01}) = Probability (n_{11}) = Probability (n_{10}) = Probability (n_{00})* which checks the probability of occurrence of the pattern $00, 01, 10$ and 11 in a binary string.

b. *Keystream (individual) period length* - The focus of keystream (individual) period length evaluation is to determine the total number of zero and one runs in the entire sequence, where a run is an uninterrupted sequence of identical bits. A run of length k means that a run consists of exactly k identical bits and is bounded before and after with a bit of the opposite value. The purpose of this evaluation is to determine whether the number of runs of ones and zeros of various lengths is as expected for a random sequence. In particular, this test determines whether the oscillation between such substrings is too fast or too slow. $\frac{1}{2^i} \times n_r$ of the runs in the sequence are of length i, where n_r is the number of runs in the sequence. Thus, the following equation 4.34 represents the period length.

$$f_2 = \sum_{i=1}^{M} \left|\left(\frac{1}{2^i} \times n_r\right) - n_i\right| \qquad (4.34)$$

Where M is maximum run length, and n_i is the desired number of runs of length i.

c. *Keystream (individual) length* - Another factor is considered in the evaluation of the fitness value which is the size of the candidate keystream (length of the individual).

Thus, the fitness function used to evaluate the chromosome x is given in equation 4.35, where $weight$ is a constant and sz *is* the key stream period length:

$$fitness(x) = \frac{SZ}{1+f_1+f_2} + \frac{weight}{length(x)} \qquad (4.35)$$

More copies to good strings and fewer copies to bad string get selected using Roulette wheel selection. In this proportional selection scheme number of copies taken to be directly proportional to its fitness. It mimics the natural selection procedure to some extent. The selection strategy, used to select chromosomes for the genetic operations, is the roulette selection. The old population is completely replaced by the new population which is generated from the old population by applying the genetic operations.

Crossover operation performs exchange of genetic information. It takes place between randomly selected parent chromosomes. In this scheme uniform crossover is performed with probability 0.6 to 0.9. Before applying the crossover operation chromosomes are converterd into binary representation. Figure 4.10 shows the uniform crossover operation having binary chromosome length of eight.

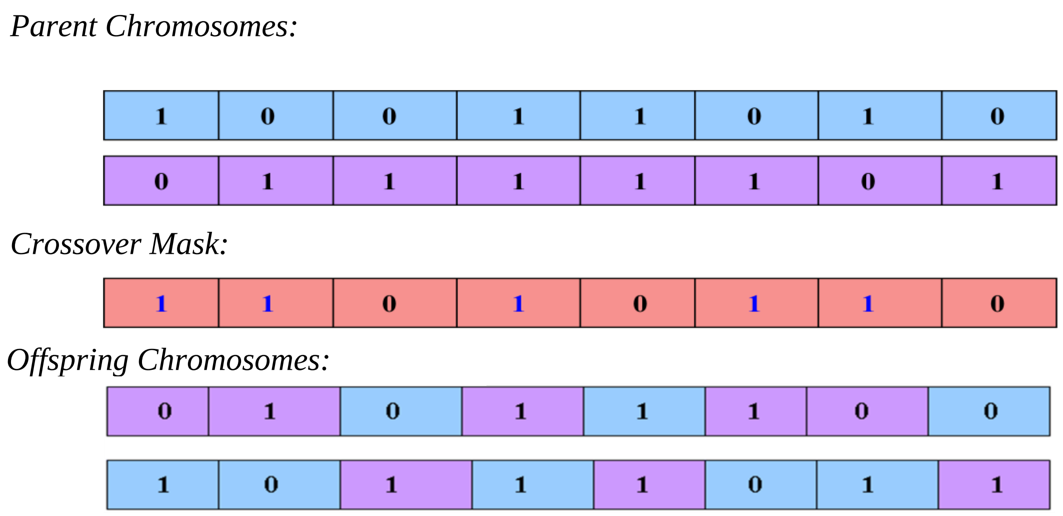
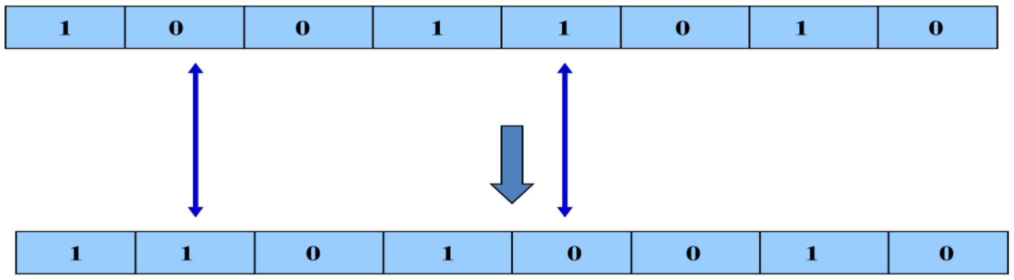


Figure 4.10: Uniform Crossover operation

Mutation operation is a random alternation in the genetic structure. It introduces genetic diversity into the population. performs exchange of genetic information. It takes place between randomly selected parent chromosomes. In this scheme mutation is performed with probability 0.001 to 0.01. Figure 4.11 shows the mutation operation having chromosome length eight.

Parent Chromosome:



Mutated Chromosome:

Figure 4.11: Mutation operation

The goals of GA with adaptive probabilities of crossover and mutation are to maintain the genetic diversity in the population and prevent the GAs to converge prematurely to local minima. Crossover rate and Mutation rate get modified using the equation 4.36 , 4.37, 4.38 and 4.39.

$if\ (fitness \geq \max\_fitness)\ then$

$$Crossover\_Prob = Crossover\_Prob_1 - \frac{(Crossover\_Prob_1 - Crossover\_Prob_2)(fitness - avg\_fitness)}{(\max\_fitness - avg\_fitness)} \quad (4.36)$$

$$Mutation\_Prob = Mutation\_Prob_1 - \frac{(Mutation\_Prob_1 - Mutation\_Prob_2)(fitness - avg\_fitness)}{(\max\_fitness - avg\_fitness)} \quad (4.37)$$

$else$

$$Crossover\_Prob = Crossover\_Prob_1 \quad (4.38)$$

$$Mutation\_Prob = Mutation\_Prob_1 \quad (4.39)$$

Where $\max\_fitness$ is the highest fitness value in the population. $avg\_fitness$ is the average fitness value in every population. $fitness$ is higher fitness value between two individuals. $Crossover\_Prob_1 = 1.0$, $Crossover\_Prob_2 = 0.7$ and $Mutation\_Prob_1 = 0.2$, and $Mutation\_Prob_1 = 0.01$. The parameters used in this work were set based on the experimental results, the parameter value that show the highest performance was chosen to be used in the implementation of the algorithm. Population size is usually fixed in this experiment. String length also usually fixed and a probability of crossover is kept high and a probability of mutation is kept low. The maximum chromosome length is 300 characters. The run of GA is stopped after a fixed number of generations. The solution is the best chromosome of the final generation. Thus, the genetic operations used to update the population are uniform crossover with probability pc (probability of crossover) $= 0.6\ to\ 0.9$ and mutation with probability pm (probability of mutation) $= .001\ to\ 0.1$. The probability of the function SR is 0.5, and all other function are of probability 0.5. Finally, the maximum LFSR length is 20 bits. The run of this proposed algorithm is stopped after a fixed number of iterations depend on resource available in wireless communication. The solution is the best keystream (chromosome) of the final iteration. The figure 4.12 shows the flowchart of GA based keystream generation and section 4.2.1.2.1 presents the complete encryption/decryption keystream generation algorithm.

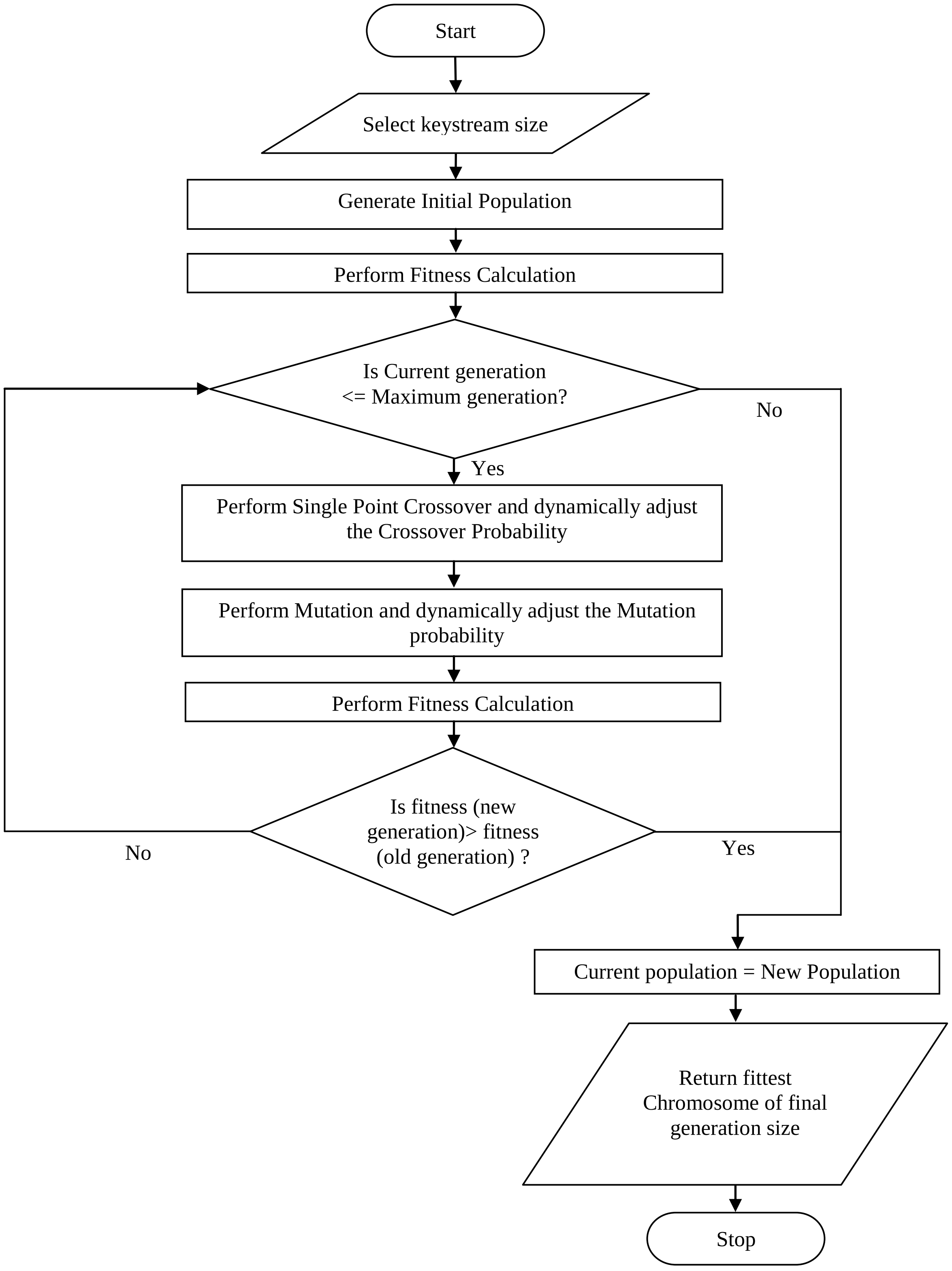

Start
Select keystream size
Generate Initial Population
Perform Fitness Calculation
Is Current generation <= Maximum generation?
No
Yes
Perform Single Point Crossover and dynamically adjust the Crossover Probability
Perform Mutation and dynamically adjust the Mutation probability
Perform Fitness Calculation
Is fitness (new generation)> fitness (old generation) ?
No
Yes
Current population = New Population
Return fittest Chromosome of final generation size
Stop

Figure 4.12: Flow chart of GA based keystream generation

4.2.1.2.1 Genetic Algorithm based Fittest Keystream Generation Algorithm

GA based encryption/decryption keystream generation algorithm takes length of the keystream and maximum number of iterations as an input. After the final number of iteration algorithm generates the fittest keystream as an output. The maximum number of iterations depends on the resource available in wireless communication.

Input : Length of the keystream, Maximum number of iteration

Output : Genetic Algorithm based best fittest keystream (chromosome) at the final iteration

Method : The process performs Genetic Algorithm procedure on set of keystream and finally produces best fittest keystream. .

```
Step 1.   Generate the initial population (pop) randomly.
Step 2.   Evaluate the Population.
Step 3.   Perform the following steps until maximum number of generation reach.
    Step 3. 1.   Generate a new population (pop1) by applying crossover and mutation and self acclimatizing adjustment of the population size, crossover and mutation probabilities
    Step 3. 2.   Evaluate the fitness of the new generated chromosomes of pop1.
    Step 3. 3.   Calculate the averages of fitness values for pop and pop1, av and av1 respectively.
    Step 3. 4.   $If\ (new\ fitness > old\ fitness)$ then replace the old population by the new one.
Step 4.   Return the best chromosome of the final generation
```

The GA based fittest keystream is used to perform the encryption operation on the plaintext. The detail step of GA based encryption process is given in section 4.2.1.3.

4.2.1.3 Encryption Algorithm

Input : *Source file/source stream i.e. plaintext*

Output : *Encrypted file/encrypted stream i.e. cipher text*

Method : *The process operates on binary stream and generates encrypted bit stream through Genetic Algorithm (GA) based encryption.*

Step 1. *Perform Exclusive-OR with Genetic Algorithm (GA) generated* $128/192/256$ *bits key and the plaintext to form intermediate cipher text. If the size of the plaintext to be encrypted is larger than* $128/192/256$ *bits then square edge extension based keystream expansion strategy get perform to expand the GA based keystream and then expanded keystream get Exclusive-OR with the plaintext for forming the intermediate cipher text.*

Step 2. *Divide the outcomes of step 1 into variable blocks.*

Step 3. *For each block* $S = s_0\, s_1\, s_2\, s_3\, s_4 \dots s_{L-1}$ *of length* L *bits, where* $L = 8$ *perform the following operation in a stepwise manner to generate the target block* $T = t_0\, t_1\, t_2\, t_3\, t_4 \dots t_{L-1}$ *of the same length* (L).

Step 3. 1. *Corresponding to the each block* $S = s_0\, s_1\, s_2\, s_3\, s_4 \dots s_{L-1}$, *evaluate the equivalent decimal integer,* D_L.

Step 3. 2. *Apply step 3.3 and step 3.4 exactly* L *number of times, for the values of the variable* P *ranging from* 0 *to* $(L-1)$ *increasing by 1 after each execution of the loop.*

Step 3. 3. *Apply modulo-2 operation on* D_{L-P} *to check if* D_{L-P} *is even or odd.*

Step 3. 4. *If* D_{L-P} *is found to be even, compute* $D_{L-P-1} = D_{L-P}/2$, *where* D_{L-P-1} *is its position in the series of natural even numbers. Assign* $t_P = 0$.

If D_{L-P} *is found to be odd, compute* $D_{L-P-1} = (D_{L-P} + 1)/2$, *where* D_{L-P-1} *is its position in the series of natural odd numbers. Assign* $t_P = 1$.

Step 3. 5. With the values of all the t_P's being available, p ranging from 0 to $(L-1)$, $T = t_0\, t_1\, t_2\, t_3\, t_4 \ldots t_{L-1}$ constructs the target block corresponding to $S = s_0\, s_1\, s_2\, s_3\, s_4 \ldots s_{L-1}$.

Step 4. Merge all the encrypted blocks of step 3.

The detail of square edge based keystream expansion in step 1 is discussed in section 4.2.1.3.1. Step 2 of the algorithm is used to divide the outcomes of step 1 in variable blocks. After that in step 3 a modulo-2 based even odd checking operation is performed on each block. Finally, in step 4 all the encrypted blocks of previous step is merged together to generate GA based encrypted text.

4.2.1.3.1 Square Edge Extension based Keystream Expansion Technique

If the size of the plaintext to be encrypted is larger than $128/192/256$ bits then square edge extension based key expansion strategy get perform to expand the keystream. Considers the keystream as a stream of finite number of bits N, and is divided into a finite number of blocks, each also containing a finite number of bits n, where $1 \leq n \leq N$.

Let $K = k_0^0\, k_1^0\, k_2^0\, k_3^0\, k_4^0 \ldots k_{n-1}^0$ is a block of size n in the plaintext. Then the first intermediate block $I_1 = k_0^1\, k_1^1\, k_2^1\, k_3^1\, k_4^1 \ldots k_{n-1}^1$ can be generated from K in the following way:

$$k_0^1 = k_0^0 \tag{4.40}$$

$$k_{n-1}^1 = k_{n-1}^0 \tag{4.41}$$

$$k_i^1 = k_{i-1}^1 \oplus k_i^0,\ 1 \times i \times (n-2); \tag{4.42}$$

$\oplus$ stands for the *Exclusive-OR* operation. Now, in the same way, the second intermediate block $I_2 = k_0^2\, k_1^2\, k_2^2\, k_3^2\, k_4^2 \ldots k_{n-1}^2$ of the same size (n) can be generated by:

$$k_0^2 = k_0^1 \tag{4.43}$$

$$k_{n-1}^2 = k_{n-1}^1 \tag{4.44}$$

$$k_i^2 = k_{i-1}^2 \oplus k_i^1,\ 1 \times i \times (n-2); \tag{4.45}$$

After this process continues for a finite number of iterations, which depends on the value of n, the source keystream block k is regenerated.

If the number of iterations required to regenerate the source block is assumed to be I, the generation of any intermediate or the final block can be generalized as follows:

$$k_0^j = k_0^{j-1} \tag{4.46}$$

$$k_{n-1}^j = k_{n-1}^{j-1} \tag{4.47}$$

$$k_i^j = k_{i-1}^2 \oplus k_i^{j-1}, 1 \times i \times (n-2); \text{ where } 1 \times j \times I. \tag{4.48}$$

In this generalized formulation system, the final block, which in turn is the source keystream block, is generated when $j = I$.

Figure 4.13 shows the different color side, black side represents the original key, red and blue side represents the left and right side of square.

Figure 4.13: Different color side, black side represents the original key, red and blue side represents the left and right side of square

Any of the intermediate state is attached at the front and end of the original keystream. The new expanded keystream shown in figure 4.14.

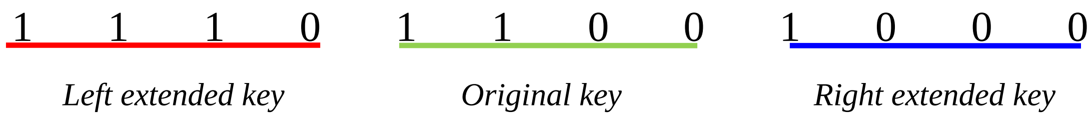


Figure 4.14: Expanded keystream

Bits of the left edge of the square (i.e. 1110) is generated at the front of the original key and bits of the right edge of the square (i.e. 1000) is attached at the end. As per keystream expansion strategy the new expand key will be three times longer than original one.

4.2.1.4 Session Key based Encryption

During final step of the technique a cascaded *Exclusive-OR* operation between CDHLP synchronized session key and GA encrypted cipher text is performed to generate final encoded cipher text.

The decryption algorithm takes the cipher text as a binary stream of bits and perform first level of operation using CDHLP generated synchronized session key to produce intermediate decrypted text. Finally, GA generated fittest keystream based decryption is performed on the intermediate decrypted text to regenerate the plaintext. The algorithm for the complete process is given in section 4.2.2.

4.2.2 CDHLPSCT Algorithm at Receiver

Input : Encrypted file/encrypted stream i.e. cipher text

Output : Source file/source stream i.e. plaintext

Method : The process operates on encrypted binary stream and generates decrypted bit stream through Chaos based DHLP guided Genetic algorithm (GA) based decryption operations.

Step 1. Perform cascaded Exclusive-OR operation between CDHLP based session key and cipher text.

Step 2. Perform Genetic Algorithm (GA) based decryption on the outcomes of the step 1 to regenerate starting combination i.e. plaintext.

Step 1 of the algorithm is discussed in section 4.2.2.1. Step 2 of the algorithm for performing Genetic Algorithm based decryption is discussed in section 4.2.2.2.

4.2.2.1 Session Key based Decryption

Initially cascaded *Exclusive-OR* operation between CDHLP synchronized session key and cipher text is performed to produce session key decrypted text. Outcomes of this operation used as an input of GA based decryption algorithm discussed in 4.2.2.2 to regenerate the plaintext.

In the decryption process the GA based cipher text is divided into blocks. Modulo-2 guided odd even based decryption is performed on each block. After that all blocks are merged together. The GA generated keystream is use to *Exclusive-OR* with the merged blocks to regenerate the plaintext. The detail step of GA based decryption process is given in section 4.2.2.2.

4.2.2.2 GA based Decryption Algorithm

Input : GA encrypted file/ GA encrypted stream

Output : Source file/source stream i.e. plaintext

Method : The process operates on GA encrypted bit stream and regenerates the plaintext through GA based decryption.

```
Step 1.    Divide the GA encrypted text into different blocks.
Step 2.    Perform decryption operation on each block of step 1. For each block
           $T = t_0\ t_1\ t_2\ t_3\ t_4 \ldots t_{L-1}$ of length $L$ bits, the following scheme is
           followed in a stepwise manner.
           Step 2. 1.   Set $P = L - 1$ and $T = 1$.
           Step 2. 2.   Repeat step 3.3 and step 3.4 for the value of $P$ ranging
                        from $(L - 1)$ to $0$.
           Step 2. 3.   If $t_P = 0$, $T = T^{th}$ even number in the series of natural
                        even numbers;
                        If $t_P = 1$, $T = T^{th}$ odd number in the series of natural
                        even numbers.
           Step 2. 4.   Set $P = P - 1$.
```

Step 2. 5. *Convert T into the corresponding stream of bits*

$$S = s_0\, s_1\, s_2\, s_3\, s_4 \dots s_{L-1}.$$

Step 3. *Merge outcomes of step 2.*

Step 4. *Check if the length of the GA based keystream is less than the length of outcomes of step 3 then perform triangle edge based key expansion method to enhance the length of the keystream. Otherwise, select the* 128 *bit fittest keystream for decryption.*

Step 1. *Finally, perform Exclusive-OR operation between outcomes of step 3 and GA generated fittest encryption keystream of same length to produce the plaintext.*

4.3 Implementation

Consider Initial population size as 200 and randomly generated each keystream having 128 bits. The population gets evaluated with the help of fitness function using generations through a fitness technique which consist of number of statistical tests to examine whether the pseudorandom number sequences are sufficiently random or not.

On receipt of fittest generation the proposed GA based key generation algorithm let generate the best fittest keystream having length of 128 bits. Let the binary form of 128 bits GA based keystream is
10011011/01011110/11001101/10010111/01010100/11010001/10101010/10110011/
01001010/01110001/01010101/10011100/11111001/01101110/11011111/00110101

Consider the plaintext to be encrypted is "**Network Security**", binary representation of the ASCII value of plaintext is
01001110/01100101/01110100/01110111/01101111/01110010/01101011/00100000/
01010011/01100101/01100011/01110101/01110010/01101001/01110100/01111001
Here "/" is used as the separator between successive bytes.

Perform *Exclusive-OR* operation between plaintext and GA based keystream. So, GA based key stream encoded intermediate cipher text is
11010101/00111011/10111001/11100000/00111011/10100011/11000001/10010011/

00011001/00010100/00110110/11101001/10001011/00000111/10101011/01001100

Divide the intermediate cipher text into different segments illustrate below. Here segment of variable size has been chosen. Say, following are the different stream segments constructed from S (level 1 encoded text):

S_1 = 11010101001110111011100111100000 (32 bits)

S_2 = 00111011110100011110000011001001 1 (32 bits)

S_3 = 00011001000101000011011011101001 (32 bits)

S_4 = 10001011000001111010101101001100 (32 bits)

For the segment S_1, corresponding to which the decimal value is $(3577461216)_{10}$, the process of encryption is shown below:

$3577461216 \rightarrow 1788730608^{0} \rightarrow 894365304^{0} \rightarrow 447182652^{0} \rightarrow 223591326^{0} \rightarrow 111795663^{0} \rightarrow 55897832^{1} \rightarrow 27948916^{0} \rightarrow 13974458^{0} \rightarrow 6987229^{0} \rightarrow 3493615^{1} \rightarrow 1746808^{1} \rightarrow 873404^{0} \rightarrow 436702^{0} \rightarrow 218351^{0} \rightarrow 109176^{1} \rightarrow 54588^{0} \rightarrow 27294^{0} \rightarrow 13647^{0} \rightarrow 6824^{1} \rightarrow 3412^{0} \rightarrow 1706^{0} \rightarrow 853^{0} \rightarrow 427^{1} \rightarrow 214^{1} \rightarrow 107^{0} \rightarrow 54^{1} \rightarrow 27^{0} \rightarrow 14^{1} \rightarrow 7^{0} \rightarrow 4^{1} \rightarrow 2^{0} \rightarrow 1^{0} \rightarrow 1^{1}$.

So, T_1 =000001000110001000100011010101001

For the segment S_2, corresponding to which the decimal value is $(1000587667)_{10}$, the process of encryption is shown below:

$1000587667 \rightarrow 500293834^{1} \rightarrow 250146917^{0} \rightarrow 125073459^{1} \rightarrow 62536730^{1} \rightarrow 31268365^{0} \rightarrow 15634183^{1} \rightarrow 7817092^{1} \rightarrow 3908546^{0} \rightarrow 1954273^{0} \rightarrow 977137^{1} \rightarrow 488569^{1} \rightarrow 244285^{1} \rightarrow 122143^{1} \rightarrow 61072^{1} \rightarrow 30536^{0} \rightarrow 15268^{0} \rightarrow 7634^{0} \rightarrow 3817^{0} \rightarrow 1909^{1} \rightarrow 955^{1} \rightarrow 478^{1} \rightarrow 239^{0} \rightarrow 120^{1} \rightarrow 60^{0} \rightarrow 30^{0} \rightarrow 15^{0} \rightarrow 8^{1} \rightarrow 4^{0} \rightarrow 2^{0} \rightarrow 1^{0} \rightarrow 1^{1}$.

So, T_2 =1011011001111100001110100010001

For the segment S_3, corresponding to which the decimal value is $(420755177)_{10}$, the process of encryption is shown below:

$420755177 \rightarrow 210377589^{1} \rightarrow 105188795^{1} \rightarrow 52594398^{1} \rightarrow 26297199^{0} \rightarrow 13148600^{1} \rightarrow 6574300^{0} \rightarrow 3287150^{0} \rightarrow 1643575^{0} \rightarrow 821788^{1} \rightarrow 410894^{0} \rightarrow 205447^{0} \rightarrow 102724^{1} \rightarrow 51362^{0} \rightarrow 25681^{0} \rightarrow 12841^{1} \rightarrow 6420^{1} \rightarrow 3210^{0} \rightarrow 1605^{0} \rightarrow 803^{1} \rightarrow 402^{1} \rightarrow 201^{1} \rightarrow 101^{1} \rightarrow 51^{1} \rightarrow 26^{1} \rightarrow 13^{0} \rightarrow 7^{1} \rightarrow 4^{1} \rightarrow 2^{0} \rightarrow 1^{0} \rightarrow 1^{1}$.

So, T_3 =11101000100100110011111011001

For the segment S_4, corresponding to which the decimal value is $(2332535628)_{10}$, the process of encryption is shown below:

$2332535628 \rightarrow 1166267814^{0} \rightarrow 583133907^{0} \rightarrow 291566954^{1} \rightarrow 145783477^{0} \rightarrow$
$72891738^{0} \rightarrow 36445869^{0} \rightarrow 18222935^{1} \rightarrow 9111468^{1} \rightarrow 4555734^{0} \rightarrow 2277867^{0} \rightarrow$
$1138934^{1} \rightarrow 569467^{0} \rightarrow 284734^{1} \rightarrow 142367^{0} \rightarrow 71184^{1} \rightarrow 35592^{0} \rightarrow 17796^{0} \rightarrow 8898^{0} \rightarrow$
$4449^{0} \rightarrow 2225^{1} \rightarrow 1113^{1} \rightarrow 557^{1} \rightarrow 279^{1} \rightarrow 140^{1} \rightarrow 70^{0} \rightarrow 35^{0} \rightarrow 18^{1} \rightarrow 9^{0} \rightarrow 5^{1} \rightarrow 3^{1} \rightarrow$
$2^{1} \rightarrow 1^{0} \rightarrow 1^{1}$.

So, T_4 =00100011001010100001111001011101

The following stream is constructed on merging segments T_1, T_2, T_3 and T_4.

00000010/00110001/00010001/10101010/01101101/10011111/00001110/10001000/
11110100/01001001/10011111/10110010/01000110/01010100/00111110/01011101

Let Chaos based Double Layer Perceptron (CDHLP) generated 128 bits following session key is generated
11100011/01001100/11011101/01100110/01010011/11000010/10010101/11010110/
01101101/01011001/01101101/01100111/11010101/01011110/01001101/11101010

Session key encrypted final cipher text produce on performing *Exclusive-OR* operation between merged segments T_1, T_2, T_3 and $T_4$and session key.
10100101/01101110/11101000/00101011/11100000/00100011/01000100/11001000/
10011001/00010000/11110010/11010101/10010011/00001010/01110011/10110111.

4.4 Security Analysis

The security of DHLP based technique proposed in chapter 3 has been enhanced in chapter 4 by introducing chaos synchronization and authentication step during synchronization to prevent synchronization of unauthorized entity. In this section some of the attacks are considered to check the immunity power of the proposed cryptographic technique against the attack. In key exchange protocol the major threat is the attacker who resides in the middle of the sender and receiver has access to all the messages exchanged by both synchronizing parties, also he/she knows all about the protocol details. The following standard attacks are considered to ensure the robustness of the proposed technique.

- *Attacks during synchronization attempts:* In this type of attack, the attacker tries to synchronize with the chaotic system by eavesdropping on all the messages exchanged by sender and receiver. This type of attack will not work as the attacker does not know the initial conditions of any of the z components of any of the chaotic systems, and also the parameters a and r are hidden too. By nature, the Lorenz system is very sensitive to initial conditions meaning that the error between attacker and receiver is going to grow exponentially if there is a very slight difference between their initial conditions. The main difference between receiver and attacker is that the output of receiver (y_2) influences the sender chaotic system and hence affects its output (x_1) resulting in a lack of synchronization between sender and attacker.
- *Attacks by solving the chaotic system differential equations:* As the nature of chaotic systems, the problem of solving the system of differential equations representing the system is proven to be very hard. Numerical solution is of no use due to the approximation nature of the numerical methods and the butter fly effect of chaotic systems.
- *Cipher text only Attack:* This technique nullifies the success rate of this attack by producing a completely random GA based encryption/decryption keystream. The strength of resisting exhaustive key search attack relies on a large key space. Initially, GA based large keystream is used to encrypt the plaintext after that, outcomes of this passes through CDHLP guided encryption. So, cipher text produces by this proposed technique is mathematically difficult to break. This method makes it difficult for the hacker to find out the keystream used for encryption. Proposed methodology helps to generate long period of random keystreams along with no obvious relationship between the individual bits of the sequence. Also the generated keystreams are of large linear complex. Finally key stream have high degrees of correlation immunity. Thus it is practically difficult to perform a brute-force search in a key-space.
- *Known Plaintext Attack:* The technique offers better floating frequency of characters and in GA based encryption technique cycle formation operation also enhance the security of the technique. So, known plaintext attack is difficult in this proposed technique.

- *Chosen Plaintext Attack:* Proposed technique passes the frequency (monobit) test, runs test, binary matrix rank test and in each session a fresh CDHLP based session key is used for encryption which confirms that chosen plaintext attack is very difficult in this technique.
- *Chosen Cipher text Only Attack:* Proposed technique passes the discrete Fourier transform test, approximate entropy test, overlapping (periodic) template matching test which confirms that chosen plaintext attack is difficult in this technique.
- *Brute Force Attack:* In CDHLPSCT, security is improve by increasing the synaptic depth L of the CDHLP. The security increases proportional to L^2 while the probability of a successful attack decreases exponentially with L. The approach is thus regarded computationally secure with respect to these attacks for sufficiently large L. For a brute force attack using $K1$ hidden neurons in layer 1, $K2$ hidden neurons in layer 2, $K1 \times N$ input neurons and boundary of weights L, gives $(2L+1)^{(K1\times N+K1\times K2)}$ possibilities. For example, the configuration $K1 = 3$, $K2 = 3$, $L = 3$ and $N = 100$ gives $(2\times 3+1)(3\times100+3\times3)$ key possibilities, making the attack difficult.
- Consider an attack where eavesdropper E just trains a third CDHLP with the examples consisting of input vector Y and output bits τ^{A}. These can be obtained easily by intercepting the messages transmitted by the partners over the public channel. E's CDHLP has the same structure as A's and B's and starts with random initial weights, too. In each time step the attacker calculates the output of his/her CDHLP. Afterwards E uses the same learning rule as the partners, but τ^{E} is replaced by τ^{A}. Thus the update of the weights is given by one of the following equations 4.49, 4.50 and 4.51:

Hebbian learning rule: $w_{i,j}^{E+} = g\left(w_{i,j}^{E} + x_{i,j}\tau^{A}\Theta\left(\sigma_i^{E}\tau^{A}\right)\Theta\left(\tau^{A}\tau^{B}\right)\right)$ (4.49)

Anti-Hebbian learning rule: $w_{i,j}^{E+} = g\left(w_{i,j}^{E} - x_{i,j}\tau^{A}\Theta\left(\sigma_i^{E}\tau^{A}\right)\Theta\left(\tau^{A}\tau^{B}\right)\right)$ (4.50)

Random Walk learning rule: $w_{i,j}^{E+} = g\left(w_{i,j}^{E} + x_{i,j}\Theta\left(\sigma_i^{E}\tau^{A}\right)\Theta\left(\tau^{A}\tau^{B}\right)\right)$ (4.51)

So, E uses the internal representation $(\sigma_1, \sigma_2, \ldots, \sigma_K)$ of his/her own network in order to estimate A's, even if the total output is different. As $\tau^A \neq \tau^E$indicates that there is at least one hidden unit with $\sigma_i^A \neq \sigma_i^E$, this is certainly not the best algorithm available for an attacker.

- Consider an attack at the time of CDHLP synchronization process where the attacker (E) can imitates one of the parties (A or B), but if attacker output disagrees with the imitated party's output $\tau^E \neq \tau^A$, attacker certainly knows that either one or all hidden units are mistaken. In order to get $\tau^E = \tau^A$ attacker negates the sign of one of attacker's hidden units. As $\sigma = sgn(h)$ the unit most likely to be wrong is the one with the minimal $|h|$, therefore that is the unit which is negate. This policy results a immense enhancement in the attacker's achievement. It can be seen that the success rate is quite high for all L values presented, but it drops exponentially as L increases. On the other hand parties' synchronization time increases like L^2, and therefore it can be conclude that in the boundary of large L values the proposed technique is secure against the this attack.

4.5 Discussions

The technique is simple and easy to implement in various high level language. The test results show that the performance and security provided by the CDHLPSCT is good and comparable to standard technique. The security provided by the proposed technique is comparable with other techniques. To enhance the security of the technique, CDHLPSCT offers chaos synchronization between sender and receiver for generating identical seed value for generating common input vector. During chaos synchronization some parameters which take major roles for synchronization of two end never get transmit through private channel, which confirms the security against MITM attack. Also the technique introduces an authentication step which nullifies the possibility of synchronization of unauthorized entity and also prevents MITM attack. Since the encryption and decryption times are much lower, so processing speed is very high. Proposed method takes minimum amount of resources which is greatly handle the resource constraints criteria of wireless communication. CDHLPSCT outperform than existing TPM, PPM and method proposed in chapter 2 and chapter 3. No platform specific optimizations were done in the actual implementation, thus performance should be similar over varied implementation platform. The whole procedure is randomized, thus resulting in a unique process for a unique session, which makes it harder for a cryptanalyst to find a base to start with. This technique is applicable to ensure security in message transmission in any form and in any size in wireless communication.

Some of the salient features of CDHLPSCT are summarized as follows:

a) *Session key generation and exchange – Identical session key can be generate after the tuning of CDHLP in both sender and receiver side with the help of chaos synchronization. So, no need to transfer the whole session key via vulnerable public channel.*

b) *Degree of security – Proposed technique does not suffers from cipher text only attack, known plaintext attack, chosen plaintext attack, chosen cipher text only attack, brute force attack and attacks during CDHLP synchronization process. It offers authentication steps during synchronization.*

c) *Variable block size – Encryption algorithm can work with any block length and thus not require padding, which result identical size of files both in original and encrypted file. So, CDHLPSCT has no space overhead.*

d) *Variable key –* $128/192/256$ *bit CDHLP based session key and* $128/192/256$ *bits GA based encrypted keystream with high key space can be used in different sessions. Since the session key is used only once for each transmission, so there is a minimum time stamp which expires automatically at the end of each transmission of information. Thus the cryptanalyst may not be able guess the session key for that particular session.*

e) *Complexity – The technique has the flexibility to adopt the complexity based on infrastructure, resource and energy available for computing in a node or mesh through wireless communication. So, the proposed technique is very much suitable in wireless communication.*

f) *Non-homogeneity – Measures of central tendency, dispersion and Chi-Square value have been performed between the source and corresponding cipher streams generated using proposed technique. All measures indicate that the degree of non-*

homogeneity of the encrypted stream with respect to the source stream is good. This technique has a better Chi-Square value than technique proposed in chapter 2 and 3.

g) *Floating frequency – In CDHLPSCT it is observed that floating frequencies of encrypted characters are indicates the high degree of security for the proposed technique. This technique has a better floating frequency than technique proposed in chapter 2 and 3.*

h) *Entropy – The entropy of encrypted characters is near to eight which indicate the high degree of security of technique. This technique also has a better entropy value than technique proposed in chapter 2 and 3.*

i) *Correlation – The cipher stream generated through CDHLPSCT is negligibly correlated with the source stream. Therefore the proposed technique may effectively resist data correlation statistical attack.*

j) *Key sensitivity – The technique generates an entirely different cipher stream with a small change in the key and technique totally fails to decrypt the cipher stream with a slightly different secret session key.*

k) *Security and performance trade-off – The technique may be ideal for trade-off between security and performance of light weight devices having very low processing capabilities or limited computing power in wireless communication.*

Chapter 5

Chaos based Triple Hidden Layer Perceptron Synchronized Cryptographic Technique (CTHLPSCT)

5.1 Introduction

In this chapter a novel soft computing assisted cryptographic technique CTHLPSCT, based on synchronization of Chaos based two Triple Hidden Layer Perceptron (CTHLP), one at sender and another at receiver has been proposed. The CDHLPSCT technique proposed in chapter 4 had some drawbacks like for the increased of length of the session key if number of neurons in input layer get increased then it in turn increase the number of synaptic links (weight) between input layer and hidden layer. So, the synchronization also get increased. Again large diversity among each weight values generated randomly can slower down the synchronization process. Also in the CDHLPSCT technique proposed in chapter 4, authentication steps are performed after the synchronization steps which consumes significant amount of time for authentication purpose. Proposed CTHLPSCT method of this chapter eliminates all the above stated drawbacks of the CSHLPSCT in chapter 4. This novel method of presented in this chapter introduces chaos based Triple Hidden Layer Perceptron (CTHLP) synchronization mechanism where CTHLP uses three hidden layers.[206] Addition of one extra layer enhances the security by making the complex internal structure of the CTHLP. It will be difficult for the attacker to guess the internal structure of the proposed CTHLP. Here, number of neurons in input layer does not get increased as the increased of length of the session key because neurons in extra hidden layers helps to increased the key length. Also a novel parallel key exchange and authentication techniques using secret common input vector has been proposed. So, attacker can't be able to make distinguish between synchronization steps and authentication steps. At the time of key exchange procedure key authentication technique is performed parallel by selecting last m bits of the identical input vector and transmitting directly as an output bit towards the other party over public channel. Receiving party checks these last m bits to its last m bits of identical input vector. If both the sequences are same then both are authenticated otherwise authentication fails.

Here, CTHLP based synchronization is performed for tuning both sender and receiver. On the completion of the tuning phase identical session keys is generated at the both end with the help of synchronized CTHLP. This synchronized network can be used for transmitting message using any light weight encryption/decryption technique with the help of session key of the synchronized network. To illustrate the cryptographic technique using CTHLP in wireless communication one of the simple and secure encryption/decryption technique has been presented. A plaintext is considered as a stream of binary bits. Ant Colony Intelligence (ACI) guided enciphering technique[207] with the help of CTHLP tuned session key is used to generate the cipher text. The plaintext is regenerated from the cipher text using same technique with the help of CTHLP tuned session key at the receiver.

Section 5.2 represents a description of proposed technique in detail. Section 5.3 deals with the implementation of the proposed cryptographic technique. Section 5.4 discussed the security issues related to the proposed technique. Discussions are presented in section 5.5.

5.2 The Technique

The technique performs the CTHLP based synchronization for generation of secret session key at both ends. This synchronized session key of the tuned network is used for the transmission of secured message through wireless network with the help of any light weight encryption/decryption algorithm. To illustrate the cryptographic technique in wireless communication one of the simple and secure encryption/decryption technique has been proposed, where plaintext (i.e. the input file) is considered as a stream of binary bits, which is encrypted using ACI generated fittest encryption/decryption keystream. The session key based on CTHLP is used to encrypt intermediate output which produces final cipher text. Identical CTHLP is used to tune sender and receiver to generate the secret session at both end. Session key is generated by performing CTHLP based synchronization and authentication procedure in parallel between sender and receiver side using secret common input vector. In this proposed technique for key generation purpose both sender and receiver uses its own CTHLP having identical structure of three hidden layers. Both parties' uses identical input vector with the help of Chaos synchronized common seed value and use anonymous random weight vector to initializes the weights of the synaptic links of CTHLP. Identical input vector for both the parties kept secret for security reason. Depending on these input vector and weights value both the machine produces some output value. Then these output values is transmitted to the receiver over public channel. If output values are same for both the sender and receiver machine then CTHLP learning step is applied to both machines for synchronization purpose. When the full synchronization is achieved both machines then produce identical weights vector which is used as a secret session key.

Ant Colony Intelligence (ACI) based encryption/decryption procedure is used to produce group of characters based on distribution of characters in the plaintext known as keystream having a size less than or equal to the length of the plaintext. As the ants move, they deposit a chemical substance called pheromone on their path. In the ACI based keystream generation technique pheromone composed of characters that imply the key. In this technique an ant agent having a pheromone deposition consisting of a group of alphanumeric characters is called a keystream and each character in the key stream is known as key. Plaintext is examined to find out total number of characters matched with the characters presents in pheromone of an ant agent. Each ant has an energy level which is computed by counting number of characters in the plaintext matched with the pheromone characters (key) divided by the total number of character in the pheromone (key). A threshold value is selected to evaluate against energy level of each ant agent. Ant agent having highest energy level more than predefined threshold value is selected as a keystream for encryption. If the length of the plaintext is grater than the length of the ACI based keystream then the values of the keystream are added to a predetermined value to generate the keys for the characters in the plaintext which is at a position grater than the length of the keystream. Stream of plaintext is then encrypted using the ACI based keystream/extended keystream. Finally a cascaded *Exclusive-OR* operation is performed between ACI encrypted text and the CTHLP based session key to generate final cipher text.

Receiver has same CTHLP synchronized session key. This session key is used to perform first step of the deciphering technique. In the next step, ACI guided keystream based deciphering operation gets performed to regenerate the plaintext.

The CTHLPSCT does not cause any storage overhead. This greatly handles the resource constraints criteria of wireless communication. A comparison of CTHLPSCT with previously proposed technique in chapter 4, chapter 3, chapter 2, existing Tree Parity Machine (TPM), Permutation Parity Machine (PPM), and industry accepted AES, RC4, Vernam Cipher, Triple DES (TDES) and RSA have been done. Analyses of results are given in chapter 7.

In CTHLPSCT, encryption algorithm takes the plaintext as a binary stream of bits which is encrypted using ACI generated fittest encryption keystream based encryption process. Chaos based THLP synchronized session key is used to further encrypt the ACI encoded text to produce final cipher text. The algorithm for the complete process is given in section 5.2.1.

5.2.1 CTHLPSCT Algorithm at Sender

Input : Source file/source stream i.e. plaintext

Output : Encrypted file/encrypted stream i.e. cipher text

Method : The process operates on binary stream and generates encrypted bit stream through CTHLP guided Ant Colony Intelligence (ACI) based encryption operations.

Step 1. Perform tuning of sender's and receiver's CTHLP to generate common secret session key.

Step 2. Generates ACI based fittest encryption keystream.

Step 3. Perform ACI based encryption operation on the plaintext.

Step 4. Perform cascaded Exclusive-OR operation between CTHLP based session key and outcomes of step 3.

Step 1 of the algorithm generate common session key through synchronization of CTHLP at both end. The detailed step is discussed in section 5.2.1.1. Step 2 of the algorithm generates ACI based fittest encryption keystream. The detailed description of the process is given in section 5.2.1.2. Algorithm for performing ACI based encryption operation (step 3) on the plaintext is discussed in 5.2.1.3. The technique of cascading encryption process (step 4) which takes the intermediate output generated in step 3 is given in details in section 5.2.1.4.

5.2.1.1 Chaos based Triple Hidden Layer Perceptron (CTHLP) Synchronization and Session Key Generation

Chaos based Triple Hidden Layer (CTHLP) guided synchronization mechanism has been proposed to improve the efficiency and enhance the security of the Chaos based Double Hidden Layer (CDHLP) guided synchronization, proposed in chapter 4. For the increased of length of the session key if number of neurons in input layer get increased then it in turn increase the number of synaptic links between input layer and hidden layer. So, the synchronization steps also get increased. Again large diversity among each weight values generated randomly can slower down the synchronization process. Also in the previously proposed CDHLPSCT method in chapter 4, authentication steps are performed after the synchronization steps which consumes significant amount of time for authentication purpose. The proposed method of the current chapter introduces chaos based two Triple Hidden Layer Perceptron (CTHLP) synchronization mechanism where CTHLP uses three hidden layers instead of two. Addition of this extra layer enhances the security by making the complex internal architecture. So, it will be difficult for the attacker to guess the internal architecture of the CTHLP. In CTHLP technique number of neurons in input layer does not get increased as the increased of length of the session key because neurons in extra hidden layers helps to increased the key length. So, number of input required in each iteration also gets minimized by minimizing the number of neurons in input layer. This also significantly improves the synchronization time. In this technique for key generation both sender and receiver uses its own machine having identical structure of three hidden layers. Both parties' uses identical input vector generated using Chaos synchronized seed and use anonymous random weight vector to initializes the weights of the synaptic links of CTHLP. Identical input vector for both the parties kept secret for security reason. Attackers has no idea about the internal state of both the machines at a particular instant of time and this is achievable by keeping secret the common input vector and internal state of the machine. Depending on these input vector and random weights value both the machine produces some output value. Then these output values is transmitted to the receiver over public channel. If output values are same for both the sender and receiver machine then CTHLP learning step is applied to both machines for synchronization purpose. When the full synchronization is achieved both machines then produce identical weights vector and which is use as a secret session key. At the time of key exchange key authentication technique is also performed parallel by selecting last m bits of the identical input vector and transmitting directly as an output bit towards the other party over public channel. Receiving party checks these last m bits to its last m bits of identical input vector. If both the sequences are same then both are authenticated otherwise not. Attacker does not have identical input vector like sender and receiver. By sniffing the public channel attacker can gets some bits but from them attacker will not be able to understand which one is output bit of the machine and which one is one of the bits of m bits sequence of the identical input vector. Even if attacker hacks the m bits then for getting the rest of the

$(d-m)$ bits of the identical input vector attacker has to perform checking with all $(d-m)$ combination that is computationally infeasible. Here d is the total number of bits in the identical input vector of proposed technique offers synchronization and authentication step in parallel. An attacker also cannot distinguish an authentication step from a synchronization step from observing the exchanged outputs. Attacker thus does not know, whether the currently observed output bit is used for either of the two purposes if the attacker does not know the secret identical common input vector. The figure 5.1 shows the single path from input neuron to the output neuron.

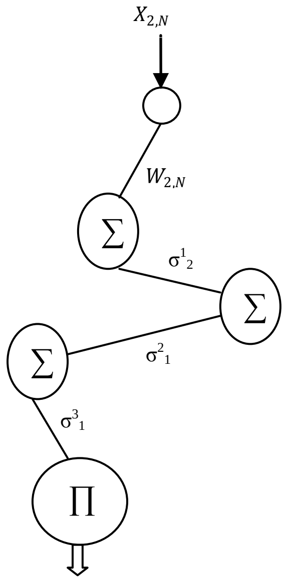


Figure 5.1: Snapshot of the single path from input neuron to the output neuron.

The figure 5.2 shows a perceptron with three hidden layers. Here the $K1 = 8$ and $K2 = 4$ and $K3 = 2$. So, the first hidden layer from the top has $K1 = 8$ hidden neurons. The second hidden layer has $K2 = 4$ hidden neurons. The third hidden layer has $K3 = 2$ hidden neurons. The total number of inputs neurons $= N \times K1$, where N is the number of inputs to each hidden neuron in layer 1.

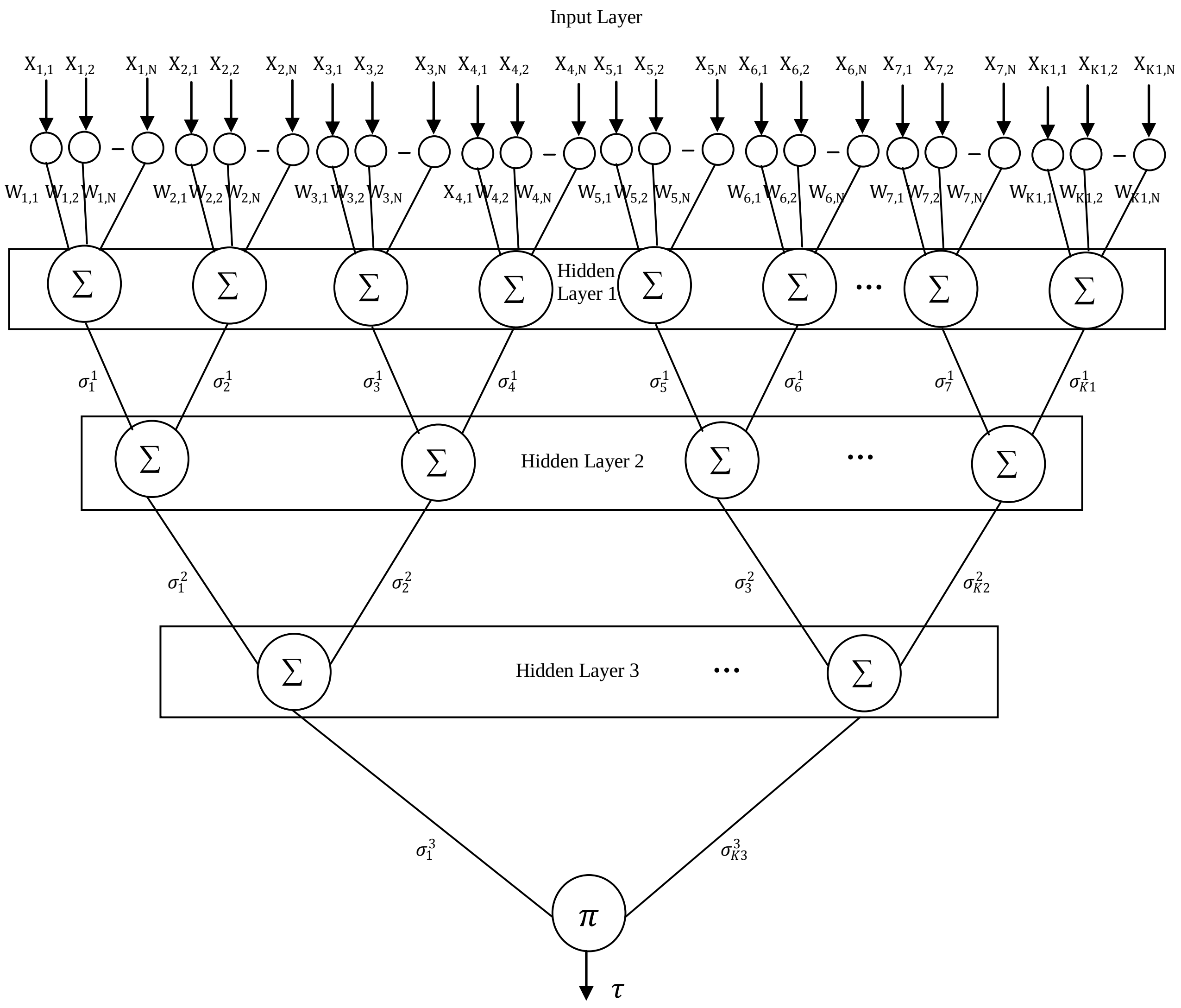


Figure 5.2: A CTHLP with three hidden layers

The CTHLP consist of one input layer, one output layer and three hidden layers instead of two hidden layers in DHLP and CDHLP technique in chapter 3 and chapter 4 respectively. Here, the parameter K is being divided into $K1, K2$ and $K3$ value. $K1$ hidden neurons resides in the hidden layer adjacent to the input layer. $K2$ represents number of hidden neurons in the middle hidden layer. For each $K1$ hidden neurons there are N inputs possible. So, finally it can be stated that, the input layer has $N \times K1$ input neurons. The size of the CTHLP is represented by $N \times K1 \times K2 \times K3$. Each hidden neuron in hidden layer number 1 produces $\sigma^1{}_i$ values. Similarly, each hidden neuron in hidden layer number 2 produces $\sigma^2{}_i$ value. Each hidden neuron in hidden layer number 3 produces $\sigma^3{}_i$ value. These can be represented using equation 5.1, 5.2 and 5.3.

$$\sigma^1{}_{\mathrm{i}} = sgn\left(\sum_{j=1}^{N} W_{i,j}\, X_{i,j}\right) \tag{5.1}$$

$$\sigma^2{}_{\mathrm{i}} = sgn\left(\sum_{j=1}^{N} \sigma_i^1\right) \tag{5.2}$$

$$\sigma^3{}_{\mathrm{i}} = sgn\left(\sum_{j=1}^{N} \sigma_i^2\right) \tag{5.3}$$

Sgn is a function, which returns $-1, 0$ or 1 illustrate in equation 5.4.

$$sgn = \begin{cases} -1 \;\; if\; x < 0 \\ 0 \;\;\; if\; x = 0 \\ 1 \;\; if\;\; x > 0 \end{cases} \tag{5.4}$$

The output of CTHLP is then computed as the multiplication of all values produced by hidden elements given in equation 5.5.

$$\tau = \prod_{i=1}^{K2} \sigma_i^3 \tag{5.5}$$

Total number of weights generated by the CTHLP is $(N \times K1 + K1 \times K2 + K2 \times K3)$. Each weight decimal value can be represented in eight bits binary. So, total $((N \times K1 + K1 \times K2 + K2 \times K3 \times 8)$ numbers of bits present in a weight (length of a session key). If $N = 2$, $K1 = 2, K2 = 3, K3 = 2, L = 5$ then $\big((2 \times 2 + 2 \times 3 + 3 \times 2) \times 8\big) = 128$ bits weight value act as a session key. Consider the synaptic depth i.e. weight limits $L = \pm 127$. So, eight binary bits are needed to represents each weight, where the MSB represents the sign bit and rest of the seven bits represents the magnitude of the weight.

In CTHLP based session key generation technique if sender's (A) and receiver's (B) do not have the identical input vector i.e.$\forall t : X^A(t) \neq X^B(t)$ then synchronization is not achievable between them. If the inputs are identical for both parties then only two parties can be trained using each other outputs. Given diverse inputs, the two parties are trying to learn totally dissimilar relations between inputs $X^{A/B}(t)$ and output $\tau^{A/B}(t)$ as result synchronization is not possible and thus in turn prevent the generation of time-dependent equal weights. The development of normalized sum of absolute differences $diff\big(W^A(t), W^B(t)\big) \in [0,1]$ over time for different offsets .$\forall t : X^A(t) = X^B(t + \varphi), \varphi \in N$ in the input vector and for completely different input vector. If attackers are deals with completely different set of inputs then attacker never synchronized with two parties. Because the distance between attackers and two parties that do not acquire the same inputs remains fluctuating within a certain limited range and never decreases towards zero. Two parties A and B with entirely diverse inputs illustrate the same qualitative performance. Taking into consideration the number of repulsive and attractive steps, it can be observed that on average there must be as many repulsive as attractive steps for such performance. Two parties A and B having the same inputs (offset zero) soon decrease their distance and synchronies. If both parties A and B uses identical inputs but a certain proportion of uniformly scattered 'noise' has been imposed on the transmitted outputs of either party. Despite of presence of noise in a certain time, the system would synchronies with a delay of approximately the duration of the noisy period plus the time used up for unproductive synchronization before the noisy period. So, if dissimilar random input vector are considered for two parties then the distance between the weights value of A and B is therefore not going to zero after each bounding action and the two parties deviate. So, no common inputs lead to the non-synchronization. For this reason common input of both parties i.e. $X^{A/B}(t)$ kept secret between the two parties in addition to their own arbitrarily assigned secret initial weights $W^{A/B}(t)$. Here, brute force attacks become computationally very costly because of $2^{\mathrm{K1 \times N}} - 1$ computations are needed for finding out possible common inputs. By this authentication scheme attack likes Man-In-The-Middle (MITM) attack and all other known attacks can be prevented.

The CTHLP technique usually allows splitting a protocol into an iterative procedure of comparatively light communication, as an alternative of a single (heavy) transmission which is not feasible in wireless communication because of resource constraints. Typically such a principle depends on random numbers in some way. The security that can be achieved is probabilistic, i.e. depending on the number of interactions, but security can always be increased beyond some acceptable variable security threshold. In CTHLP input of both parties acts as a common secret. The probability of an input vector $X^{A/B}(t)$ having a particular parity $p \in \{0, 1\}$ is 0.5. For authentication purpose this parity will at this moment use the output bit $\tau^{A/B}(t)$. At any given time t with common inputs for both parties, the probability of identical output is given in equation 5.6.

$$P(\tau^{A}(t) = p = \tau^{B}(t)) = \frac{1}{2} \tag{5.6}$$

Given a number n $(1 \leq n \leq \alpha)$ of pure authentication steps, in which one transmits the parity of the consequent input vector as output $\tau^{A/B}(t)$ directly, the probability that the two parties subsequently produce the same output n times (and thus are likely to have the same n

inputs) decreases exponentially with n i.e. $P(\tau^A(t) = p = \tau^B(t)) = \frac{1}{2^n}$ For statistical security of $\varepsilon \in [0, 1]$ select $n = \alpha$ authentication steps such that $1 - \frac{1}{2^\alpha} \geq \varepsilon$ which can be computed as $\alpha = \left\lceil log_2\left(\frac{1}{1-\varepsilon}\right)\right\rceil$ With $\alpha = 14$ the achievable statistical security $\varepsilon = 0.9999\ (i.e. 99.9999\ \%)$. The synchronization period for this technique therefore increases by α authentication steps depending on the necessary level of security ε. Select certain bit sub pattern in the input vector used for authentication only, such that the security threshold will be reached soon enough with a certain probability. Inputs are uniformly distributed so last m bit are also uniformly distributed. Now select those entries that possess a defined bit sub-pattern (e.g. '0101' for $m = 4$). The probability of such a fixed bit sub pattern of m bit to occur is $\frac{1}{2^m}$, because each bit has a fixed value with a probability of 0.5. Thus for four bit, on average every sixteenth input would be used for authentication. Authentication step is performed when the sub pattern arise and then one of the party send out the parity of the consequent input vector as output $\tau^{A/B}(t)$. This will only occur at the other party if it has the same inputs. Such an authentication does not manipulate the learning process at all. Because of the truth that the inputs are secret, an attacker cannot know when exactly such an authentication procedure takes place. Let the 64 bits identical input vector is 11011001/11010101/00010111/11100101/01011010/11110100/10100010/10100011.
Select the last fifteen bits of the identical input vector (where $m = 15$) 010001010100011 and transmit towards the other party. Attacker can have access the m bits from the public channel. Then for getting the rest of the $(d - m)$ bits of the identical input vector attacker has to perform checking with all $(d - m)$ combination. That is computationally difficult. Where, d is the total number of bits in the identical input vector. In this technique A always succeeds in convincing B by synchronies within a finite time if A knows the common secret i.e. the same inputs. In the case of the authentication principle, A will reach the security threshold ε in the specified α authentication steps. If A does not know the secret input of B then success probability of A becomes very small. As a result synchronization will be unsuccessful. The two parties will always be diverse by the repulsive steps. In the case of authentication principle, A will not reach the security threshold ε in the specified α authentication steps and will be rejected. At the time of synchronization procedure No information on the common secret is seep out at all. The only information transmitted is the

unknown bit-strings. In the case of the authentication the inputs are randomly chosen only for authentication principle. An attacker also cannot differentiate an authentication step from a synchronization step from observing the exchanged outputs. Attackers are not able to know whether the currently observed output bit is used for authentication or synchronization purpose if attackers do not know the common input vector.

In the CTHLP technique CTHLPs start synchronization by exchanging control frames. The process involves message integrity and synchronization test. CTHLP synchronization uses transmission of control frames at the time of three way handshaking based TCP connection establishment phase, as given in table 5.1.

Table 5.1
Control frames of CTHLP synchronization

Frame	Description
SYN	SYN frame transmitted to the receiver for synchronization in connection establishment phase
ACK_SYN	ACK_SYN frame transmitted to the sender for positive acknowledgement respect to SYN frame
NAK_SYN	NAK_SYN frame transmitted to the sender for negative acknowledgement respect to SYN frame
FIN_SYN	FIN_SYN frame transmitted by either party for closing the connection

The SYN frame is used to establish the connection to the other side. It carries index information of different initial parameters. On transmitting the SYN frame, the sender starts a timer and waits for a reply from the receiver. If the receiver does not take any action until a certain time limit and number of attempts exceeded a certain value, the sender restarts the synchronization procedure. When the receiver receives the SYN frame, the it carry out the integrity test. If the messages are received as sent (with no replication, incorporation, alteration, reordering, or replay) the receiver will execute the synchronization check. The sender and receiver have an identical T variable formally store in their respective memory. The sender sends the encrypted T to the receiver. Here the receiver utilizes its $128/192/256$ bits weights to decrypt the encrypted T. If the result is identical to T formerly stored in receiver memory i.e. $(Decrypt_{Receiver\_weight}(Encrypt_{Sender\_weight}(T)) = T$ then the networks are synchronized. This is the best case solution where sender and receiver arbitrarily choose weight vector which are identical. So, networks are synchronized at initial stage. The receiver should send the FIN_SYN frame to alert the sender. But most of the time this best case is not achievable. If decryption algorithm does not produce predictable result, the receiver should use the Chaos synchronized secret seed to generate the input vector (X) which is identical to sender. With this input vector the receiver will work out its $output$ $(\tau^{Receiver})$. If the receiver's and sender's outputs are different, the receiver should not fine-tune its weights and inform the sender its output. The receiver sends the $NACK\_SYN$ frame to notify the sender, with the same ID value. The proposed NAK_SYN frame in this methodology is used for providing the negative acknowledgement in respect to the SYN frame. If receiver's output is equal to sender's output i.e. $(\tau^{Receiver} = \tau^{Sender})$ then receiver update it weights. At the end of weights update, the receiver should report the sender that outputs are equal. The receiver uses the ACK_SYN frame to notify the sender, with the same ID value received from sender. The proposed ACK_SYN frame in this methodology is used for providing the positive acknowledgement in respect to the SYN frame. On receipt of ACK_SYN, the sender also updates its weight. If sender receives ACK_SYN it should update its weights. The sender will create new synchronization frame until receive the FIN_ACK frame from receiver. When the sender receives the frame FIN_ACK, it stops the further synchronization. The proposed FIN_SYN frame in this methodology is used for closing the connection. At end of synchronization, both networks provide the identical weight vector which acts as a session key identical to both end. The figure 5.3 shows the exchange of frames during CTHLP synchronization process.

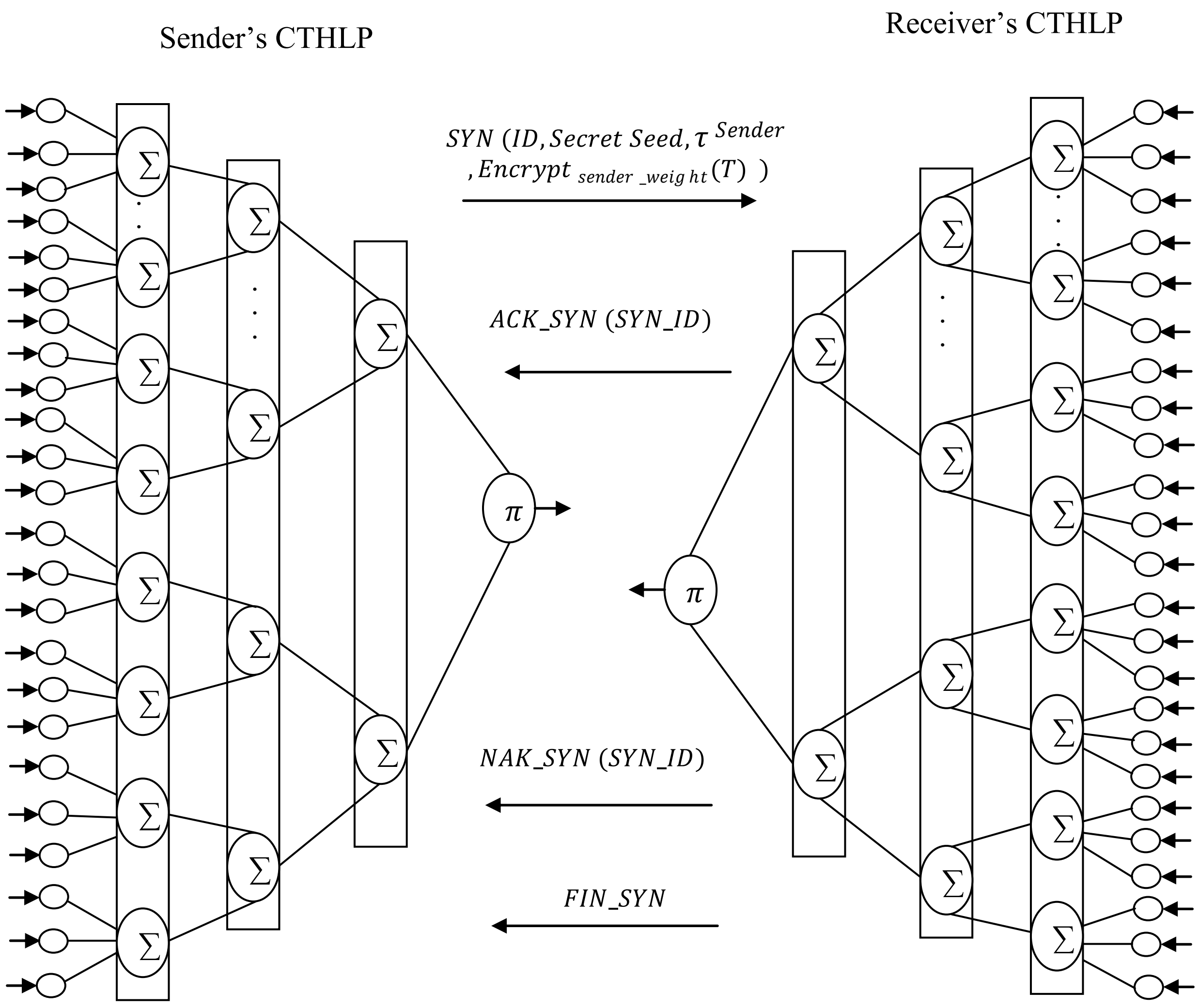


Figure 5.3: Exchange of control frames between sender and receiver during CTHLP synchronization

Table 5.2 shows the different frames and their corresponding command codes.

Table 5.2
CTHLP control frames and their command codes

Frame	**Command**
SYN	0000
FIN_SYN	0001
ACK_SYN	0010
NAK_SYN	0011
$AUTH$	0100
$Reserved$	0101-1111

The identifier (ID) is the function of informing the sender and receiver where the message is a recent message. The variable ID starts with zero and is incremented every time that the sender sends a synchronization frame. The detailed frame format of SYN frame is discussed in section 5.2.1.1.1. The detailed frame format of ACK_SYN frame is given in section 5.2.1.1.2. The frame format of NAK_SYN frame has been discussed in section 5.2.1.1.3. The frame format of FIN_SYN frame is discussed in section 5.2.1.1.4.

5.2.1.1.1 Synchronization (SYN) Frame

During synchronization process sender constructs a SYN frame and transmit to the receiver for handshaking in connection establishment phase. Sender utilizes its initial 128 weights as key for encryption of T variable (formerly stored in its memory) $Encrypt_{Sender\_weight}(T)$. Sender constructs a SYN frame and transmitted to the receiver for handshaking purpose in connection establishment phase. SYN usually comprises of $Command\ Code$, $SYN\ ID$, $Sender\ output(\tau^{Sender})$, $Encrypt_{Sender\_weight}(T)$, authentication bits and CRC. SYN frame has the fixed $Command\ Code$ i.e. 0000. So, $Command\ Code$ needs four bits. $SYN\ ID$, $Sender\ output(\tau^{Sender})$, $Encrypt_{Sender\_weight}(T)$, authentication bits and CRC needs eight bits, one bits, 128 bits, m bits and sixteen bits respectively. When the receiver receive SYN frame, the receiver should carry out integrity test. Receiver performs Integrity test on receiving the SYN frame. If the messages are received as sent (with no replication, incorporation, alteration, reordering, or replay) the receiver will execute the synchronization test. In synchronization test receiver utilize its 128 first weights as key for decryption of $Encrypt_{Sender\_weight}(T)$ that was received from the sender. After decryption operation if $(Decrypt_{Receiver\_weight}(Encrypt_{Sender\_weight}(T)) = T$ then networks are synchronized. Figure 5.3 shows the complete frame format of SYN frame.

$Command\ Code$ 0000	$SYN\ ID$	τ^{Sender}	$Encrypt_{Sender\_weight}(T)$	$Authenticaion$ bits	CRC $(Cyclic\ Redundancy\ Checker)$
4	8	1	128	m	16 $(bits)$

Figure 5.3: Synchronization (SYN) frame

5.2.1.1.2 Synchronization (ACK_SYN) Acknowledgement Frame

ACK_SYN frame send by the receiver to the sender in respect of ACK frame for positive acknowledgement of the parameters value. This proposed frame comprises of $Command\ Code, SYN\ ID, CRC$. $Command\ Code$ needs four bits. The ACK_SYN frame has the fixed $Command\ Code$ i.e. 0010. Eight bits is used for representing the $SYN\ ID$ and CRC needs sixteen bits for error checking purpose. Now check the condition i.e. If $(Decrypt_{Receiver\_weight}(Encrypt_{Sender\_weight}(T)) \neq T$ then receiver use the $Secret\ Seed$ received from sender to produce the receiver input vector (X) identical to sender input vector (X) and calculates the output $\tau^{Receiver}$. If $(\tau^{Receiver} = \tau^{Sender})$ then receiver should update their weights where $\sigma_k^{Sender/Receiver} = \tau^{Sender/Receiver}$ using learning rule. At end of weight updation of the receiver, then it sends ACK_SYN with the same ID to instruct the sender for updating the weights. If sender receives ACK_SYN it should update its weights. Figure 5.4 shows the complete frame format of ACK_SYN frame.

$Command\ Code$ 0010	$SYN\ ID$	CRC $(Cyclic\ Redundancy\ Checker)$
4	8	16 $(bits)$

Figure 5.4: Acknowledgement of Synchronization (ACK_SYN) frame

5.2.1.1.3 Negative Acknowledgement (NAK_SYN) Frame of Synchronization

NAK_SYN frame send by the receiver to the sender in respect of ACK frame for negative acknowledgement of the parameters value. This proposed frame comprises of three fields, $Command\ Code, SYN\ ID, CRC$. $Command\ Code$ needs four bits. The ACK_SYN frame has the fixed $Command\ Code$ i.e. 0011. Eight bits are used for representing the $SYN\ ID$ and CRC needs sixteen bits for error checking purpose. If $(\tau^{Receiver} \neq \tau^{Sender})$ then the receiver sends the message NAK_SYN to notify the sender. If the receiver and sender outputs are different, the receiver should not fine-tune its weights and inform the sender. The receiver sends the message NAK_SYN to notify the sender, with the same ID value. Figure 5.5 shows the complete frame format of NAK_SYN frame.

$Command\ Code$ 0011	SYN_ID	CRC $(Cyclic\ Redundancy\ \ Checker)$
4	8	$16\ (bits)$

Figure 5.5: Negative Acknowledgement of Synchronization (NAK_SYN) frame

5.2.1.1.4 Finish Synchronization (FIN_SYN) Frame

FIN_SYN frame send by the either party for finish the synchronization process. This proposed frame comprises of $Command\ Code, SYN\ ID, CRC$. $Command\ Code$ needs four bits. The FIN_SYN frame has the fixed $Command\ Code$ i.e. 0001. Eight bits is used for representing the $SYN\ ID$ and CRC needs sixteen bits for error checking purpose. If $(Decrypt_{Receiver\ \_weig\ ht}(Encrypt_{Sender\ \_weig\ ht}(T)) = T$ then networks are synchronized. Receiver sends the FIN_SYN frame to the sender. Figure 5.6 shows the complete frame format of FIN_SYN frame.

$Command\ Code$ 0001	$SYN\ ID$	CRC $(Cyclic\ Redundancy\ \ Checker)$
4	8	$16\ (bits)$

Figure 5.6: Finish Synchronization (FIN_SYN) frame

The CTHLP synchronization algorithm for generating synchronized session key is discussed in section 5.2.1.1.5. Section 5.2.1.1.6 presents the computational complexity of the CTHLP synchronization algorithm and CTHLP learning is discussed in section 5.2.1.1.7.

5.2.1.1.5 CTHLP Synchronization

Sender and receiver initially initiate Chaos synchronization between them to construct a common seed value at both sides. The Chaos synchronized identical seed value is used to generate the common input vector for sender and receiver. Two CTHLPs start with identical input vector and anonymous random weight vector. In each time both CTHLPs compute their final output based on input and weight vector, and communicate to each other. If both are be in agreement on the mapping between the present input and the output, their weights are updated according to an appropriate learning rule. In the case of discrete weight values this process leads to full synchronization in a finite number of steps. After synchronization procedure weight vector of both CTHLP's become identical. This indistinguishable weight vector forms the session key for a particular session. Authentication steps also get performed parallel to the synchronization steps.

Input : *Tuning parameters* $(\sigma, b, r, x_1, y_2$ *and* $z_2)$, *random weights*

Output : *Sender's and receiver's synchronized CTHLP along with synchronized session key*

Method : *Sender's and receiver's CTHLPs both are be in agreement on the mapping between the present input and the output, their weights are updated according to an appropriate learning rule. After synchronization procedure weight vector of both CTHLPs become identical. These indistinguishable weight vector forms the session key for a particular session.*

Step 1. *Sender initializes the value of* σ *and* b, *after that value of* b *is send to the receiver.*

Step 2. *Receiver initializes the value of* r.

Step 3. *Sender generates the point* x_1 *and* z_1.

Step 4. *Receiver generates the point* y_2*and* z_2.

Step 5. *Sender sends* x_1 *to receiver and receiver sends* y_2 *and* z_2 *to sender.*

Step 6. *Receiver calculates the new value of* $\dot{y}_2$ *and* $\dot{z}_2$*with the help of* r *and* b *using the equations 5.7 and 5.8 then returns the value of* $\dot{y}_2$ *and* $\dot{z}_2$ *to the sender.*

$$\dot{y}_2 = rx - y_2 - xz_2 \tag{5.7}$$

$$\dot{z}_2 = xy_2 - bz_2 \tag{5.8}$$

Step 7. *Sender calculates the value of* $\dot{x}_1$ *and* $\dot{z}_1$ *with the help of* y_2, σ *and* b *using equations 5.9 and 5.10 then sends the value of* $\dot{x}_1$ *to the receiver and so on.*

$$\dot{x}_1 = \sigma(x_1 - y_2) \tag{5.9}$$

$$\dot{z}_1 = x_1y_2 - bz_1 \tag{5.10}$$

Step 8. *Sender generates a nonce. This nonce gets encrypted using a symmetric cipher with z_1 as the key and sends the results of the encryption using equation 5.11.*

$$En\_Nonce = Encrypt_{z_1}(Nonce) \tag{5.11}$$

Step 9. *The receiver decrypts En_Nonce using z_2 as the key, performs a defined function on it using equation 5.12 and 5.13.*

$$De\_Nonce = Decrypt_{z_2}(En\_Nonce) \tag{5.12}$$

$$Fn\_Nonce = f(De\_Nonce) \tag{5.13}$$

Step 10. *The receiver encrypts the result of the previous step using z_2 as the key and sends the result to the sender illustrated in equation 5.14.*

$$En\_Fn\_Nonce = Encrypt_{z_2}(Fn\_Nonce) \tag{5.14}$$

Step 11. *The sender decrypts this message using z_1 as the key, performs the inverse of the pre-defined function and checks if the original nonce is obtained as shown in equation 5.15.*

$$Nonce = f^{-1}\left(Decrypt_{z_1}(En\_Fn\_Nonce)\right) \tag{5.15}$$

Step 12. *If synchronization is not achieved, the process is repeated from step 5.*

Step 13. *If synchronization is achieved i.e. $z_1 = z_2$ then z_1 is used as a seed for a pseudo random number generator to generate identical input vector(X) at both end.*

Step 14. *Initialization of synaptic links between input layer and first hidden layer and between first hidden layer and second hidden layer using random weights values. Where,$W_{ij} \epsilon \{-L, -L+1, \dots, +L\}$.*

Repeat step 15 to step 24 until the full synchronization is achieved,

Step 15. *The input vector(X) is generated both end using the Chaos synchronized seed value.*

Step 16. *Computes the values of hidden neurons by the weighted sum over the current input values. Each hidden neuron in first Hidden layer produces σ^1_i values. Similarly, each hidden neuron in second hidden layer produces σ^2_i value. Each hidden neuron in hidden layer number*

3 produces $\sigma^3{}_i$ value. These can be represented using equation 5.16, 5.17 and 5.18.

$$\sigma^1{}_i = sgn\left(\sum_{j=1}^{N} W_{i,j} X_{i,j}\right) \tag{5.16}$$

$$\sigma^2{}_i = sgn\left(\sum_{j=1}^{N} \sigma_i^1\right) \tag{5.17}$$

$$\sigma^3{}_i = sgn\left(\sum_{j=1}^{N} \sigma_i^2\right) \tag{5.18}$$

$sgn(x)$ is a function represents in equation 5.19, which returns $-1, 0$ or 1:

$$sgn(x) = \begin{cases} -1 & if\ x < 0 \\ 0 & if\ x = 0 \\ 1 & if\ x > 0 \end{cases} \tag{5.19}$$

If the weighted sum over its inputs is negative then set $\sigma_i = -1$. Hence, set $\sigma_i = +1$, if the weighted sum over its inputs is positive, or else if weighted sum is zero then it is set to, $\sigma_i = 0$.

Step 17. *Compute the value of the final output neuron by computing multiplication of all values produced by $K2$ no. hidden neurons using equation 5.20.*

$$\tau = \prod_{i=1}^{K2} \sigma_i^3 \tag{5.20}$$

Step 18. *Sender utilizes its 128 weights as key for encryption of T variable (formerly stored in its memory) $Encrypt_{Sender\_weight}(T)$.*

Step 19. *Sender constructs a SYN frame and transmitted to the receiver for handshaking purpose in connection establishment phase. SYN usually comprises of the $Command\ Code, ID, Sender\ output\ (\tau^{Sender})$, $Encrypt_{Sender\_weight}(T)$ and CRC (Cyclic Redundancy Checker) and last m bits of the identical input vector. In this way performed authentication step parallel by selecting last m bits of the identical input vector and transmitting towards the other party over public channel using SYN frame.*

Step 20. *Receiver performs Integrity test after receiving the SYN frame. Then receiver perform authentication step to*

```
        Check if (Sender (m bits) = Receiver (m bits)) then
        authentication = TRUE
        Else
        authentication = FALSE
        If authentication is true then receiver utilize its 128 first weights as key for decryption of $Encrypt_{Sender\_weight}(T)$ that was received from the sender.
        If authentication is false then receiver sends ACK_NAK to sender.
Step 21.  If $(Decrypt_{Receiver\_weight}(Encrypt_{Sender\_weight}(T)) = T$ then networks are synchronized. Go to step 25.
Step 22.  If $(Decrypt_{Receiver\_weight}(Encrypt_{Sender\_weight}(T)) \neq$ then receiver use the Chaos based secret seed to produce the receiver input vector$(X)$ identical to sender input vector$(X)$ and calculates the output $\tau^{Receiver}$ using step 16 and step 17
Step 23.  If $(\tau^{Receiver} = \tau^{Sender})$ then performs the following steps.
        Step 23.1   Receiver update their weights where $\sigma_k^{Sender/Receiver} = \tau^{Sender/Receiver}$ using any of the learning rules discussed in chapter 1 section 1.8.
        Step 23.2   At the end of receivers weights update, the receiver sends ACK_SYN to instruct the sender for updating the weights using step 23.1.
        Step 23.3   Sender transmits
                    $Encrypt_{Sender\_updated\_weight}(T)$ to receiver.
        Step 23.4   Receiver checks
    if $(Decrypt_{Receiver\_updated\_weight}(Encrypt_{Sender\_updated\_weight}(T)) = T$
                    then networks are synchronized. Go to step 25.
        Step 23.5   Perform the following checking
    if $(Decrypt_{Receiver\_updated\_weight}(Encrypt_{Sender\_updated\_weight}(T)) \neq T$
                    then networks are still not synchronized. Go to step 23.1.
```

Step 24. *If* $(\tau^{Receiver} \neq \tau^{Sender})$ *then the receiver sends the message NAK_SYN to notify the sender. Go to step 15.*

Step 25. *Finally, the receiver sends the frame* FIN_SYN *to inform the sender to finish the synchronization phase.*

5.2.1.1.6 Complexity Analysis

In CTHLP synchronization algorithm sender initialization of the value of σ and b takes needs unit amount of computation. Receiver initialization of the value of r also takes unit amount of computation. Generation of the point x_1 and z_1 takes unit amount of computation. Generation of the point y_2 and z_2 takes unit amount of computation. Receiver calculates the new value of y_2 and z_2 with the help of r and b. This step also takes unit amount of computation. Sender calculates the value of x_1 and z_1 with the help of y_2, σ and b. This step also takes unit amount of computation. Sender generates a nonce having a random value. This nonce is encrypted using a symmetric cipher with z_1 as the key and sends the results of the encryption. This step needs $(nonce\ length)$ amount of computation. The receiver decrypts En_Nonce using z_2 as the key. It also takes $(nonce\ length)$ amount of computation. The receiver encrypts the result of the previous step using z_2 as the key. It takes $(message\ length)$ amount of computation. The sender decrypts this message using z_1 as the key, performs the inverse of the pre-defined function and checks if the original nonce is or not. It takes $(encrypted\ message\ length)$ amount of computation. Initialization of weight vector takes $(N \times K1 + K1 \times K2 + K2 \times K3)$ amount of computations. For example, if $N = 2, K1 = 2, K2 = 3, K3 = 2$ then total numbers of synaptic links (weights) are $(2 \times 2 + 2 \times 3 + 3 \times 2) = 16$. So, it takes sixteen amount of computations. Generation of N number of input vector for each $K1$ number of hidden neurons takes $(N \times K1)$ amount of computations. Computation of the hidden neuron outputs takes $(K1 + K2 + K3)$ amount of computations. Where $K1$,$K2$ and $K3$ are the number of hidden units in 1st, 2nd and 3rd layer respectively. *C*omputation of final output value takes unit amount of computation because it needs only a single operation to compute the value. Encryption of T using *Exclusive-OR* operation also takes unit amount of computations. Decryption of T using *Exclusive-OR* operation also takes unit amount of computations. Checking if

$(Decrypt_{Receiver\_weight}(Encrypt_{Sender\_weight}(T)) = T$ or not takes unit amount of computation. In CTHLP the weight updating procedure takes place where $\sigma_k^{Sender/Receiver} = \tau^{Sender/Receiver}$ using any of the learning rules which takes $\left(no.of\ \sigma_k^{Sender/Receiver} = \tau^{Sender/Receiver}\right)$ amount of computations.

In best case of CTHLP synchronization algorithm, sender's and receiver's arbitrarily chosen weight vectors are identical. So, networks are synchronized at initial stage do not needs to update the weight using learning rule. Here, $(nonce\ length + message\ length + encrypted\ message\ length + (N \times K1) + (N \times K1 + K1 \times K2 + K2 \times K3) + (K1 + K2 + K3))$ amount of computation is needed in best case which is in form of $O(\text{Generation of common seed value} + \text{initialization of input vector} + \text{initialization of weight vector} + \text{Computation of the hidden neuron outputs})$.

If the sender's and receiver's arbitrarily chosen weight vector are not identical then in each iteration the weight vectors of the hidden unit which has a value equivalent to the pereceptron output are updated according to the learning rule. This scenario leads to average and worst case situation where I number of iteration to be performed to generate the identical weight vectors at both ends. So, the total computation for the average and worst case is $\left(nonce\ length + message\ length + encrypted\ message\ length + (N \times K1) + (N \times K1 + K1 \times K2 + K2 \times K3) + (K1 + K2 + K3)\right) + \left(I \times (no.of\ \sigma_k^{Sender/Receiver} = \tau^{Sender/Receiver})\right)$ which is can be expressed in $\mathrm{O}(\text{Time complexity in first iteration} + (\text{No. of iteration} \times \text{No. of weight updation}))$.

5.2.1.1.7 CTHLP Learning Mechanism

In learning mechanism if the output bits are different for sender (A) and receiver (B) i.e. $\tau^A \neq \tau^B$, nothing get changed. If $\tau^A = \tau^B = \tau$, only the weights of the hidden units with $\sigma_k^{\frac{A}{B}} = \tau^{\frac{A}{B}}$ will be updated. The weight vector of this hidden unit is adjusted using any of the learning rules discussed in chapter 1 section 1.8. For small L values Hebbian takes less synchronization steps than other two learning rules in the range of $2-2-3-2-5$ $(N-K1-K2-K3-L)$ to $2-2-3-2-15$ but as the L value increases Hebbian rule takes more steps to synchronize than other two learning rules. Here, Anti-Hebbian rules take fewer steps than the other two learning rules in the range of $2-2-3-2-8-20$ to $2-2-3-2-30$. Random walk outperform from $3-2-2-8-35$ and beyond that. The most vital findings is that if the synaptic depth i.e. weight range (L) is increased, the complexity of a successful attack grows exponentially, but there is only a polynomial increase of the effort needed to generate a key. So, increasing the L value security of the system can be increased.

5.2.1.2 Ant Colony Intelligence (ACI) based Fittest Keystream Generation

In this section Ant Colony Intelligence (ACI) based keystream generation technique for message encryption/decryption has been presented to illustrate the complete cryptographic technique. Instead of this technique any other light weight encryption/decryption technique also may use for exchanging message between sender and receiver.

In the Ant Colony Intelligence (ACI) based approach an ant agent is used to denote a keystream (collection of alphanumeric characters). Each Ant can have multiple dimensions. Each dimension denotes an individual key within that keystream. The dimensions in the keystream can be filled or unfilled. For example if the ceiling of dimension of each Ant is equal to 192 then it is represented by equation 5.21.

$$Ant_i \ or \ Keystream_i = (Key_1, Key_2, \dots, Key_{192}) \tag{5.21}$$

This signifies a keystream comprises of 192 keys i.e. 192 alphanumeric characters. Keystream length can be obtained by counting number of dimensions are filled in the keystream. Generally keystream length is less than or equal to the plaintext. With 192 alphanumeric characters multiple keystream can be generated of predetermined fixed length by permutation of these predetermined fixed length characters ordering all feasible ways without any reappearance. So, for example if total number of alphanumeric characters = 192 and keystream length = 128 then among 192 alphanumeric characters 128 alphanumeric characters are elected such a way so that by ordering all possible ways without any duplication these 128 characters forms multiple keystream having fixed length i.e. 128. For an example if five characters $\mathrm{A, C, M, H, R}$ are taken to form keystream of length four among 192 alphanumeric characters. Then there are 120 possible ways of obtaining keystream which are as follows.

ACMHR, CAMHR, MACHR, HACMR, RACMH

ACMRH, CAMRH, MACRH, HACRM, RACHM

ACHMR, CAHMR, MAHCR, HAMCR, RAMCH

ACHRM, CAHRM, MAHRC, HAMRC, RAMHC

ACRMH, CARMH, MARCH, HARCM, RAHCM

ACRHM, CARHM, MARHC, HARMC, RAHMC

AMCHR, CMAHR, MCAHR, HCAMR, RCAMH

AMCRH, CMARH, MCARH, HCARM, RCAHM

AMHCR, CMHAR, MCHAR, HCMAR, RCMAH

AMHRC, CMHRA, MCHRA, HCMRA, RCMHA

AMRCH, CMRAH, MCRAH, HCRAM, RCHAM

AMRHC, CMRHA, MCRHA, HCRMA, RCHMA

AHCMR, CHMRA, MHACR, HMACR, RMACH

AHCRM, CHMAR, MHARC, HMARC, RMAHC

AHMCR, CHRMA, MHCAR, HMCAR, RMCAH

AHMRC, CHRAM, MHCRA, HMCRA, RMCHA

AHRCM, CHAMR, MHRAC, HMRAC, RMHAC

AHRMC, CHARM, MHRCA, HMRCA, RMHCA

ARCMH, CRAMH, MRACH, HRACM, RHACM

ARCHM, CRAHM, MRAHC, HRAMC, RHAMC

ARMCH, CRMAH, MRCAH, HRCAM, RHCAM

ARMHC, CRMHA, MRCHA, HRCMA, RHCMA

ARHCM, CRHAM, MRHAC, HRMAC, RHMAC

ARHMC, CRHMA, MRHCA, HRMCA, RHMCA

Using 192 characters total number of generated possible keystream is given in equation 5.22.

$$\sum_{c=1}^{192} \frac{192!}{(192-c)!} \approx 192!\,(e) \approx 192! \times 2.718 \qquad (5.22)$$

According to Ant Colony Intelligence technique each ant should have an allied energy. The ACI technique also offers energy for each and every ant or keystream. The energy value of the ant agent is computed by taking the number of characters in the keystream occurring in the plaintext divided by the keystream length. The pheromone deposition of the ant agent with a maximum energy value greater than a specified threshold value is the solution and the keystream is chosen for encryption i.e. $if\ (Energy(Ant_i) > threshold\ value)\ then\ return\ (Ant_i\ with\ Energy(Ant_i) = max\ energy\ value)$. Energy value for each ant agent is computed by the equation 5.23.

$$Energy\ (Ant_i) = (count\ (key_{\ j\ \in}\ plain\ text))/(lengthof(Ant_i)\ ) \qquad (5.23)$$

$$where\ j = 1, 2, \dots, lengthof(Ant_i)$$

In the pheromone updating phase of Ant agent $While\ (Energy(Ant_i) < threshold\ value)$ change the keystream i.e. update the pheromone deposition of Ant_i agent until $Energy(Ant_i in\ a\ trial) > threshold\ value)$. For update the pheromone deposition at first energy value for each ant agent in the current trial is evaluated. Next select the ant agent where, $Energy(Ant_i in\ current\ trial) > threshold\ value)$ and return Ant_i in current trial with $Energy(Ant_i) = max\ energy$ value in current trial.

In ACI based keystream generation technique following parameters are used

- Maximum length of ACI based keystream i.e. maximum number of character represents a keystream is $L = 192$. N is the number of characters to represents keystream. Maximum value of N is L i.e. 192.
- A predefined threshold value for describing energy factor of Ant agent. This scheme used 0.65 as a threshold value.
- A predetermined value to generate the keys for the characters in the plaintext which is at a position greater than the length of the keystream. The technique uses equation 5.24 to compute the predetermined value.

$$Predetermined\_value = \ lengthof(plaintext)/2 \qquad (5.24)$$

The figure 5.8 shows the flowchart of ACI based keystream generation and section 5.2.1.2.1 presents the complete encryption/decryption keystream generation algorithm.

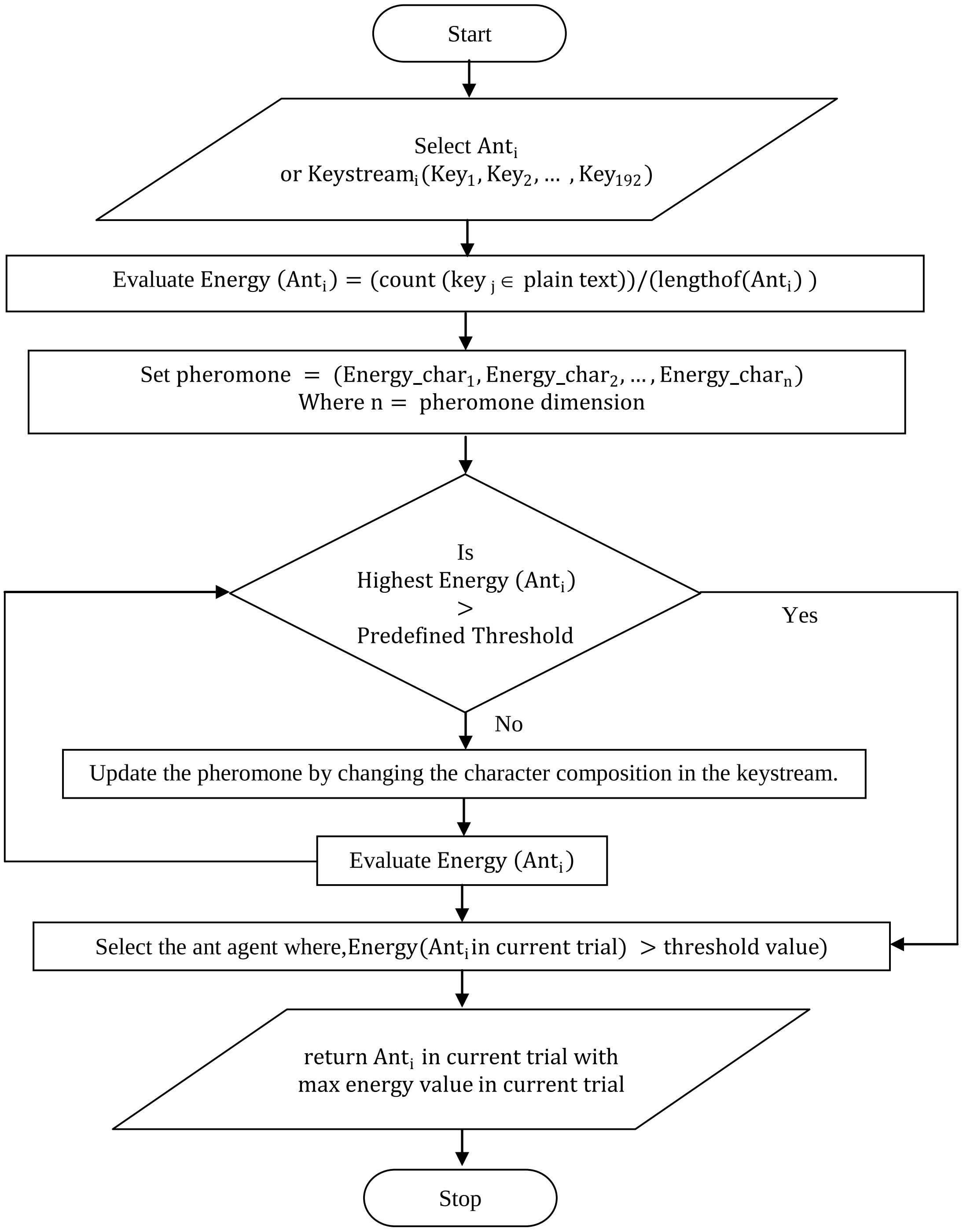

Start
Select Ant_i or $Keystream_i(Key_1, Key_2, \dots, Key_{192})$
Evaluate Energy $(Ant_i) = (count\ (key_j \in plain\ text))/(lengthof(Ant_i))$
Set pheromone $= (Energy\_char_1, Energy\_char_2, \dots, Energy\_char_n)$
Where n = pheromone dimension
Is Highest Energy (Ant_i) > Predefined Threshold
Yes
No
Update the pheromone by changing the character composition in the keystream.
Evaluate Energy (Ant_i)
Select the ant agent where, Energy(Ant_i in current trial) > threshold value)
return Ant_i in current trial with max energy value in current trial
Stop

Figure 5.8: Flow chart of Ant Colony Intelligence (ACI) based fittest keystream generation

5.2.1.2.1 Ant Colony Intelligence (ACI) based Fittest Keystream Generation Algorithm

ACI based encryption/decryption keystream generation algorithm a threshold value is selected to weigh against energy level of each ant agent. Ant agent having highest energy level more than predefined threshold value is selected as a keystream.

Input : Ant agent with Pheromone

Output : ACI based keystream

Method : A threshold value is selected to weigh against energy level of each ant agent. Ant agent having highest energy level more than predefined threshold value is selected as a keystream.

```
Step 1.    Set length of a keystream as L. Choose arbitrary N characters to
           representing pheromone deposition.
Step 2.    Perform possible permutation by choosing arbitarily N characters to
           represents keystream denoting the pheromone deposition of length L.
Step 3.    Evaluate energy value of each ant agent $Ant_i$ according to the
           following equation 5.25.
           $Energy\ (Ant_i) = (count\ (key_j \in plain\ text))/(lengthof\,(Ant_i))$   (5.25)
Step 4.    Check, $if\ (Energy\ (Ant_i) > Predefined\ Threshold)$ then chooses
           keystream for encryption having maximum energy value grater than
           threshold.
Step 5.    Else repeat the following steps until
           $(Highest\ energy\ value\ in\ a\ trial > Predefined\ Threshold)$
    Step 5. 1.  Update the pheromone by changing the character
                composition in the keystream.
    Step 5. 2.  Compute the energy value for updated pheromone
                deposition.
    Step 5. 3.  Select the ant agent,
                $if\ (Energy\ (Ant_i) > Predefined\ Threshold)$
                then return ant agent having maximum energy value.
```

Step 6. *If the length of the text is greater than the length of the keystream then the values of the keystream are added to a predetermined value to generate the keys for the characters in the text which is at a position greater than the length of the keystream.*

The ACI based fittest keystream is used to perform the encryption operation on the plaintext. The detail step of ACI based encryption process is given in section 5.2.1.3.

5.2.1.3 Encryption Algorithm

Input : *Source file/source stream i.e. plaintext*

Output : *Encrypted file/encrypted stream i.e. cipher text*

Method : *The process operates on binary stream and generates encrypted bit stream through Ant Colony Intelligence (ACI) based encryption.*

Step 1. *If the length of the plaintext is grater than the length of the ACI based keystream then the values of the keystream are added to a predetermined value to generate the keys for the characters in the plaintext which is at a position grater than the length of the key stream. Predetermined value is calculated using the equation 5.26.*

$$Predetermined\_value = lengthof(plaintext)/2 \qquad (5.26)$$

Step 2. *For the very first plaintext block keys are form by the values of the characters in the ACI based keystream.*

Step 3. *For the successive plaintext blocks ACI based keys are generated by adding predetermined value with the keys of the previous block given in equation 5.27 for reducing the key storage load that in turn reduces the space complexity.*

$$Keyforblock(i) = Keyforblock(i-1) + \ Predetermined\,value, \\ where\ i >= 2 \qquad (5.27)$$

Step 4. *Perform Exclusive-OR operation between plaintext block with key in the ACI based keystream.*

Step 5. *Considers the outcomes of step 4 as a stream of finite number of bits N, and is divided into a finite number of blocks, each also containing a finite number of bits n, where $1 \leq n \leq N$. Consider the block $C = c_0^j\, c_1^j\, c_2^j\, c_3^j\, c_4^j\, \dots\, c_{n-1}^j$ of size n in the outcomes of step 4.*

Step 6. *Perform cycle formation techniques on $C = c_0^j\, c_1^j\, c_2^j\, c_3^j\, c_4^j\, \dots\, c_{n-1}^j$ of block of size n. In the following cases $\oplus$ is used to represents the Exclusive-OR operation. Perform the operations given in equation 5.34 to 5.37 for generating the intermediate block*

$I_j = c_0^{j+1} c_1^{j+1}\, c_2^{j+1}\, c_3^{j+1} c_4^{j+1}\, \dots\, c_{n-1}^{j+1}$ from C in the following way:

$$c_{n-1}^{j+1} = c_{n-1}^{j} \quad (5.28)$$

$$c_{n-2}^{j+1} = c_{n-2}^{j} \oplus c_{n-1}^{j+1} \quad (5.29)$$

$$c_1^{j+1} = c_1^{j} \oplus c_2^{j+1} \quad (5.30)$$

$$c_0^{j+1} = c_0^{j} \oplus c_1^{j+1} \quad (5.31)$$

The process continues for a finite number of iterations, which depends on the value of n, the source block C is regenerated. If the number of iterations required regenerating the source block is assumed to be I, then any of the intermediate block is considered as a encrypted block.

5.2.1.4 Session Key based Encryption

During final step of the technique a cascaded *Exclusive-OR* operation between CTHLP synchronized session key and ACI encrypted cipher text is performed to generate final encoded cipher text.

The decryption algorithm takes the cipher text as a binary stream of bits and perform first level of operation using CTHLP generated synchronized session key to produce intermediate decrypted text. Finally, ACI generated fittest keystream based decryption is performed on the intermediate decrypted text to regenerate the plaintext. The algorithm for the complete process is given in section 5.2.2.

5.2.2 CTHLPSCT Algorithm at Receiver

Input : Encrypted file/encrypted stream i.e. cipher text

Output : Source file/source stream i.e. plaintext

Method : The process operates on encrypted binary stream and generates decrypted bit stream through Chaos based CTHLP guided Genetic algorithm (ACI) based decryption operations.

Step 1. Perform cascaded Exclusive-OR operation between CTHLP based session key and cipher text.

Step 2. Perform Ant Colony Intellogence (ACI) based decryption on the outcomes of the step 1 to regenerate starting combination i.e. plaintext.

Step 1 of the algorithm is discussed in section 5.2.2.1. Step 2 of the algorithm for performing Genetic Algorithm based decryption is discussed in section 5.2.2.2.

5.2.2.1 Session Key based Decryption

Initially cascaded *Exclusive-OR* operation between CTHLP synchronized session key and cipher text is performed to produce session key decrypted text. Outcomes of this operation used as an input of ACI based decryption algorithm discussed in 5.2.2.2 to regenerate the plaintext.

In the decryption process the ACI based cipher text is divided into blocks. Exclusive-OR guided cycle formation based decryption is performed on each block. After that all blocks are merged together. The ACI generated keystream is use to *Exclusive-OR* with the merged blocks to regenerate the plaintext. The detail step of ACI based decryption process is given in section 5.2.2.2.

5.2.2.2 Decryption Algorithm

Input : ACI Encrypted file/ ACI encrypted stream

Output : Source file/source stream i.e. plaintext

Method : The process operates on ACI encrypted bit stream and regenerates the plaintext through ACI based decryption.

Step 1. *Divide the ACI encrypted text into different blocks.*

Step 2. *Perform operation given in equation 5.42 to 5.45 upto* $(P - i)$ *steps on each block* $T = t_0^i\, t_1^i\, t_2^i\, t_3^i\, t_4^i \ldots t_{n-1}^i$ *if the total number of iterations required to complete the cycle is* P *and the* i^{th} *step is considered to be the encrypted block.*

$$t_{n-1}^i = t_{n-1}^{i-1} \tag{5.32}$$

$$t_{n-2}^i = t_{n-2}^i \oplus t_{n-1}^i \tag{5.33}$$

$$t_1^i = t_1^{i-1} \oplus t_2^i \tag{5.34}$$

$$t_0^i = t_0^{i-1} \oplus t_1^i \tag{5.35}$$

Step 3. *Merge outcomes of step 2.*

Step 4. *Compute the predetermined value.*

Step 5. *Using predetermined value and keys in the ACI based keystream receiver generates the keys for the portion of the text exceeding the length of the ACI based keystream.*

Step 6. *Generate plaintext by performing Exclusive-OR operation between outcomes of step 3 and ACI based keystream.*

5.3 Implementation

Consider the plaintext to be encrypted is "**antcolonyintelligence**" threshold value is assumed to be 0.65. Each ant agent has a pheromone deposition comprising of characters representing the keystream. The energy level of the ant agent is a count of the characters in the keystream occurring in the plaintext divided by the length of the keystream. The ant agent with a maximum energy level greater than the specified threshold value is chosen as the key stream for text encryption. Table 5.3 show the pheromone deposition of ant agents denoting the keystream and their corresponding energy value. Since the second ant agent in second iteration has the maximum energy value 0.66 which is greater than the threshold value, the keystream "**ueigunscaoblyt**" corresponding to that ant agent is chosen for encryption. Each character in the keystream is chosen as the key for encryption. Since the keystream is smaller than the length of the plaintext to be encoded, the values of the keys of the keystream are added to a predetermined value to generate the keys for the remaining portion of the plaintext. The predetermined value can be generated by dividing the length of the plaintext by half of its length. Here the value is chosen as 15. Thus the keys for the portion of the plaintext exceeding the length of the keystream is generated by adding the values of the keys in the keystream with the value 15.

Table 5.3
ACI based keystream generation

Iteration 1	**Energy**	**Iteration 2**	**Energy**
ckyaptseifdorgq	0.46	cyusadkleownjgm	0.53
anwghqbcletzduo	0.53	ueigunscaoblyt	0.66
yurtdfbnczfsvam	0.33	tedcbkhouesxvaq	0.40
rqewcalkygtxifo	0.60	ivbjtwaxrdgnzpu	0.33
Highest energy	$\mathbf{0.60}$	**Highest energy**	$\mathbf{0.66}$

Binary representations of ASCII value of the plaintext is
01100001/01101110/01110100/01100011/01101111/01101100/01101111/01101110/01111001/01101001/01101110/01110100/01100101/01101100/01101100/01101001/01100111/01100101/01101110/01100011/01100101

The ACI based keystream "**ueigunscaoblyt**" has 14 characters. The plaintext "**antcolonyintelligence**" has 21 characters. So, for the extra seven characters ACI based keys are generated by adding predetermined value with the keys of the previous block for reducing the key storage load that in turn reduces the space complexity. Predetermined value gets calculated by the equation 5.36.

$$Predetermined\_value = lengthof(plaintext)/2 \qquad (5.36)$$

So, the predetermined value will be $\left(\frac{21}{2}\right) = 10$

So, binary representation of ASCII value of the ACI based keystream is
01110101/01100101/01101001/01100111/01110101/01101110/01110011/01100011/01100001/01101111/01100010/01101100/01111001/01110100/01111111/01101111/01110011/01110001/01111111/01111000/01111101

On performing ACI keystream based encryption operation the new intermediate encoded text is

00010100/00001011/00011101/00000100/00011010/00000010/00011100/00001101/
00011000/00000110/00001100/00011000/00011100/00011000/00010011/00000110/
00010100/00010100/00010001/00011011/00011000

Binary representations of ASCII value of the ACI encrypted text are divided into variable size segments. Following are the different segments constructed from S.

S_1 = 0001010000001011 (16 bits)

S_2 = 0001110100000100 (16 bits)

S_3 = 0001101000000010 (16 bits)

S_4 = 0001110000001101 (16 bits)

S_5 = 0001100000000110 (16 bits)

S_6 = 0000110000011000 (16 bits)

S_7 = 0001110000011000 (16 bits)

S_8 = 0001001100000110 (16 bits)

S_9 = 00010100 (8 bits)

S_{10} = 0001010000010001 (16 bits)

S_{11} = 00011011 (8 bits)

S_{12} = 00011000 (8 bits)

Cycle formation operation is now performed on S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_{10}, S_{11}, S_{12} segments respectively. For each of the segments, an arbitrary intermediate stream segment is considered as the encrypted stream segment.

The formation of cycles for segments S_1 (0001010000001011) is shown below. After 16 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (0110001100100111) after iteration-6 considered as an encrypted segment for the segment S_1.

0001010000001011→1111001111111001^{1}→0101000101010111^{2}→0011000011001101^{3}→
1110111110111011^{4}→1010010101101001^{5}→0110001100100111^{6}→0010000100011101^{7}→
0001111100001011^{8}→0000101011111001^{9}→0000011001010111^{10}→1111110111001101^{11}
→0101010010111011^{12}→1100110001101001^{13}→0100010000100111^{14}→

0011110000011101^{15}→0001010000001011^{16}

The formation of cycles for segments S_2 (0001110100000100) is shown below. After 16 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (1111100001010100) after iteration-10 considered as an encrypted segment for the segment S_2.
0001110100000100→1111010011111100^{1}→1010110001010100^{2}→1001101111001100^{3}→
1000100101000100^{4}→1000011100111100^{5}→0111110100010100^{6}→0010101100001100^{7}→
0001100100000100^{8}→0000100011111100^{9}→1111100001010100^{10}→0101011111001100^{11}
→1100110101000100^{12}→1011101100111100^{13}→0110100100010100^{14}→
0010011100001100^{15}→0001110100000100^{16}

The formation of cycles for segments S_3 (0001101000000010) is shown below. After 16 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (1010100001100110) after iteration-3 considered as an encrypted segment for the segment S_3.
0001101000000010→0000100111111110^{1}→1111100010101010^{2}→1010100001100110^{3}→
1001100000100010^{4}→1000100000011110^{5}→0111100000001010^{6}→0010100000000110^{7}→
0001100000000010^{8}→1111011111111110^{9}→0101001010101010^{10}→1100111001100110^{11}
→1011101000100010^{12}→1001011000011110^{13}→0111001000001010^{14}→
0010111000000110^{15}→0001101000000010^{16}

The formation of cycles for segments S_4 (0001110000001101) is shown below. After 16 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (0001000100001101) after iteration-8 considered as an encrypted segment for the segment S_4.
0001110000001101→0000101111111011^{1}→0000011010101001^{2}→0000001001100111^{3}→
0000000111011101^{4}→1111111101001011^{5}→0101010100111001^{6}→0011001100010111^{7}→
0001000100001101^{8}→1111000011111011^{9}→1010111110101001^{10}→0110010101100111^{11}
→1101110011011101^{12}→1011010001001011^{13}→0110110000111001^{14}→
0010010000010111^{15}→0001110000001101^{16}

The formation of cycles for segments S_5 (0001100000000110) is shown below. After 16 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (1111111001100110) after iteration-4 considered as an encrypted segment for the segment S_5.

0001100000000110→0000100000000010^{1}→0000011111111110^{2}→0000001010101010^{3}→
1111111001100110^{4}→1010101000100010^{5}→0110011000011110^{6}→0010001000001010^{7}→
0001111000000110^{8}→0000101000000010^{9}→1111100111111110^{10}→1010100010101010^{11}
→1001100001100110^{12}→1000100000100010^{13}→0111100000011110^{14}→
0010100000001010^{15}→0001100000000110^{16}

The formation of cycles for segments S_6 (0000110000011000) is shown below. After 8 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (1100110010001000) after iteration-5 considered as an encrypted segment for the segment S_6.

0000110000011000→0000010000001000^{1}→0000001111111000^{2}→1111111010101000^{3}→
0101010110011000^{4}→1100110010001000^{5}→0100010001111000^{6}→0011110000101000^{7}→
0001010000011000^{8}→0000110000001000^{9}→1111101111111000^{10}→0101011010101000^{11}
→1100110110011000^{12}→0100010010001000^{13}→0011110001111000^{14}→
0001010000101000^{15}→0000110000011000^{16}

The formation of cycles for segments S_7 (0001110000011000) is shown below. After 8 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (0101001111111000) after iteration-2 considered as an encrypted segment for the segment S_7.

0001110000011000→1111010000001000^{1}→0101001111111000^{2}→1100111010101000^{3}→
0100010110011000^{4}→0011110010001000^{5}→0001010001111000^{6}→0000110000101000^{7}→
0000010000011000^{8}→1111110000001000^{9}→1010101111111000^{10}→0110011010101000^{11}
→1101110110011000^{12}→1011010010001000^{13}→0110110001111000^{14}→
0010010000101000^{15}→0001110000011000^{16}

The formation of cycles for segments S_8 (0001001100000110) is shown below. After 16 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (1111001100000010) after iteration-9 considered as an encrypted segment for the segment S_8.

$0001001100000110 \rightarrow 1111000100000010^1 \rightarrow 0101000011111110^2 \rightarrow 1100111110101010^3 \rightarrow 0100010101100110^4 \rightarrow 1100001100100010^5 \rightarrow 0100000100011110^6 \rightarrow 0011111100001010^7 \rightarrow 0001010100000110^8 \rightarrow 1111001100000010^9 \rightarrow 1010111011111110^{10} \rightarrow 0110010110101010^{11} \rightarrow 0010001101100110^{12} \rightarrow 1110000100100010^{13} \rightarrow 0101111100011110^{14} \rightarrow 0011010100001010^{15} \rightarrow 0001001100000110^{16}$

The formation of cycles for segments S_9 (00010100) is shown below. After 8 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (11001100) after iteration-5 considered as an encrypted segment for the segment S_9.

$00010100 \rightarrow 00001100^1 \rightarrow 00000100^2 \rightarrow 11111100^3 \rightarrow 01010100^4 \rightarrow 11001100^5 \rightarrow 01000100^6 \rightarrow 00111100^7 \rightarrow 00010100^8$

The formation of cycles for segments S_{10} (0001010000010001) is shown below. After 8 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (0000010000000101) after iteration-2 considered as an encrypted segment for the segment S_{10}.

$0001010000010001 \rightarrow 0000110000001111^1 \rightarrow 0000010000000101^2 \rightarrow 1111110000000011^3 \rightarrow 0101010000000001^4 \rightarrow 0011001111111111^5 \rightarrow 0001000101010101^6 \rightarrow 0000111100110011^7 \rightarrow 0000010100010001^8 \rightarrow 0000001100001111^9 \rightarrow 0000000100000101^{10} \rightarrow 1111111100000011^{11} \rightarrow 0101010100000001^{12} \rightarrow 1100110011111111^{13} \rightarrow 0100010001010101^{14} \rightarrow 0011110000110011^{15} \rightarrow 0001010000010001^{16}$

The formation of cycles for segments S_{11} (00011011) is shown below. After 16 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (10011001) after iteration-5 considered as an encrypted segment for the segment S_{11}.

$00011011 \rightarrow 00001001^1 \rightarrow 00000111^2 \rightarrow 11111101^3 \rightarrow 10101011^4 \rightarrow 10011001^5 \rightarrow 01110111^6 \rightarrow 00101101^7 \rightarrow 00011011^8$

The formation of cycles for segments S_{12} (00011000) is shown below. After 16 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (10011000) after iteration-4 considered as an encrypted segment for the segment S_{12}.

$00011000 \rightarrow 00001000^1 \rightarrow 11111000^2 \rightarrow 10101000^3 \rightarrow 10011000^4 \rightarrow 10001000^5 \rightarrow 01111000^6 \rightarrow$
$00101000^7 \rightarrow 00011000^8$

On completion of the cycle formation technique on each segment twelve intermediate segments are considered as the encrypted segments. After merging the above twelve encrypted segments following ACI based encrypted text is generated.

01100011/00100111/11111000/01010100/10101000/01100110/00010001/00001101/111111
10/01100110/11001100/10001000/01010011/11111000/11110011/00000010/11001100/000
00100/00000101/10011001/10011000

For example CTHLP based following session key is generated

10100101/01101110/11101000/00101011/11100000/00100011/01000100/11001000/100110
01/00010000/11110010/11010101/100100110/00010100/11101010/00101111/00101000/00
101010/10111111/1010111/01101110

Following is the session key encrypted final cipher text produce on performing *Exclusive-OR* operation between ACI based encrypted text and CTHLP based session key.

11000110/01001001/00010000/01111111/01001000/01000101/01010101/11000101/011001
11/01110110/00111110/01011101/11000000/11110010/10000110/00010101/01011000/000
10001/01011010/01001110/11110110

5.4 Security Analysis

In CTHLPSCT, identical input vector for both the parties kept secret for security reason. Attackers has no idea about the internal state of both the machines at a particular instant of time and this is achievable by keeping secret the common input vector. At the time of key exchange procedure key authentication technique is also performed parallel by selecting last m bits of the identical input vector and transmitting directly as an output bit towards the other party over public channel. Receiving party checks these last m bits to its last m bits of identical input vector. If both the sequences are same then both are authenticated otherwise not. Attacker does not have identical input vector like sender and receiver. By sniffing the public channel attacker can gets some bits but from them attacker will not be able to understand which one is output bit of the machine and which one is one of the bits of m bits sequence of the identical input vector. Even if attacker hacks the m bits then for getting the rest of the $(d - m)$ bits of the identical input vector attacker has to perform checking with all $(d - m)$ combination that is computationally infeasible. Where d is the total number of bits in the identical input vector. The security aspects of the algorithm are discussed based on the attack model. It is assumed that the detail of encryption or decryption algorithm is known to the cryptanalyst. The following standard attacks are considered to ensure the robustness of the CTHLPSCT.

- *Cipher text only Attack:* The technique nullifies the success rate of this attack by producing a robust Chaos based Group session key and ACI based encrypted cipher text. The strength of resisting exhaustive key search attack relies on a large key space. The cryptanalyst has only the cipher text to work with. In this ACI technique the key is changed for each character of the plaintext to produce a cipher text that is mathematically difficult to break. Since 192 characters are taken and a permutation of these characters is done to get groups of characters of all possible orderings without any repetition forming the keystream, the total number of keystreams will be $192! \times 2.718$. Thus the possible number of combinations to be searched is $192! \times 2.718$. Thus a hacker has to try all such keystreams to find an appropriate one. This method makes it difficult for the hacker to find out the keystream used for encryption. Thus the size of the key space is $192! \times 2.718$. The technique helps to generate long period of random keystreams along with no obvious relationship between the individual bits of the sequence. Also the generated keystreams are of large linear complex. Finally keystream have high degrees of correlation immunity. Thus it is practically difficult to perform a brute-force search in a key-space.
- *Known Plaintext Attack:* The plaintext is encoded using the cycle formation technique. This would increase the security in such a manner that it is difficult to know the values assigned for the characters in the plaintext. This is because there are 2^l possible combination and the hacker has to search those combinations for the values. Here,

l denotes the length of block. Also the keys used for encryption has to be found by the cryptanalyst. The technique offers better floating frequency of characters. So, known plaintext attack is difficult in this technique.

- *Chosen Plaintext Attack:* The objective of this attack is to find the secret key. This attack is difficult because there is no obvious relationship between the individual bits of the sequence in plaintext and cipher text. In the technique the cipher text is obtained by performing an *Exclusive-OR* operation between the encoded plaintext and the characters in the key stream. This technique is not vulnerable to chosen-plaintext attack, since the plaintext is encoded first using cycle generation technique then outcomes of this get *Exclusive-OR* with ACI based keystream and the outcomes of this is *Exclusive-OR* with the session key. It is difficult for the hacker to find the key chosen for encryption. So, it is difficult to choose a plaintext of his/her choice and get the corresponding cipher text. The technique passes the frequency (monobit) test, runs test, binary matrix rank test and in each session a fresh CTHLP based session key is used for encryption which confirms that chosen plaintext attack is very difficult in this technique.
- *Chosen Cipher text Only Attack:* The technique has a good Chi-Square value this confirms good degree of non-homogeneity and also it passes the discrete Fourier transform test, approximate entropy test, overlapping (periodic) template matching test which confirms that chosen plaintext attack is difficult in this technique. So, it will be difficult get plaintext from the cipher text.
- *Brute Force Attack:* The ACI based key is changed for each character of the plaintext to produce a cipher text that is mathematically impossible to break. Since 192 characters are chosen the total number of keystreams will be $192! \times 2.718$. Thus a hacker has to try all such keystreams to find an appropriate one. This method makes it difficult for the hacker to find out the keystream used for encryption. Encryption is an important issue in wireless communication since it is carried out over the air interface, and is more vulnerable to fraud and eavesdropping. Also the keystream is used to generate the keys for the portion of the plaintext exceeding the length of the keystream. This method of encryption reduces the number of keys to be stored and distributed. Due to high complexity brute force attack will not be feasible. The technique has a good entropy

value near to eight which indicates that brute force attack is not be possible in this technique.

- Consider an attack where E takes τ^E and the local fields of his/her hidden units into account. In fact, it is the most successful method for an attacker using only a single CTHLP. E tries to imitate B without being able to interact with A. As long as $\tau^A = \tau^E$, this can be done by just applying the same learning rule as the partners A and B. But in the case of $\tau^E \neq \tau^A$. E cannot stop A's update of the weights. Instead the attacker tries to correct the internal representation of his/her own DHLP using the local fields h_1^E, $h_2^E, \ldots, h_k^E$ as additional information. These quantities can be used to determine the level of confidence associated with the output of each hidden unit. As a low absolute value $\left|h_i^E\right|$ indicates a high probability of $\sigma_i^A \neq \sigma_i^E$, the attacker changes the output σ_i^E of the hidden unit with minimal $\left|h_i^E\right|$ and the total output τ^E before applying the learning rule. Of course, this attack does not always succeed in estimating the internal representation of A's CTHLP correctly. Sometimes there are several hidden units with $\sigma_i^A \neq \sigma_i^E$. In this case the change of one output bit is not enough. It is also possible that $\sigma_i^A = \sigma_i^E$ for the hidden unit with minimal $\left|h_i^E\right|$, so that the correction makes the result worse than before.

5.5 Discussions

The technique is very simple and easy to implement in various high level language. The test results also show that the performance and security provided by the technique is good and comparable to standard technique. The security provided by the CTHLPSCT is comparable with other techniques. To enhance the security of the technique, CTHLPSCT offers changes of some parameters randomly in each session. To generate the secret session key index mask get exchanged between sender and receiver. This technique has a unique ability to construct the secret key at both sides using this exchanged information. Since the encryption and decryption times are much lower, so processing speed is very high. The method takes minimum amount of resources which is greatly handle the resource constraints criteria of wireless communication. This method generates a large number of keys which is the same number of neurons in the map. For ensuring the randomness in every session, some of the parameters get change randomly at each session. CTHLPSCT outperform than existing TPM, PPM, Diffie-Hellman Key exchange methods and does not suffers from Brute Force or Man-In-The-Middle (MITM) attack. No platform specific optimizations were done in the actual implementation, thus performance should be similar over varied implementation platform. The whole procedure is randomized, thus resulting in a unique process for a unique session, which makes it harder for a cryptanalyst to find a base to start with. This technique is applicable to ensure security in message transmission in any form and in any size in wireless communication. Some of the salient features of CTHLPSCT are summarized as follows:

a) Session key generation and exchange – Identical session key can be generate after the tuning of CTHLP in both sender and receiver side with the help of chaos synchronization. So, no need to transfer the whole session key via vulnerable public channel.

b) Degree of security – The technique does not suffers from cipher text only attack, known plaintext attack, chosen plaintext attack, chosen cipher text only attack, brute force attack and attacks during CTHLP synchronization process. It offers authentication steps during synchronization.

c) Variable block size – Encryption algorithm can work with any block length and thus not require padding, which result identical size of files both in original and encrypted file. So, CTHLPSCT has no space overhead.

d) Variable key – $128/192/256$ *bit CTHLP based session key and* $128/192/256$ *bits ACI based keystream with high key space can be used in different sessions. Since the session key is used only once for each transmission, so there is a minimum time stamp which expires automatically at the end of each transmission of information. Thus the cryptanalyst may not be able guess the session key for that particular session.*

e) Complexity – The technique has the flexibility to adopt the complexity based on infrastructure, resource and energy available for computing in a node or mesh through wireless communication. So, the CTHLPSCT may be suitable in wireless communication.

f) *Non-homogeneity – Measures of central tendency, dispersion and Chi-Square value have been performed between the source and corresponding cipher streams generated using proposed technique. All measures indicate that the degree of non-homogeneity of the encrypted stream with respect to the source stream is good. This technique has a better Chi-Square value than technique proposed in chapter 2, 3 and 4.*

g) *Floating frequency – In the CTHLPSCT it is observed that floating frequencies of encrypted characters are indicates the high degree of security for the proposed technique. This technique has a better floating frequency than technique proposed in chapter 2, 3 and 4.*

h) *Entropy – The entropy of encrypted characters is near to eight which indicate the high degree of security of technique. This technique also has a better entropy value than technique proposed in chapter 2, 3 and 4.*

i) *Correlation – The cipher stream generated through the technique is negligibly correlated with the source stream. Therefore the the technique may effectively resist data correlation statistical attack.*

j) *Key sensitivity – The technique generates an entirely different cipher stream with a small change in the key and technique totally fails to decrypt the cipher stream with a slightly different secret session key.*

k) *Security and performance trade-off – The technique may be ideal for trade-off between security and performance of light weight devices having very low processing capabilities or limited computing power in wireless communication.*

Chapter 6

Chaos based Grouped Triple Hidden Layer Perceptron Synchronized Cryptographic Technique (CGTHLPSCT)

6.1 Introduction

In this chapter a novel soft computing assisted cryptographic technique CGTHLPSCT, based on synchronization of Chaos based Grouped Triple Hidden Layer Perceptron (CGTHLP)[208], has been proposed. The CTHLPSCT technique proposed in chapter 5 are only considered synchronization among two parties for generation of session key. Since in CTHLP technique each communicating party has to synchronize with other. So, if there are n parties then total number of synchronizations needed is $O(n^2)$. This is quite computationally complicated especially in wireless communication where the computational power and the resource constrain is a major issue. CGTHLPSCT of this chapter eliminates all the above stated drawbacks of the CTHLPSCT in chapter 5. CGTHLPSCT of the present chapter introduces Chaos based Grouped Triple Hidden Layer Perceptron (CGTHLP) synchronization mechanism for offering synchronization of group of parties. In CGTHLPSCT a key swap over by synchronization among cluster of CTHLP has been proposed which is a fresh addition to the field of cryptography. The proposed technique implements the key swap over technique with the help of complete binary tree framework which makes the technique scales logarithmically with the number of parties participating in the key swap over protocol. In the previous chapter it has been shown how two parties can swap over a common key using synchronization between their own CTHLP. But the problem crop up when group of n parties desire to swap over a key. Using proposed technique a set of n parties can be able to share a common key with only $O(log_2 n)$ synchronization steps. This is logarithmic complexity and feasible in wireless communication with limited amount of resources.

Here, CGTHLP based synchronization is performed for tuning of group of parities by placing on the complete binary tree framework. On the completion of the tuning phase identical session key is generated for the entire group with the help of synchronized CGTHLP. This synchronized network can be used for transmitting message using any light weight encryption/decryption technique with the help of session key of the synchronized network. To illustrate the cryptographic technique using CGTHLP in wireless communication one of the simple and secure encryption/decryption technique has been presented. A plaintext is considered as a stream of binary bits. Particle Swarm Intelligence (PSI) guided enciphering technique[209] with the help of CGTHLP tuned session key is used to generate the cipher text. The plaintext is regenerated from the cipher text using same technique with the help of CGTHLP tuned session key.

Section 6.2 represents a description of proposed technique. Section 6.3 deals with the implementation of the proposed cryptographic technique. Section 6.4 discussed the security issue related to the proposed technique. Discussions are presented in section 6.5.

6.2 The Technique

The technique performs the CGTHLP based synchronization for generation of secret session key for the entire group. This synchronized group session key of the tuned network is used for the transmission of secured message through wireless network with the help of any light weight encryption/decryption algorithm. To illustrate the cryptographic technique in wireless communication one of the simple and secure encryption/decryption technique has been proposed, where plaintext (i.e. the input file) is considered as a stream of binary bits, which is encrypted using PSI generated fittest encryption/decryption keystream. The session key based on CGTHLP is used to encrypt intermediate output which produces final cipher text. The technique uses Chaos based Grouped of THLP to generate the group session key. This scheme implements the key swap over algorithm with the help of complete binary tree framework which makes the algorithm scales logarithmically with the number of parties participating in the key swap over protocol.

In Particle Swarm Intelligence (PSI) based encryption/decryption technique, particle and velocity vector are formed for generation of keystream by setting up the maximum dimension of each particle and velocity vector. Each particle position and probability value is evaluated. Probability value of each particle can be determined by dividing the position of a particular particle by its length. If probability value of a particle is less than minimum probability value then a velocity is applied to move each particle in a new position. After that probability value of the particle at new position is calculated. A threshold value is selected to evalutae against velocity level of each particle. Particle having highest velocity more than predefined threshold value is selected as a keystream for encryption. If the length of the plaintext is grater than the length of the PSI based keystream then the values of the keystream are added to a predetermined value to generate the keys for the characters in the plaintext which is at a position grater than the length of the keystream. Stream of plaintext is then encrypted using the PSI based keystream/extended keystream. Finally a cascaded *Exclusive-OR* operation is performed between PSI encrypted text and the CGTHLP based session key to generate final cipher text.

All the parties in the group have the same CGTHLP synchronized group session key. This session key is used to perform first step of the deciphering technique. In the next step, PSI guided keystream based deciphering operation is performed to regenerate the plaintext.

The CGTHLPSCT does not cause any storage overhead. This greatly handles the resource constraints criteria of wireless communication. A comparison of CGTHLPSCT with previously proposed technique in chapter 5, chapter 4, chapter 3, chapter 2, existing Tree Parity Machine (TPM), Permutation Parity Machine (PPM), and industry accepted AES, RC4, Vernam Cipher, Triple DES (TDES) and RSA have been done. Analyses of results are given in chapter 7.

In CGTHLPSCT, encryption algorithm takes the plaintext as a binary stream of bits which is encrypted using PSI generated fittest encryption keystream based encryption process. CGTHLP synchronized group session key is used to further encrypt the PSI encoded text to produce final cipher text. The algorithm for the complete process is given in section 6.2.1.

6.2.1 CGTHLPSCT Algorithm at Sender

Input : Source file/source stream i.e. plaintext

Output : Encrypted file/encrypted stream i.e. cipher text

Method : The process operates on binary stream and generates encrypted bit stream through CGTHLP guided Ant Colony Intelligence (PSI) based encryption operations.

Step 1. Perform tuning of CGTHLPs to generate common group secret session key.

Step 2. Generates PSI based fittest encryption keystream.

Step 3. Perform PSI based encryption operation on the plaintext.

Step 4. Perform cascaded Exclusive-OR operation between CGTHLP based session key and outcomes of step 3.

Step 1 of the algorithm generate common session key through synchronization of CGTHLP at both end. The detailed step is discussed in section 6.2.1.1. Step 2 of the algorithm generates PSI based fittest encryption keystream. The detailed description of the process is given in section 6.2.1.2. Algorithm for performing PSI based encryption operation (step 3) on the plaintext is discussed in 6.2.1.3. The technique of cascading encryption process (step 4) which takes the intermediate output generated in step 3 is given in details in section 6.2.1.4.

6.2.1.1 Chaos based Grouped Triple Hidden Layer Perceptron (CGTHLP) Synchronization and Session Key Generation

Chaos based Grouped Triple Hidden Layer Perceptron (CGTHLP) guided synchronization mechanism has been proposed here to improve the efficiency and enhance the security of the Chaos based Triple Hidden Layer Perceptron (CTHLP) guided synchronization between two parties, proposed in chapter 5. The most important hazard of cryptography is how to firmly swap over the shared secrets between the parties. As a result, key exchange protocols are mandatory for transferring keys in a protected manner. As the same time as key exchange protocols are developed for exchanging key between two parties, many applications do necessitate the need of swapping over a secret key among group of parties. So, if there are n parties then total number of synchronizations needed is $O(n^2)$. Which is quite computationally complicated especially in wireless communication where the computational power and the memory constrain is a major issue. The method of this chapter eliminates all the above stated drawbacks of the method of chapter 5. The method of the current chapter introduces CGTHLP synchronization mechanism for offering synchronization of group of parties. Here, a complete binary tree framework is used to synchronize group of CTHLP, which makes the algorithm scales logarithmically with the number of parties participating in the key swap over protocol. In the CGTHLP synchronized group key exchange algorithm, n CTHLP need to synchronize together and they are represented by an m number of leaves of a complete binary tree where m is defined as $m = 2^{log_2 n}$. In the technique n CTHLP having identical structure of three hidden layers are participated for group key generation purpose. Here, each CTHLP considered as a node of a complete binary tree framework. CTHLP in the group initially start Chaos synchronization between two pair of leaves to construct a common seed value at both sides. This Chaos synchronized identical seed values is used to generate the common input vector for the two nodes $node_i$(sending node) and $node_j$ (receiving node) of the complete binary tree framework participating in the synchronization process. Now, two nodes $node_i$ and $node_j$ in the same pair start synchronization with common input vector and completely random weight vector. In each time both $node_i$ and $node_j$ compute their final output based on input and weight vector, and communicate to each other. If both are be in agreement on the mapping between the present input and the output, their weights are updated according to an appropriate learning rule. After synchronization procedure weight vector of both $node_i$ and $node_j$ become identical. Now one of these two synchronized $node_i$ and $node_j$ is elected for proceed further synchronization steps with the elected $node_i$ form other branch of the tree. If all the nodes in the group are synchronized in this manner then indistinguishable weight vector forms the group session key for a particular session. Authentication steps also get performed parallel to the synchronization steps. Both parties' uses identical input vector generated using Chaos synchronized seed and use anonymous random weight vector to initializes the weights of the synaptic links of CTHLP. Identical input vector for both the parties kept secret for security reason. Attackers has no idea about the internal state of both the CTHLPs of $node_i$ and $node_j$ at a particular instant of time and this is achievable by keeping secret the common input vector and internal state of the CTHLP. At the time of key exchange procedure key authentication technique is also performed parallel by selecting last m bits of the identical input vector and transmitting directly as an output bit towards the other party over public channel. Receiving party checks these last m bits to its last m bits of identical input vector. If both the sequences are same then both are authenticated otherwise not. Attacker does not have identical input vector like sender and receiver. By sniffing the public channel attacker can gets some bits but from them attacker will not be able to understand which one is output bit of the machine and which one is one of the bits of m bits sequence of the identical input vector. Even if attacker hacks the m bits then for getting the rest of the $(d - m)$ bits of the identical input vector attacker has to perform checking with all $(d - m)$ combination that is computationally infeasible. Here d is the total number of bits in the identical input vector of the technique offers synchronization and authentication step in parallel. An attacker also cannot distinguish an authentication step from a synchronization step from observing the exchanged outputs. Attacker thus does not know, whether the currently observed output bit is used for either of the two purposes if the attacker does not know the secret identical common input vector.

In a complete binary tree a node with no child is called leaf node. Figure 6.1 represents the scenario when $leaves = 8$ (number of leaves in the figure 6.1) $j = 1$ (j is the round number). Then complete binary tree is divided into $number\ of\ leaves/2^j$ subtrees each with 2^j leaves i.e. four sub trees with two leaves each represented by red oval. Each pair of leaves sharing the same parent (two leaves in a single red oval) involved in mutual learning at height four.

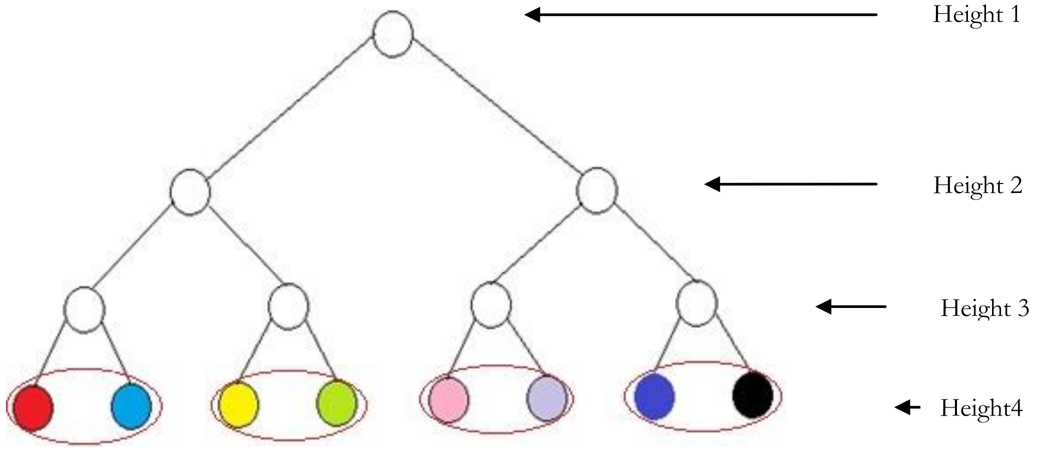


Figure 6.1: Initial state of group synchronization

After round 1 each pair of leaves sharing the same parent become synchronized using mutual learning step. Figure 6.2 represents the scenario where same colored leaves become synchronized. From each sub tree a node is nominated as a leader among the nodes having same parents to perform the next round of operation.

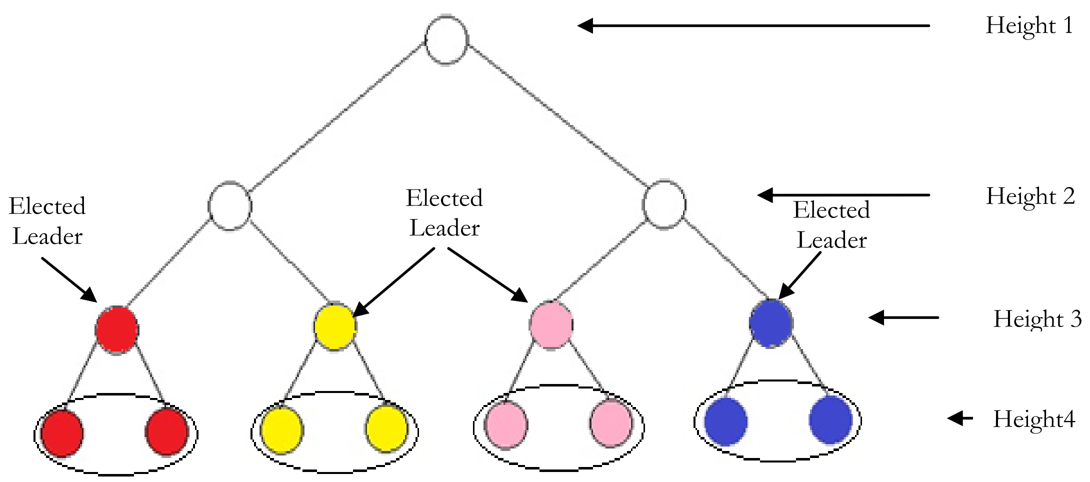


Figure 6.2: First round of group synchronization

At round 2 when $j = 2,\ leave = 8$ (number of leaves in a complete binary tree framework). Then tree is divided into two sub trees with four leaves at height 3 shown in figure 6.3.

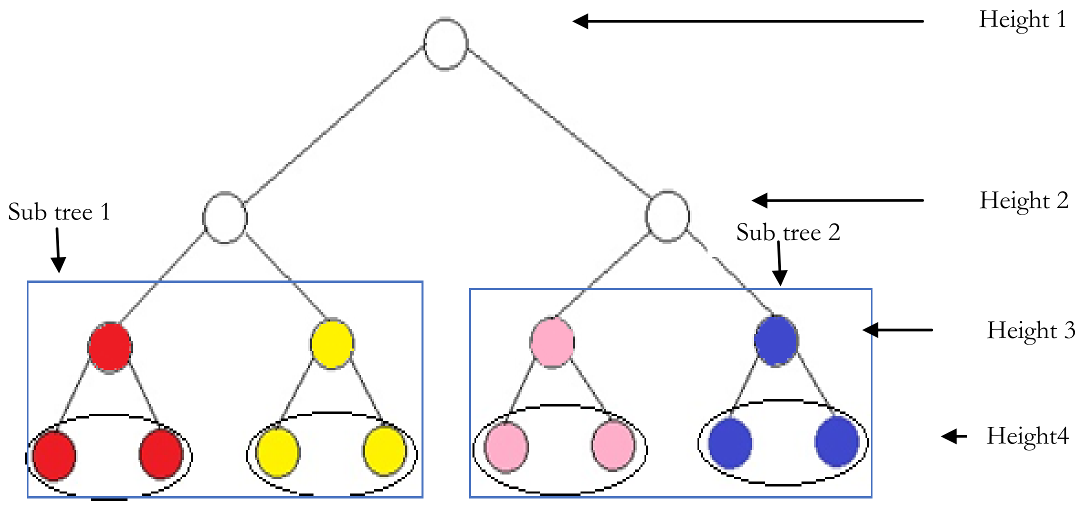


Figure 6.3: Second round of group synchronization

In this way next rounds are performed until the root node at height 1 is synchronized. On the completion of the synchronization process all the nodes in the group become synchronized based on a common group session key. In proposed technique CTHLPs start synchronization by exchanging control frames. The process involves message integrity and synchronization test. CGTHLP synchronization uses transmission of control frames at the time of three way handshaking based TCP connection establishment phase, as given in table 6.1.

Table 6.1
Control frames of CGTHLP synchronization

Frame	Description
SYN	SYN frame transmitted to the $node_j$ for synchronization in connection establishment phase
ACK_SYN	ACK_SYN frame transmitted to the $node_i$ for positive acknowledgement respect to SYN frame
NAK_SYN	NAK_SYN frame transmitted to the $node_i$ for negative acknowledgement respect to SYN frame
FIN_SYN	FIN_SYN frame transmitted by either party for closing the connection

The SYN frame is used to establish the connection to the other side. It carries index information of different initial parameters. On transmitting the SYN frame, the $node_i$ starts a timer and waits for a reply from the receiver. If the $node_j$ does not take any action until a certain time limit and number of attempts exceeded a certain value, the $node_i$ restarts the synchronization procedure. When the $node_j$ receives the SYN frame, it carry out the integrity test. If the messages are received as sent (with no replication, incorporation, alteration, reordering, or replay) the $node_j$ will execute the synchronization check. The $node_i$ and $node_j$ have an identical T variable formally store in their respective memory. The $node_i$ sends the encrypted T to the receiver. Here the $node_j$ utilizes its $128/192/256$ bits weights to decrypt the encrypted T. If the result is identical to T formerly stored in $node_j$ memory i.e. $(Decrypt_{node\ j\_weight}(Encrypt_{node\ i\_weight}(T)) = T$ then the networks are synchronized. This is the best case solution where $node_i$ and $node_j$ arbitrarily choose weight vector which are identical. So, networks are synchronized at initial stage. The $node_j$ should send the frame FIN_SYN to alert the sender. But most of the time this best case is not achievable. If decryption algorithm does not produce predictable result, the $node_j$ should use the chaos synchronized secret seed of $node_i$'s produce the input vector (X) which is identical to $node_i$. With this input vector the $node_j$ will work out its $output$ $(\tau^{node\ j})$. If the $node_i$ and $node_j$ outputs are different, the $node_j$ should not fine-tune its weights and inform the

$node_i$ its output. The $node_j$ sends the $NACK\_SYN$ frame to notify the $node_i$, with the same ID value. The proposed NAK_SYN frame in this methodology is used for providing the negative acknowledgement in respect to the SYN frame. If $node_j$'s output is equal to $node_i$'s output i.e. $(\tau^{node\,j} = \tau^{node\,i})$ then $node_j$ update it weights. At the end of weights update, the $node_j$ should report the $node_i$ that outputs are equal. The $node_j$ uses the ACK_SYN frame to notify the $node_i$, with the same ID value received from $node_i$. The proposed ACK_SYN frame in this methodology is used for providing the positive acknowledgement in respect to the SYN frame. On receipt of ACK_SYN, the $node_i$ also updates its weight. If $node_i$ receives $ACK\ \_SYN$ it should update its weights. The $node_i$ will create new synchronization frame until receive the frame FIN_ACK from $node_j$. When the $node_i$ receives the frame FIN_ACK, it stops the further synchronization. The proposed FIN_SYN frame in this methodology is used for closing the connection. At end of synchronization, both nodes provide the identical weight vector which acts as a session key identical to both. Table 6.2 shows the different frames and their corresponding command codes

Table 6.2
CGTHLP control frames and their command codes

Frame	Command
SYN	0000
FIN_SYN	0001
ACK_SYN	0010
$NAK\_\ SYN$	0011
$AUTH$	0100
$Reserved$	0101-1111

The identifier is the function of informing the $node_i$ and $node_j$ where the message is a recent message. The variable ID starts with zero and is incremented every time that the $node_i$ sends a synchronization frame. The figure 6.4 shows the exchange of frames during CGTHLP synchronization process.

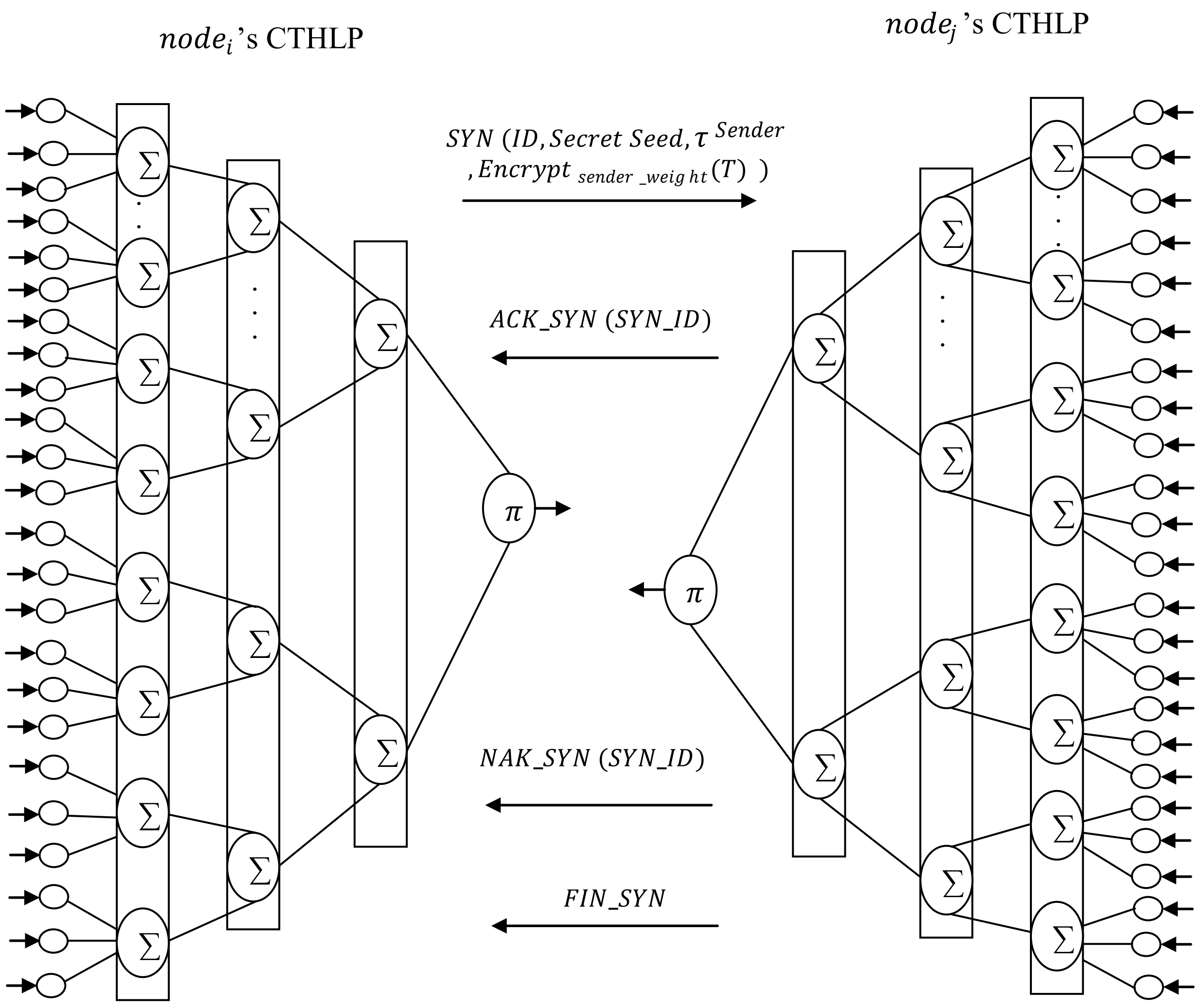


Figure 6.4: Exchange of control frames between $node_i$ and $node_j$ during CGTHLP synchronization

The detailed frame format of SYN frame is discussed in section 6.2.1.1.1. The detailed frame format of ACK_SYN frame is given in section 6.2.1.1.2. The frame format of NAK_SYN frame has been discussed in section 6.2.1.1.3. The frame format of FIN_SYN frame is discussed in section 6.2.1.1.4.

6.2.1.1.1 Synchronization (SYN) Frame

During synchronization process $node_i$ constructs a SYN frame and transmit to the $node_j$ for handshaking in connection establishment phase. $node_i$ utilizes its initial 128 weights as key for encryption of T variable (formerly stored in its memory) $Encrypt_{node\ i\_weig\ ht}(T)$. $node_i$ constructs a SYN frame and transmitted to the $node_j$ for handshaking purpose in connection establishment phase. SYN usually comprises of $Command\ Code, SYN\ ID, node_i$ $output(\tau^{node\ i})$, $Encrypt_{node\ i\_weig\ ht}(T)$, authentication bits and CRC. SYN frame has the fixed $Command\ Code$ i.e. 0000. So, number of bits required for $Command\ Code$ is four. $SYN\ ID, node_i\ output(\tau^{node\ i})$,$Encrypt_{node\ i\_weig\ ht}(T)$, authentication bits and CRC needs eight bits, one bits, 128 bits, m bits and sixteen bits respectively. When the $node_j$ receive SYN frame, the receiver should carry out integrity test. $node_j$ performs Integrity test on receiving the SYN frame. If the messages are received as sent (with no replication, incorporation, alteration, reordering, or replay) the $node_j$ will execute the synchronization test. In synchronization test $node_j$ utilize its 128 first weights as key for decryption of $Encrypt_{node\ i\_weig\ ht}(T)$ that was received from the $node_i$. After decryption operation if $(Decrypt_{node\ j\ \_weig\ ht}(Encrypt_{node\ i\_weig\ ht}(T)) = T$ then networks are synchronized. Figure 6.5 shows the complete frame format of SYN frame.

$Command\ Code$ 0000	$SYN\ ID$	τ^{node_i}	$Encrypt_{node_{i\_weig\ ht}}(T)$	$Authenticaion\ bits$	CRC ($Cyclic\ Redundancy\ Checker$)
4	8	1	128	m	16 ($bits$)

Figure 6.5: Synchronization (SYN) frame

6.2.1.1.2 Synchronization (ACK_SYN) Acknowledgement Frame

ACK_SYN frame send by the $node_j$ to the $node_i$ in respect of ACK frame for positive acknowledgement of the parameters value. This proposed frame comprises of $Command\ code, SYN\ ID, CRC$. $Command\ Code$ needs four bits. The ACK_SYN frame has the fixed $Command\ Code$ i.e. 0010. Eight bits is used for representing the $SYN\ ID$ and CRC needs sixteen bits for error checking purpose. Now check the condition i.e. If

$(Decrypt_{node\,j\,\_weig\,ht}(Encrypt_{node\,i\,\_weig\,ht}(T)) \neq T$ then $node_j$ use the $Secret\,Seed$ received from $node_i$ to produce the $node_j$ inputs (X) identical to $node_i$ input (X) and calculates the output $\tau^{\,node\,j}$. If $(\tau^{\,node\,j} = \tau^{\,node\,i})$ then $node_j$ should update their weights where $\sigma_k^{node\,i/node\,j} = \tau^{node\,i/node\,j}$ using learning rule. At end of weight updation of the $node_j$, then it sends ACK_SYN with the same ID to instruct the $node_i$ for updating the weights. If $node_i$ receives ACK_SYN it should update its weights. Figure 6.6 shows the complete frame format of ACK_SYN frame.

$Command\ Code$ 0010	$SYN\ ID$	CRC $(Cyclic\ Redundancy\ \ Checker)$
4	8	16 $(bits)$

Figure 6.6: Acknowledgement of Synchronization (ACK_SYN) frame

6.2.1.1.3 Negative Acknowledgement (NAK_SYN) Frame of Synchronization

NAK_SYN frame send by the $node_j$ to the $node_i$ in respect of ACK frame for negative acknowledgement of the parameters value. This proposed frame comprises of $Command\,Code,\ SYN\,ID,\ CRC.$ $Command\,Code$ needs four bits. The ACK_SYN frame has the fixed $Command\ Code$ i.e. 0011. Eight bits are used for representing the $SYN\,ID$ and CRC needs sixteen bits for error checking purpose. If $(\tau^{\,node\,j} \neq \tau^{\,node\,i})$ then the $node_j$ sends the message NAK_SYN to notify the $node_i$. If the $node_j$'s and $node_i$'s outputs are different, the $node_j$ should not fine-tune its weights and inform the $node_i$. The $node_j$ sends the message NAK_SYN to notify the $node_i$, with the same ID value. Figure 6.7 shows the complete frame format of NAK_SYN frame.

$Command\ Code$ 0011	SYN_ID	CRC $(Cyclic\ Redundancy\ \ Checker)$
4	8	16 $(bits)$

Figure 6.7: Negative Acknowledgement of Synchronization (NAK_SYN) frame

6.2.1.1.4 Finish Synchronization (FIN_SYN) Frame

FIN_SYN frame send by the either party for finish the synchronization process. This proposed frame comprises of $Command\ Code, SYN\ ID, CRC$. $Command\ Code$ needs four bits. The FIN_SYN frame has the fixed $Command\ Code$ i.e. 0001. Eight bits is used for representing the $SYN\ ID$ and CRC needs sixteen bits for error checking purpose. If $(Decrypt_{node_j\_weight}(Encrypt_{node_i\_weight}(T)) = T$ then networks are synchronized. $node_j$ sends the FIN_SYN frame to the $node_i$. Figure 6.8 shows the complete frame format of FIN_SYN frame.

$Command\ Code$ 0001	$SYN\ ID$	CRC ($Cyclic\ Redundancy\ \ Checker$)
4	8	16 ($bits$)

Figure 6.8: Finish Synchronization (FIN_SYN) frame

The CGTHLP synchronization algorithm for generating synchronized session key is discussed in section 6.2.1.1.5. Section 6.2.1.1.6 presents the computational complexity of the CGTHLP synchronization algorithm and CGTHLP learning is discussed in section 6.2.1.1.7.

6.2.1.1.5 CGTHLP Synchronization

Chaos synchronization initiated between $node_i$ and $node_j$ to construct a common seed value at both sides. The Chaos synchronized identical seed value is used to generate the common input vector for $node_i$ and $node_j$. Two CTHLPs one at $node_i$ and another at $node_j$ start with identical input vector and anonymous random weight vector. In each time both CTHLPs compute their final output based on input and weight vector, and communicate to each other. If both are be in agreement on the mapping between the present input and the output, their weights are updated according to an appropriate learning rule. After synchronization procedure of all parties (nodes in a complete binary tree) in the group weight vector of the group CTHLPs become identical. These indistinguishable weight vector forms the session key for a particular session. Authentication steps also get performed parallel to the synchronization steps.

Input : *Tuning parameters* $(\sigma, b, r, x_1, y_2$ *and* $z_2)$, *random weights, group of* n *CTHLPs with* l *leaves in a complete binary tree framework*

Output : *Secret session key through group synchronization*

Method : *Two CTHLPs in a pair are be in agreement on the mapping between the present input and the output, their weights are updated according to an appropriate learning rule. After synchronization procedure weight vector of both CTHLPs become identical. Now one of these two synchronized CTHLPs is elected for proceed further synchronization steps with the elected CTHLP form other branch of the tree. If all the CTHLP are synchronized in this manner then indistinguishable weight vector forms the group session key for a particular session.*

Step 1. *Represents the* n *CTHLPs by the* l *leaves node in a complete binary tree.*

Step 2. *Set* $h =$ *height of the complete binary tree and* $pos = h - 1$ *where* pos *denotes the initial starting position of the mutual learning algorithm*

Step 3. $node_i$ *initializes the value of* σ *and* b, *after that value of* b *is send to the* $node_j$.

Step 4. $node_j$ *initializes the value of* r.

Step 5. $node_i$ *generates the point* x_1 *and* z_1.

Step 6. $node_j$ *generates the point* y_2*and* z_2.

Step 7. $node_i$ *sends* x_1 *to* $node_j$ *and* $node_j$ *sends* y_2 *and* z_2 *to* $node_i$.

Step 8. $node_j$ *calculates the new value of* $\dot{y}_2$ *and* $\dot{z}_2$*with the help of* r *and* b *using the equations 6.1 and 6.2 then returns the value of* $\dot{y}_2$ *and* $\dot{z}_2$ *to the* $node_i$.

$$\dot{y}_2 = rx - y_2 - xz_2 \tag{6.1}$$

$$\dot{z}_2 = xy_2 - bz_2 \tag{6.2}$$

Step 9. $node_i$ *calculates the value of* $\dot{x}_1$ *and* $\dot{z}_1$ *with the help of* y_2, σ *and* b *using equations 6.3 and 6.4 then sends the value of* $\dot{x}_1$ *to the* $node_j$ *and so on.*

$$\dot{x}_1 = \sigma(x_1 - y_2) \quad (6.3)$$

$$\dot{z}_1 = x_1 y_2 - bz_1 \quad (6.4)$$

Step 10. *$node_i$ generates a nonce. This nonce gets encrypted using a symmetric cipher with z_1 as the key and sends the results of the encryption using equation 6.5.*

$$En\_Nonce = Encrypt_{z_1}(Nonce) \quad (6.5)$$

Step 11. *The $node_j$ decrypts En_Nonce using z_2 as the key, performs a defined function on it using equation 6.6 and 6.7.*

$$De\_Nonce = Decrypt_{z_2}(En\_Nonce) \quad (6.6)$$

$$Fn\_Nonce = f(De\_Nonce) \quad (6.7)$$

Step 12. *The $node_j$ encrypts the result of the previous step using z_2 as the key and sends the result to the sender illustrated in equation 6.8.*

$$En\_Fn\_Nonce = Encrypt_{z_2}(Fn\_Nonce) \quad (6.8)$$

Step 13. *The $node_i$ decrypts this message using z_1 as the key, performs the inverse of the pre-defined function and checks if the original nonce is obtained as shown in equation 6.9.*

$$Nonce = f^{-1}\left(Decrypt_{z_1}(En\_Fn\_Nonce)\right) \quad (6.9)$$

Step 14. *If synchronization is not achieved, the process is repeated from step 5.*

Step 15. *If synchronization is achieved i.e. $z_1 = z_2$ then z_1 is used as a seed for a pseudo random number generator to generate identical input vector(X) at both node.*

Step 16. *Initialization of synaptic links between input layer and first hidden layer and between first hidden layer and second hidden layer using random weights values. Where, $W_{ij} \epsilon \{-L, -L+1, \dots, +L\}$.*

Repeat step 17 to step 26 until the full synchronization is achieved,

Step 17. *The input vector(X) is generated both end using the Chaos synchronized seed value.*

Step 18. *Computes the values of hidden neurons by the weighted sum over the current input values. Each hidden neuron in first Hidden layer* *produces $\sigma^1{}_i$ values. Similarly, each hidden neuron in second hidden layer produces $\sigma^2{}_i$ value. Each hidden neuron in third hidden layer produces $\sigma^3{}_i$ value. These can be represented using equation 6.10, 6.11 and 6.12.*

$$\sigma^1{}_i = sgn\left(\sum_{j=1}^{N} W_{i,j}\, X_{i,j}\right) \tag{6.10}$$

$$\sigma^2{}_i = sgn\left(\sum_{j=1}^{N} \sigma_i^1\right) \tag{6.11}$$

$$\sigma^3{}_i = sgn\left(\sum_{j=1}^{N} \sigma_i^2\right) \tag{6.12}$$

$sgn(x)$ *is a function represents in equation 6.13, which returns* $-1, 0$ *or* 1*:*

$$sgn(x) = \begin{cases} -1 \ \ if\ x < 0 \\ 0 \ \ \ if\ x = 0 \\ 1 \ \ if\ \ x > 0 \end{cases} \tag{6.13}$$

If the weighted sum over its inputs is negative then set $\sigma_i = -1$. *Hence, set* $\sigma_i = +1$, *if the weighted sum over its inputs is positive, or else if weighted sum is zero then it is set to,* $\sigma_i = 0$.

Step 19. *Compute the value of the final output neuron by computing multiplication of all values produced by* $K2$ *no. hidden neurons using equation 6.14.*

$$\tau = \prod_{i=1}^{K2} \sigma_i^3 \tag{6.14}$$

Step 20. $node_i$ *utilizes its* 128 *weights as key for encryption of* T *variable (formerly stored in its memory)* $Encrypt_{node\ i\_weig\ ht}(T)$.

Step 21. $node_i$ *constructs a SYN frame and transmitted to the* $node_j$ *for handshaking purpose in connection establishment phase.* SYN *usually comprises of the fields* $Command\ Code,$ *ID,*$node_i$ *output* $(\tau^{\,node\ i})$, $Encrypt_{node\ i\_weig\ ht}(T)$ *and* CRC *(Cyclic Redundancy Checker) and last* m *bits of the identical input vector. In this way performed authentication step parallel by selecting last* m *bits of the identical input vector and transmitting towards the other party over public channel using* SYN *frame.*

Step 22. $node_j$ *performs Integrity test after receiving the* SYN *frame. Then* $node_j$ *perform authentication step to*

Check $if\ (\ node_i\ \ (m\ bits) =\ node_j\ \ (m\ bits))\ then$

$authentication\ = TRUE$

$Else$

$authentication\ = FALSE$

If authentication is true then receiver utilize its 128 *weights as key for decryption of* $Encrypt_{node\ i\_weig\ ht}(T)$ *that was received from the sender.*

If authentication is false then $node_j$ *sends* ACK_NAK *to sender.*

Step 23. *If* $(Decrypt_{node\ j\_weig\ ht}(Encrypt_{node\ i\_weig\ ht}(T)) = T$ *then networks are synchronized. Go to step 25.*

Step 24. *If* $(Decrypt_{node\ j\_weig\ ht}(Encrypt_{node\ i\_weig\ ht}(T)) \neq$ *then* $node_j$ *use the Chaos based secret seed to produce the* $\tau^{node\ j}$ *input vector* (X) *identical to sender input vector*(X) *and calculates the output* $\tau^{node\ j}$ *using step 18 and step 19*

Step 25. *If* $(\tau^{node\ j} = \tau^{\ node\ i})$ *then performs the following steps.*

Step 25.1 $node_j$ *update their weights where* $\sigma_k^{node\ i/node\ j} = \tau^{node\ i/node\ j}$ *using any of the learning rules discussed in chapter 1 section 1.8.*

Step 25.2 *At the end of* $node_j$ *weights update, the* $node_j$ *sends* ACK_SYN *to instruct the sender for updating the weights using step 25.1.*

Step 25.3 $node_i$ *transmits* $Encrypt_{node\ i\_updated\ \_weig\ ht}(T)$ *to* $node_j$.

Step 25.4 $node_j$ *checks*

$if\ (Decrypt_{node\ j\_updated\ \_weig\ ht}(Encrypt_{node\ i\_updated\ \_weig\ ht}(T)) = T$

then networks are synchronized. Go to step 27.

Step 25.5 *Perform the following checking*

$if\,(Decrypt_{node_{j}\_updated\ \_weig\,ht}(Encrypt_{node_{i}\_updated\ \_weig\,ht}(T)) \neq T$

then networks are still not synchronized. Go to step 25.1.

Step 26. *If* $(\tau^{\,node_j} \neq \tau^{\,node_i})$ *then the* $node_j$ *sends the message* NAK_SYN *to notify the* $node_i$. *Go to step 17.*

Step 27. *Finally, the* $node_j$ *sends the frame* FIN_SYN *to inform the* $node_i$ *to finish the synchronization phase.*

Step 28. *Increment* j *by* 1 *and from each sub tree, a node is nominated and the mutual learning algorithm is executed by the nominated nodes and the rest if the nodes follow then goto step 3.*

Step 29. *Terminates when the algorithm reaches the root. Hence, CTHLPs are synchronized and share the same weight vector. Otherwise, go to step 1.*

6.2.1.1.6 Complexity Analysis

For every step j (starting at $j = 1$) of the CGTHLP algorithm the complete binary tree is divided into $\frac{m}{2^j}$ sub trees each with 2^j leaves, where m is the total number of leaves. When $j = 1$ number of sub tree will be $\frac{m}{2}$ along with two leaves. When $j = 2$ number of sub tree will be $\frac{m}{4}$ along with four leaves so on. In this way a structure of complete binary tree get form and the height of the binary tree will be $log_2 n$. In CGTHLP synchronization algorithm node_{i} initialization of the value of σ and b takes needs unit amount of computation. $node_j$ initialization of the value of r also takes unit amount of computation. Generation of the point x_1 and z_1 takes unit amount of computation. Generation of the point y_2 and z_2 takes unit amount of computation. $node_j$ calculates the new value of y_2 and z_2 with the help of r and b. This step also takes unit amount of computation. $node_i$ calculates the value of x_1 and z_1 with the help of y_2, σ and b. This step also takes unit amount of computation. $node_i$ generates a nonce. This nonce is encrypted using a symmetric cipher with z_1 as the key and sends the results of the encryption. This step needs $(nonce\ length)$ amount of computation. The $node_j$ decrypts En_Nonce using z_2 as the key. It also takes $(nonce\ length)$ amount of computation. The $node_j$ encrypts the result of the previous step using z_2 as the key. It takes $(message\ length)$ amount of computation. The $node_i$ decrypts this message using z_1 as the key, performs the inverse of the pre-defined function and checks if the original nonce is or not. It takes $(encrypted\ message\ length)$ amount of computation. Initialization of weight vector takes $(N \times K1 + K1 \times K2 + K2 \times K3)$ amount of computations. For example, if $N = 2,\ K1 = 2,\ K2 = 3,\ K3 = 2$ then total numbers of synaptic links (weights) are $(2 \times 2 + 2 \times 3 + 3 \times 2) = 16$. So, it takes 16 amount of computations. Generation of N number of input vector for each $K1$ number of hidden neurons takes $(N \times K1)$ amount of computations. Computation of the hidden neuron outputs takes $(K1 + K2 + K3)$ amount of computations. Where $K1, K2$ and $K3$ are the number of hidden units in 1st, 2nd and 3rd layer respectively. Computation of final output value takes unit amount of computation because it needs only a single operation to compute the value. Encryption of T variable using *Exclusive-OR* operation also takes unit amount of computations. Decryption of T variable using *Exclusive-OR* operation also takes unit amount of computations. Checking if $(Decrypt_{node_j\_weight}(Encrypt_{node_i\_weight}(T)) = T$ or not takes unit amount of computation. Weight updating procedure takes place where $\sigma_k^{node_i/node_j} = \tau^{node_i/node_j}$ using any of the learning rules which takes $\left(no.of\ \sigma_k^{node_i/node_j} = \tau^{node_i/node_j}\right)$ amount of computations. Increment j by 1 and from each sub tree, a node is nominated and the chaos synchronization and mutual learning algorithm is executed by the nominated nodes and the rest if the nodes follow. This take $\mathrm{O(no.\,of\ pair\ in\ mutual\ learning)}$ computation. Terminates when the algorithm reaches the root. Hence, all the CTHLPs are synchronized and share the same weight vector. From the above complexity analysis it can be observed that each level of complete binary tree needs at most $\mathrm{O(no.\,of\ pair\ in\ mutual\ learning)}$ computation complexity and height of the complete binary tree is $log_2 n$.

In best case of CGTHLP synchronization algorithm, $node_i$'s and $node_j$'s arbitrarily chosen weight vectors are identical. So, networks are synchronized at initial stage do not needs to update the weight using learning rule. Here, $(nonce\ length + message\ length + encrypted\ message\ length + (N \times K1) + (N \times K1 + K1 \times K2 + K2 \times K3) + (K1 + K2 + K3) \times no.of\ pair\ in\ mutual\ learning \times log_2 n)$ amount of computation is needed in best case which is in form of $\mathrm{O(Generation\ of\ common\ seed\ value} +$

$\mathrm{initialization\ of\ input\ vector} + \mathrm{initialization\ of\ weight\ vector} +$
$\mathrm{Computation\ of\ the\ hidden\ neuron\ outputs} \times \mathrm{no.\,of\ pair\ in\ mutual\ learning} \times \log_2 \mathrm{n}).$

If the $node_i$'s and $node_j$'s arbitrarily chosen weight vector are not identical then in each iteration the weight vectors of the hidden unit which has a value equivalent to the pereceptron output are updated according to the learning rule. This scenario leads to average and worst case situation where I number of iteration to be performed to generate the identical weight vectors at both ends. So, the total computation for the average and worst case is $\big(nonce\ length + message\ length + encrypted\ message\ length + (N \times K1) + (N \times K1 + K1 \times K2 + K2 \times K3) + (K1 + K2 + K3)\big) + \big(I \times (no.of\ \sigma_k^{Sender\ /Receiver} = \tau^{Sender\ /Receiver} \times no.of\ pair\ in\ mutual\ learning \times log_2 n\ )\big)$ which is can be expressed $\mathrm{O(Time\ complexity\ in\ first\ iteration}\ + \mathrm{(No.\,of\ iteration} \times \mathrm{No.\,of\ weight\ updation} \times \mathrm{no.\,of\ pair\ in\ mutual\ learning} \times \log_2 \mathrm{n})).$

6.2.1.1.7 CGTHLP Learning Mechanism

In learning mechanism if the output bits are different for $node_i$ (A) and $node_j$ (B) i.e. $\tau^A \neq \tau^B$, nothing get changed. If $\tau^A = \tau^B = \tau$, only the weights of the hidden units with $\sigma_k^{A/B} = \tau^{A/B}$ will be updated. The weight vector of this hidden unit is adjusted using any of the following learning rules discussed in chapter 1 section 1.8. For small L values Hebbian takes less synchronization steps than other two learning rules in the range of $2-2-3-2-5$ $(N-K1-K2-K3-L)$ to $2-2-3-2-15$ but as the L value increases Hebbian rule takes more steps to synchronize than other two learning rules. Here, Anti-Hebbian rules take fewer steps than the other two learning rules in the range of $2-2-3-2-8-20$ to $2-2-3-2-30$. Random walk outperform from $3-2-2-8-35$ and beyond that. The most vital findings is that if the synaptic depth i.e. weight range (L) is increased, the complexity of a successful attack grows exponentially, but there is only a polynomial increase of the effort needed to generate a key. So, increasing the L value security of the system can be increased.

6.2.1.2 Particle Swarm Intelligence (PSI) based Fittest Keystream Generation

In this section Particle Swarm Intelligence (PSI) based kestream generation technique for message encryption/decryption has been presented to illustrate the complete cryptographic technique. Instead of this technique any other light weight encryption/decryption technique also may use for exchanging message between sender and receiver.

The PSI technique begins with an initial population comprises of set of valid and complete set of particles. Then some operators like particles local best and global best positions along with velocity updating rules are used to generate feasible valid particle from the existing one. In this technique a collection of alphanumeric characters is called a keystream and each character in the keystream is known as key. The keystream measurement lengthwise constantly be less than or equal to the plaintext to be encrypt and production of keystream is based on sharing of characters in the plaintext for encryption principle.

In PSI based approach a particle is used to designate a keystream (set of alphanumeric characters). Each particle can have numerous dimensions. Each dimension signifies an individual key inside that keystream. The dimensions in the keystream can be packed or unpacked. For example if the ceiling of dimension of each particle is equal to 256 then it is characterized by equation 6.15.

$$Particle_i \ or \ Keystream_i = (Key_1, Key_2, \dots , Key_{256}) \quad (6.15)$$

Which actually indicate a keystream comprises of 256 keys i.e. 256 alphanumeric characters. Keystream length can be attained by counting number of dimensions are packed in the keystream. Usually keystream length is less than or equal to the plaintext. With 256 alphanumeric characters various keystream can be generated of preset rigid length by variation of these prearranged set length characters ordering all viable ways devoid of any recurrence. So, for example if total number of alphanumeric characters $= 256$ and if key stream length $= 192$ then among 256 alphanumeric characters 192 alphanumeric characters are nominated such a way so that by ordering all achievable ways with no replication these 192 characters forms multiple keystream having monotonous length i.e. 192. For an example if four characters A, S, M, K are taken to structure keystream of length four among 256 alphanumeric characters. Then there are 24 doable ways of obtaining keystream which are as follows.

$$ASMK, ASKM, AMSK, AMKS, AKMS\ AKSM,$$
$$SAMK, SAKM, SMAK, SMKA, SKMA, SKAM,$$
$$MASK, MAKS, MSAK, MSKA, MKSA, MKAS,$$
$$KASM, KAMS, KSAM, KSMA, KMSA, KMAS$$

Using 256 characters total number of generated potential keystream is given in equation 6.16.

$$\sum_{c=1}^{256} \frac{256!}{(256-c)!} \approx 256!\,(e) \approx 256! \times 2.718 \qquad (6.16)$$

According to PSI technique each particle should have an allied velocity. The PSI technique also offers velocity for each and every particle or keystream. This velocity vector also has multiple dimensions. The number of velocity dimension is calculated using following logic.

```
$If\ \big(lengthof(Plaintext)\big) \leq maximum\ keystream\ dimension\ then$
    $Set\ velocity\ dimension\ (n) \leq (lengthof(Plaintext))\ -\ (lengthof(keystream))$

$Else\ if\ (lengthof(Plaintext)) > maximum\ keystream\ dimension\ then$
        $Set\ velocity\ dimension\ (n)$
            $\leq\ (maximum\ keystream\ dimension)\ -\ (lengthof(keystream))$
```

The dimensions in the velocity vector can be filled or unfilled. Total number of engaged dimension in the velocity vector denotes the length of the velocity vector. Velocity vector of $Particle_i$ is denoted by $Velocity_i$ which is a set of n velocity values one for each character $Velocity_i = (Velocity\_char_1, Velocity\_char_2, \dots, Velocity\_char_n)$. Group of velocity characters form a velocity vector. In this technique maximum keystream dimension is 256. So,

```
$If\ (lengthof(Plaintext)) \leq 256\ then$
    $Set\ velocity\ dimension\ (n) \leq (lengthof(Plaintext))\ -\ (lengthof(keystream))$

$Else\ if\ (lengthof(Plaintext)) > 256\ then$
        $Set\ velocity\ dimension\ (n) \leq\ 256\ -\ (lengthof(keystream))$
```

Each particle position is evaluated by counting number of characters in the keystream belonging to a plaintext. Using equation 6.17 particle position is evaluated.

$$Particle\_Position(Particle_i) = count\ (Key_j \in Plaintext), \qquad (6.17)$$
$$where\ j\ = 1, 2, \ldots, lengthof(Particle_i)$$

Likelihood of characters in the keystream appearing in the plaintext is calculated using equation 6.18.

$$Prob(Particle_i) = Particle\_Position(Particle_i)/lengthof\ (Particle_i) \qquad (6.18)$$

$If\ (Prob(Particle_i) \geq min\ probability\ value)\ then$

$\quad return\ (Particle_i with\ Prob(Particle_i) = max\ probability\ value)$

$If\ (Prob(Particle_i) < min\ (probability\ value)\ then$

$\quad repeat\ apply\ a\ new\ velocity\ to\ displace\ the\ particle\ potion$

Each particle having an old velocity $Velocity_i$ and can be moved to a new location by applying a new velocity ($Velocity_k$) on it. The velocity applied to a particle is a number of characters in the velocity vector occurring in the plaintext and characters are chosen such a way so that these group of characters not occurring in the keystream and velocity vector. Once applying velocity on a particle the velocity characters occupy the dimension which is vacant in the velocity vector given in equation 6.19.

$$Velocity_k = \left(Velocity_{char\,1}, Velocity_{char\,2}, \ldots\ , Velocity_{char\,m}\right),$$
$$where\ Velocity_k \notin (Particle_i, Velocity_i)\ and$$
$$m\ (current\ velocity\ dimension) = \ n\ (prevoius\ velocity\ dimension)\ - lengthof(Velocity_i) \qquad (6.19)$$

The current position of a particle is found by toting up the previous position with the applied velocity given in equation 6.20.

$$Current\_position(Particle_i) = previous\_position(Particle_i)\ +\ Velocity_k \qquad (6.20)$$

The current likelihood value can be computed by dividing the particle current position with the summation of particle length and velocity vector length.

$$Prob(Particle_i) = Current\_position(Particle_i)/(lengthof(Particle_i)\ + lengthof\ (Velocity_i)) \qquad (6.21)$$

$Compute\ Velocity_i = Velocity_i \ \&\ Velocity_k$ (6.22)

$Return(Particle_i\ and\ Velocity_i\ with\ Prob(Particle_i) = max(\ prob\ value)$ (6.23)

This velocity updating phase continued

$$until\ (Prob(Particle_i) \geq max\ probability\ value)$$

In this PSI based keystream generation technique following parameters are used

- Maximum length of PSI based keystream i.e. maximum number of character represents a keystream is $L = 256$. N is the number of characters to represents keystream. Maximum value of N is L i.e. 256.
- A predefined threshold value for describing energy factor of Ant agent. This scheme used 0.75 as a threshold value.
- A predetermined value to generate the keys for the characters in the plaintext which is at a position greater than the length of the key stream. The technique uses equation 6.24 to compute the predetermined value.

$$Predetermined\_value = lengthof(plaintext)/2 \quad (6.24)$$

The figure 6.9 shows the flowchart of PSI based keystream generation and section 6.2.1.2.1 presents the complete encryption/decryption keystream generation algorithm.

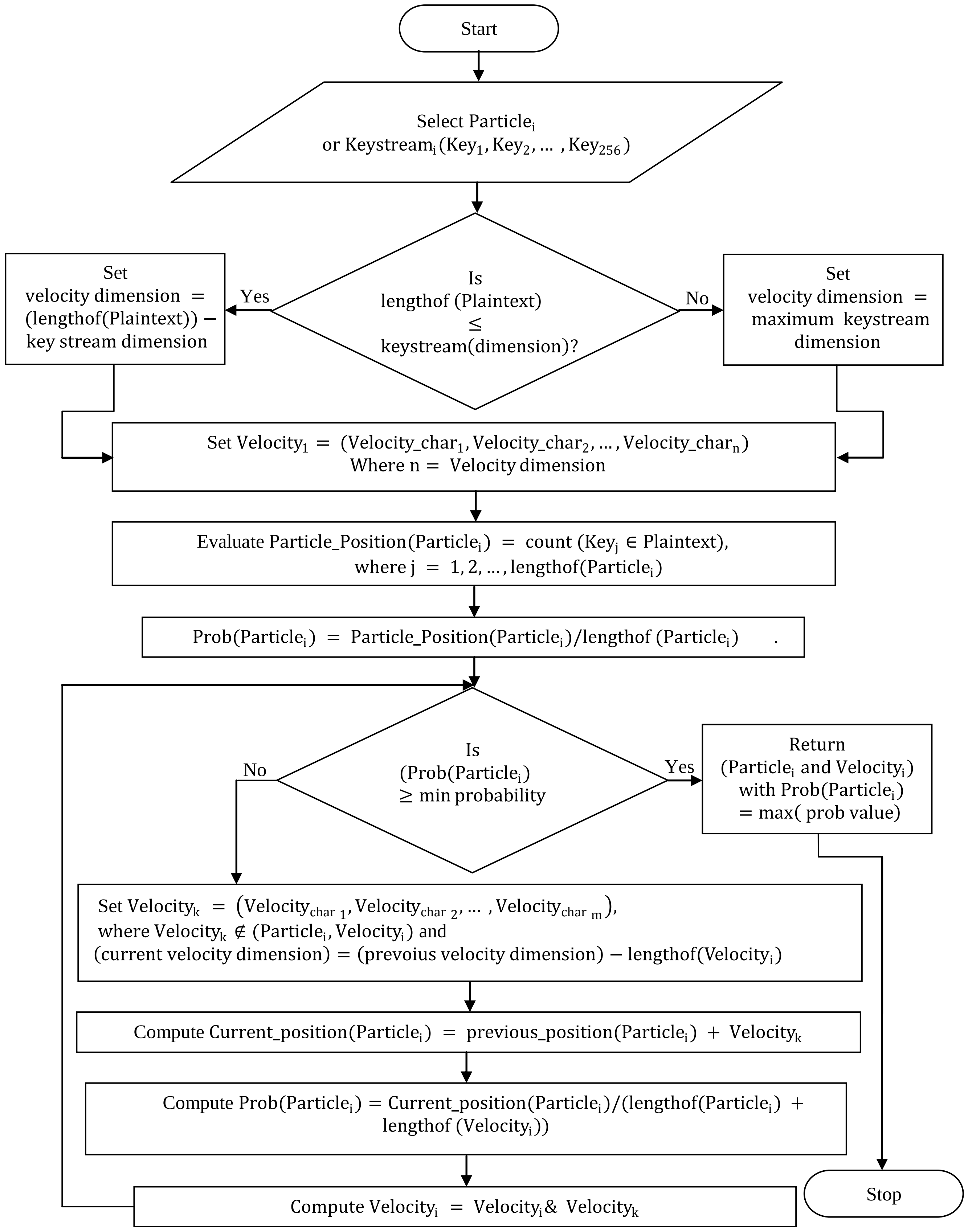

Start
Select Particle$_i$ or Keystream$_i$(Key$_1$, Key$_2$, … , Key$_{256}$)
Is lengthof (Plaintext) ≤ keystream(dimension)?
Yes
No
Set velocity dimension = (lengthof(Plaintext)) − key stream dimension
Set velocity dimension = maximum keystream dimension
Set Velocity$_1$ = (Velocity_char$_1$, Velocity_char$_2$, … , Velocity_char$_n$) Where n = Velocity dimension
Evaluate Particle_Position(Particle$_i$) = count (Key$_j$ ∈ Plaintext), where j = 1, 2, … , lengthof(Particle$_i$)
Prob(Particle$_i$) = Particle_Position(Particle$_i$)/lengthof (Particle$_i$) .
Is (Prob(Particle$_i$) ≥ min probability
No
Yes
Return (Particle$_i$ and Velocity$_i$) with Prob(Particle$_i$) = max(prob value)
Set Velocity$_k$ = (Velocity$_{char\ 1}$, Velocity$_{char\ 2}$, … , Velocity$_{char\ m}$), where Velocity$_k$ ∉ (Particle$_i$, Velocity$_i$) and (current velocity dimension) = (prevoius velocity dimension) − lengthof(Velocity$_i$)
Compute Current_position(Particle$_i$) = previous_position(Particle$_i$) + Velocity$_k$
Compute Prob(Particle$_i$) = Current_position(Particle$_i$)/(lengthof(Particle$_i$) + lengthof (Velocity$_i$))
Compute Velocity$_i$ = Velocity$_i$ & Velocity$_k$
Stop

Figure 6.9: Flow chart of Particle Swarm Intelligence (PSI) based fittest keystream generation

6.2.1.2.1 PSI based Keystream Generation Algorithm

PSI based keystream generation algorithm a threshold value is selected to weigh against velocity level of each Particle. Particle having highest energy level more than predefined threshold value is selected as a keystream.

Input : Particle with velocity

Output : PSI based keystream

Method : A threshold value is selected to weigh against velocity level of each particle. Particle having highest probability more than predefined threshold value is selected as a keystream.

Step 1. *Select particle and velocity vector for generation of keystream. Set the maximum dimension of each particle (keystream) equal to* 256. *Where*

$$Particle_i \text{ or } Keystream_i = (Key_1, Key_2, \dots, Key_{256}) \qquad (6.25)$$

$$Velocity_i = (Velocity\_char_1, Velocity\_char_2, \dots, Velocity\_char_n) \qquad (6.26)$$

$If\ (lengthof(Plaintext)) \leq 256\ then$

$$Set\ velocity\ dimension\ (n) \leq \big(lengthof(Plaintext)\big) - (lengthof(keystream)) \qquad (6.27)$$

$Else\ if\ (lengthof(Plaintext)) > 256\ then$

$$Set\ velocity\ dimension\ (n) \leq 256 - (lengthof(keystream)) \qquad (6.28)$$

Step 2. *Evaluate particle position using following function*

$$Particle\_Position(Particle_i) = count\ (Key_j \in Plaintext), \qquad (6.29)$$

$$where\ j = 1, 2, \dots, lengthof(Particle_i)$$

Evaluate the probability value using following equation

$$Prob(Particle_i) = Particle\_Position(Particle_i)/lengthof\ (Particle_i) \qquad (6.30)$$

Step 3. $If\ (Prob(Particle_i) \geq min\ probability\ value)\ then$

$$return\ (Particle_i with\ Prob(Particle_i) = max\ probability\ value \qquad (6.31)$$

Step 4. $While\ (Prob(Particle_i) < \ min\,probability\ value)\ do$

$repeat\ apply\ a\ \ Velocity_k$

$$Velocity_k \ = \left(Velocity_{char\,1}, Velocity_{char\,2}, \ldots\ , Velocity_{char\,m}\right), \tag{6.32}$$

$where\ Velocity_k \notin (Particle_i, Velocity_i\ )\ and$

$$m\ (current\ velocity\ dimension) = n\ (prevoius\ velocity\ dimension) - lengthof\,(Velocity_i) \tag{6.33}$$

$$Current\_position(Particle_i) = previous\_position(Particle_i)\ + Velocity_k \tag{6.34}$$

$$Prob(Particle_i) = Current\_position(Particle_i)/(lengthof\,(Particle_i) + lengthof\ \ (Velocity_i)) \tag{6.35}$$

$$Compute\ Velocity_i = Velocity_i\ \&\ Velocity_k \tag{6.36}$$

$until\ \left(Prob(Particle_i) \geq max(\ prob\ value)\right)$

$Return$

$$(Particle_i\ and\ Velocity_i\ with\ Prob(Particle_i) = max(prob\ value) \tag{6.37}$$

Step 5. *If the length of the plaintext is grater than the length of the keystream then the values of the keystream are added to a predetermined value to generate the keys for the characters in the plaintext which is at a position grater than the length of the ke stream.*

The PSI based fittest keystream is used to perform the encryption operation on the plaintext. The detail step of PSI based encryption process is given in section 6.2.1.3.

6.2.1.3 Encryption Algorithm

Input : *Source file/source stream i.e. plaintext*

Output : *Encrypted file/encrypted stream i.e. cipher text*

Method : *The process operates on binary stream and generates encrypted bit stream through Particle Swarm Intelligence (PSI) based encryption.*

Step 1. *If the length of the plaintext is grater than the length of the PSI based keystream then the values of the keystream are added to a predetermined value to generate the keys for the characters in the plaintext which is at a position grater than the length of the keystream. Predetermined value is calculated using the equation 6.38.*

$$Predetermined\_value = lengthof(plaintext)/2 \qquad (6.38)$$

Step 2. *For the very first plaintext block keys are form by the values of the characters in the PSI based keystream.*

Step 3. *For the successive plaintext blocks PSI based keys are generated by adding predetermined value with the keys of the previous block given in equation 6.39 for reducing the key storage load that in turn reduces the space complexity.*

$$Keyforblock(i) = Keyforblock(i-1) + \ Predetermined\, value,$$
$$where\ i >= 2 \qquad (6.39)$$

Step 4. *Perform Exclusive-OR operation between plaintext block with key in the PSI based keystream.*

Step 5. *Considers the outcomes of step 4 as a stream of finite number of bits N, and is divided into a finite number of blocks, each also containing a finite number of bits n, where $1 \leq n \leq N$. Consider the block $C = c_0^j\, c_1^j\, c_2^j\, c_3^j\, c_4^j\ \dots\ c_{n-1}^j$ having size n in the outcomes of step 4.*

Step 6. *Perform cycle formation techniques on $C = c_0^j\, c_1^j\, c_2^j\, c_3^j c_4^j\ \dots\ c_{n-1}^j$ of block of size n. In the following cases $\oplus$ is used to represents the Exclusive-OR operation. Perform the operations given in equation 6.40 to 6.43 for generating the first intermediate block $I_1 = c_0^{j+1} c_1^{j+1}\, c_2^{j+1}\, c_3^{j+1}\, c_4^{j+1}\ \dots\ c_{n-1}^{j+1}$ from C in the following way:*

$$c_{n-1}^{j+1} = c_{n-1}^{j} \quad (6.40)$$

$$c_{n-2}^{j+1} = c_{n-2}^{j} \oplus c_{n-1}^{j+1} \quad (6.41)$$

$$c_{1}^{j+1} = c_{1}^{j} \oplus c_{2}^{j+1} \quad (6.42)$$

$$c_{0}^{j+1} = c_{0}^{j} \quad (6.43)$$

This process continues for a finite number of iterations, which depends on the value of n, the source block C is regenerated. If the number of iterations required regenerating the source block is assumed to be I, then any of the intermediate block is considered as a encrypted block.

6.2.1.4 Session Key based Encryption

During final step of the technique a cascaded *Exclusive-OR* operation between CGTHLP synchronized group session key and PSI encrypted cipher text is performed to generate final encoded cipher text.

The decryption algorithm takes the cipher text as a binary stream of bits and perform first level of operation using CGTHLP generated synchronized session key to produce intermediate decrypted text. Finally, PSI generated fittest keystream based decryption is performed on the intermediate decrypted text to regenerate the plaintext. The algorithm for the complete process is given in section 6.2.2.

6.2.2 CGTHLPSCT Algorithm at Receiver

Input : Encrypted file/encrypted stream i.e. cipher text

Output : Source file/source stream i.e. plaintext

Method : The process operates on encrypted binary stream and generates decrypted bit stream through Chaos based CGTHLP guided Particle Swarm Intelligence (PSI) based decryption operations.

Step 1. Perform cascaded Exclusive-OR operation between CGTHLP based session key and cipher text.

Step 2. *Perform Particle Swarm Intelligence (PSI) based decryption on the outcomes of the step 1 to regenerate starting combination i.e. plaintext.*

Step 1 of the algorithm is discussed in section 6.2.2.1. Step 2 of the algorithm for performing Particle Swarm Intelligence (PSI) based decryption is discussed in section 6.2.2.2.

6.2.2.1 Session Key based Decryption

Initially cascaded *Exclusive-OR* operation between CGTHLP synchronized session key and cipher text is performed to produce session key decrypted text. Outcomes of this operation used as an input of PSI based decryption algorithm discussed in 6.2.2.2 to regenerate the plaintext.

In the decryption process the PSI based cipher text is divided into blocks. *Exclusive-OR* guided cycle formation based decryption is performed on each block. After that all blocks are merged together. The PSI generated keystream is use to *Exclusive-OR* with the merged blocks to regenerate the plaintext. The detail step of PSI based decryption process is given in section 6.2.2.2.

6.2.2.2 Decryption Algorithm

Input : *PSI Encrypted file/ PSI encrypted stream*

Output : *Source file/source stream i.e. plaintext*

Method : *The process operates on PSI encrypted bit stream and regenerates the plaintext through PSI based decryption.*

Step 1. *Divide the PSI encrypted text into different blocks.*

Step 2. *Perform operation given in equation 6.44 to 6.47 upto* $(P - i)$ *steps on each block* $T = t_0^i t_1^i t_2^i t_3^i t_4^i \ldots t_{n-1}^i$ *if the total number of iterations required to complete the cycle is* P *and the* i^{th} *step is considered to be the encrypted block.*

$$t^{i}_{n-1} = t^{i-1}_{n-1} \tag{6.44}$$

$$t^{i}_{n-2} = t^{i}_{n-2} \oplus t^{i}_{n-1} \tag{6.45}$$

$$t^{i}_{1} = t^{i-1}_{1} \oplus t^{i}_{2} \tag{6.46}$$

$$t^{i}_{0} = t^{i-1}_{0} \tag{6.47}$$

Step 3. *Merge outcomes of step 2.*

Step 4. *Compute the predetermined value.*

Step 5. *Using predetermined value and keys in the PSI based keystream receiver generates the keys for the portion of the text exceeding the length of the PSI based keystream.*

Step 6. *Generate plaintext by performing Exclusive-OR operation between outcomes of step 3 and PSI based keystream.*

6.3 Implementation

Consider the text to be encrypted is “**softcomputing**”. The minimum probability value is assumed to be 0.75. Each particle comprises of characters representing the particle keystream. The position of the particle is computed by counting the number of characters in the particle key stream occurring in the plaintext. The probability value is found by dividing the particle position by the length of the particle keystream. If the value is less than the minimum probability value a velocity is applied to the particle to move to a new position and the position of the new particle and the probability value is found. The particle having maximum probability value greater than or equal to the minimum probability value in the iteration is the solution. The corresponding particle keystream and velocity keystream are concatenated which forms the keystream for encryption. Table 6.3 shows the process of obtaining the keystream using PSI based approach. A group of particles denoting the key stream are taken. In this the first particle has a particle keystream “**hcv**”. Since one character in the particle keystream occurs in the plaintext to be encrypted the position of the particle is one. The probability value of the particle is found to be 0.33 which is less than the minimum probability value. Thus a velocity containing one character “**gm**” is given to the particle to move the particle to a new position. Since the character in the group denoting the velocity occurs in the plaintext, the velocity is found to be two. This is added to the old position of the particle and the new position of the particle is found to have a value of three. The characters in the group denoting the velocity occupy the dimensions in the velocity keystream. The probability value is found to be 0.6 by dividing the new position by the sum of particle keystream length and the velocity keystream length. Since this is also lesser than the minimum probability value a velocity is again given to the particle and the process is repeated and the probability value is found to be 0.75 which is equal than the minimum probability value. This procedure is repeated for other particles in the group. Since the first particle in iteration three has the maximum probability value 0.75 which is greater than the minimum probability value the particle keystream "**hcv**" and the velocity keystream "**gmtof**" corresponding to that particle are concatenated to form the keystream "**hcvgmtof**" chosen for encryption. Each character in the keystream is chosen as the key for encryption. The keys used for encryption looks like a series of random numbers. Using this method the keys cannot be cracked since the keys depends on the characters in the plaintext and a random stream generator is not used for key generation.

Table 6.3
PSI based keystream generation

Particle Keystream	**Position**	**Probability Value**	**Velocity**	**New Position**	**Velocity Keystream**	**Probability Value**	**Velocity**	**New Position**	**Velocity Keystream**	**Probability Value**
hcv	1	0.33	gm−2	3	gm	0.6	tof−3	6	gmtof	0.75
rbzlsy	1	0.16	pcu−3	4	pcu	0.44	ma−1	5	pcuma	0.45
csegdx	3	0.5	jb−0	3	jb	0.37	pm−2	5	jbpm	0.50
ecg	2	0.66	$uhv - 1$	3	uhv	0.50	gre−1	4	uhvgre	0.44
Maximum Probability Value		**0.66**				**0.6**				**0.75**

Consider the plaintext to be encrypted is "**softcomputing**", binary representation of the ASCII value of plaintext is
01110011/01101111/01100110/01110100/01100011/01101111/01101101/01110000/01110101/01110100/01101001/01101110/01100111

Binary representations of ASCII value of the plaintext are divided into variable size segments. Following are the different segments constructed from S.

S_1 = 0111001101101111 (16 bits)

S_2 = 0110011001110100 (16 bits)

S_3 = 0110001101101111 (16 bits)

S_4 =0110110101110000 (16 bits)

S_5 =0111010101110100 (16 bits)

S_6 = 01101001 (8 bits)

S_7 = 0110111001100111 (16 bits)

For each of the segments, an arbitrary intermediate segment, is considered as the encrypted segment.

The formation of cycles for segments (0111001101101111) is shown below. After 16 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (0101001111100011) after iteration-10 considered as an encrypted segment for the segment S_1.

$0111001101101111 \rightarrow 0101000100100101^{1} \rightarrow 0011000011100011^{2} \rightarrow 0110111110100001^{3} \rightarrow 0101101010011111^{4} \rightarrow 0011011001110101^{5} \rightarrow 0110110111010011^{6} \rightarrow 0010010010110001^{7} \rightarrow 0001110001101111^{8} \rightarrow 0111010000100101^{9} \rightarrow 0101001111100011^{10} \rightarrow 0100111010100001^{11} \rightarrow 0100010110011111^{12} \rightarrow 0100001101110101^{13} \rightarrow 0011111011010011^{14} \rightarrow 0001010110110001^{15} \rightarrow 0111001101101111^{16}$

The formation of cycles for segments S_2 (0110011001110100) is shown below. After 16 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (0001001001110100) after iteration-8 considered as an encrypted segment for the segment S_2.

$0110011001110100 \rightarrow 0010001000101100^{1} \rightarrow 0110000111100100^{2} \rightarrow 0101111101011100^{3} \rightarrow 0011010100110100^{4} \rightarrow 0110110011101100^{5} \rightarrow 0101101110100100^{6} \rightarrow 0011011010011100^{7} \rightarrow 0001001001110100^{8} \rightarrow 0000111000101100^{9} \rightarrow 0000010111100100^{10} \rightarrow 0000001101011100^{11} \rightarrow 0000000100110100^{12} \rightarrow 0000000011101100^{13} \rightarrow 0111111110100100^{14} \rightarrow 0010101010011100^{15} \rightarrow 0110011001110100^{16}$

The formation of cycles for segments S_3 (0110001101101111) is shown below. After 16 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (0101010110011111) after iteration-12 considered as an encrypted segment for the segment S_3.

0110001101101111→0010000100100101^{1}→0110000011100011^{2}→0101111110100001^{3}→
0100101010011111^{4}→0100011001110101^{5}→0011110111010011^{6}→0001010010110001^{7}→
0000110001101111^{8}→0000010000100101^{9}→0000001111100011^{10}→0111111010100001^{11}
→0101010110011111^{12}→0011001101110101^{13}→0110111011010011^{14}→
0010010110110001^{15}→0110001101101111^{16}

The formation of cycles for segments S_4 (0110110101110000) is shown below. After 16 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (0100101001110000) after iteration-4 considered as an encrypted segment for the segment S_4.

0110110101110000→0010010011010000^{1}→0110001110110000^{2}→0101111010010000^{3}→
0100101001110000^{4}→0011100111010000^{5}→0110100010110000^{6}→0010011110010000^{7}→
0001110101110000^{8}→0111010011010000^{9}→0101001110110000^{10}→0100111010010000^{11}
→0011101001110000^{12}→0110100111010000^{13}→0101100010110000^{14}→
0011011110010000^{15}→0110110101110000^{16}

The formation of cycles for segments S_5 (0111010101110100) is shown below. After 16 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (0110011001011100) after iteration-11 considered as an encrypted segment for the segment S_5.

0111010101110100→0101001100101100^{1}→0100111011100100^{2}→0011101001011100^{3}→
0001011000110100^{4}→0000110111101100^{5}→0000010010100100^{6}→0000001110011100^{7}→
0000000101110100^{8}→0111111100101100^{9}→0010101011100100^{10}→0110011001011100^{11}
→0010001000110100^{12}→0110000111101100^{13}→0010000010100100^{14}→
0001111110011100^{15}→0111010101110100^{16}

The formation of cycles for segments S_6 (01101001) is shown below. After 8 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (00011101) after iteration-2 considered as an encrypted segment for the segment S_6.

01101001→00100111^1→00011101^2→00001011^3→01111001^4→01010111^5→01001101^6→ 00111011^7→01101001^8

The formation of cycles for segments S_7 (0110111001100111) is shown below. After 16 steps cycle is complete and the plaintext is regenerated. An arbitrary intermediate segment (0010110011111011) after iteration-6 considered as an encrypted segment for the segment S_7.

0110111001100111→0010010111011101^1→0110001101001011^2→0010000100111001^3→ 0001111100010111^4→0111010100001101^5→0010110011111011^6→0001101110101001^7→ 0000100101100111^8→0111100011011101^9→0010100001001011^{10}→0001100000111001^{11} →0000100000010111^{12}→0111100000001101^{13}→0101011111111011^{14}→ 0011001010101001^{15}→0110111001100111^{16}

On completion of the cycle formation technique on each segment seven intermediate segments are considered as the encrypted segments. On merging the above seven encrypted segments following PSI based encrypted text is generated.

01010011/11100011/00010010/01110100/01010101/10011111/01001010/01110000/011001 10/01011100/00011101/00101100/11111011

The PSI based keystream "**hcvgmtof**" has eight characters. Whereas the plaintext "**softcomputing**" has thirteen characters. So, for the extra five characters PSI based keys are generated by adding predetermined value with the keys of the previous block for reducing the key storage load that in turn reduces the space complexity. Predetermined value is calculated by the equation 6.48.

$$Predetermined\_value = lengthof(plaintext)/2 \tag{6.48}$$

So, the predetermined value will be $\left(\frac{13}{2}\right) = 6$

So binary representation of ASCII value of the PSI based keystream is

01101000/01100011/01110110/01100111/01101101/01110100/01101111/01100110/011011 10/01101001/01111100/01101101/01110011

On performing PSI keystream based encryption operation new intermediate encoded text is 00111011/10000000/01100100/00010011/00111000/11101011/00100101/00010110/00001000/00110101/01100001/01000001/10001000

For example CGTHLP based following group session key is generated 10010101/01010010/11111000/01010101/01011101/01111110/00110101/01001111/10001010/00011100/10001010/10010010/11011100

Following is the session key encrypted final cipher text produce after performing *Exclusive-OR* operation between PSI based encrypted text and CGTHLP based session key. 10101110/11010010/10011100/01000110/01100101/10010101/00011010/00101100/00010000/01101010/11000010/10000011/00010000

6.4 Security Analysis

The security of CGTHLPSCT can be analyzed by considering an attacker E with a CTHLP of identical structure to the CTHLP of parties A and B in the group, as well as with identical output generation as, can never remain synchronous with A or B having different inputs from the synchronizing parties A and B. Consider the two CTHLPs A and B and a third CTHLP of Attacker E all with identical structure. Suppose parties A and B are not synchronous at iteration t_s, i.e. $W_{ij}^{A}(t_s) \neq W_{ij}^{B}(t_s)$ for at least one component j in an arbitrary summation unit i. Let the attacker E already be synchronous to A (or B) at iteration t_s, i.e. $W_{ij}^{A}(t_s) = W_{ij}^{E}(t_s)\ \forall i, j$. Note that if the attacker is synchronous to A and B, the two parties themselves would be synchronous already. Again, two CTHLPs can only become synchronous, when all their corresponding summation units become synchronous. Assume only one component remains that is not identical at iteration t_s, i.e. $W_{ij}^{A}(t_s) \neq W_{ij}^{B}(t_s)$ for a particular component j. As inputs are considered to be different for any subsequent iteration for at least one arbitrary component j in each summation unit i, E cannot remain synchronous even if E is synchronous (by guessing e.g.) in one iteration. For different inputs, the two parties are trying to adapt completely different non-linear relations between (different) inputs $x^{A}(t) \neq x^{B}(t)$ and outputs $\tau^{A/B}(t)$. The random walks with reflecting boundaries performed by the coefficients in the iterative process now make uncorrelated moves. Even moves in the wrong direction with regard to the aim of learning common outputs are made. Two corresponding components $W_{ij}^{A}(t)$and $W_{ij}^{A}(t)$ now receive a different random component $x_{ij}^{A}(t) \neq x_{ij}^{B}(t)$ of their (differing) input vector. The distance between the components is thus no longer successively reduced to zero after each bounding operation and the two parties' coefficients remain different. Parties with identical inputs always converge to the dynamic common trajectory. Parties with differing always diverge. Partner A and B have the advantage over an attacker E in because only A and B have the Chaos synchronized seed values for generating identical input vector. The following standard attacks are considered to ensure the robustness of the CGTHLPSCT.

- *Cipher text only Attack:* The technique nullifies the success rate of this attack by producing a robust Chaos based Grouped session key and ACI based encrypted cipher text. The strength of resisting exhaustive key search attack relies on a large key space. The cryptanalyst has only the cipher text to work with. In this PSI technique the key is changed for each character of the plaintext to produce a cipher text that is mathematically difficult to break. Since 256 characters are taken and a permutation of these characters is done to get groups of characters of all possible orderings without any repetition forming the key stream, the total number of key streams will be $256! \times 2.718$. Thus the possible number of combinations to be searched is $256! \times 2.718$. Thus a hacker has to try all such key streams to find an appropriate one. This method makes it difficult for the hacker to find out the key stream used for encryption. Thus the size of the key space is $256! \times 2.718$. The technique helps to generate long period of random key streams along with no obvious relationship between the individual bits of the sequence. Also the generated keystreams are of large linear complex. Finally keystream have high degrees of correlation immunity. Thus it is practically difficult to perform a brute-force search in a key-space.
- *Known Plaintext Attack:* The plaintext is encoded using the cycle formation technique. This would increase the security in such a manner that it is difficult to know the values assigned for the characters in the plaintext. This is because there are $2^{lengt\ hofblock}$ possible combination and the hacker has to search those combinations for the values. Also the keys used for encryption has to be found by the cryptanalyst. The technique

offers better floating frequency of characters. So, known plaintext attack is difficult in this technique.

- *Chosen Plaintext Attack:* The objective of this attack is to find the secret key. This attack is difficult because there is no obvious relationship between the individual bits of the sequence in plaintext and cipher text. In the technique the cipher text is obtained by performing an *Exclusive-OR* operation between the encoded plaintext and the characters in the key stream. This technique is not vulnerable to chosen-plaintext attack, since the plaintext is encoded first using cycle generation technique then outcomes of this get *Exclusive-OR* with PSI based keystream and the outcomes of this is *Exclusive-OR* with the session key. It is difficult for the hacker to find the key chosen for encryption. So, it is difficult to choose a plaintext of his/her choice and get the corresponding cipher text. The technique passes the frequency (monobit) test, runs test, binary matrix rank test and in each session a fresh CGTHLP based session key is used for encryption which confirms that chosen plaintext attack is very difficult in this technique.
- *Chosen Cipher text Only Attack:* This technique has a good Chi-Square value this confirms good degree of non-homogeneity and also it passes the discrete Fourier transform test, approximate entropy test, overlapping (periodic) template matching test which confirms that chosen plaintext attack is difficult in this technique. So, it will be difficult get plaintext from the cipher text.
- *Brute Force Attack:* The PSI based key is changed for each character of the plaintext to produce a cipher text that is mathematically impossible to break. Since 256 characters are chosen the total number of keystreams will be $256! \times 2.718$. Thus a hacker has to try all such keystreams to find an appropriate one. This method makes it difficult for the hacker to find out the keystream used for encryption. Encryption is an important issue in wireless communication since it is carried out over the air interface, and is more vulnerable to fraud and eavesdropping. Also the keystream is used to generate the keys for the portion of the plaintext exceeding the length of the keystream. This method of encryption reduces the number of keys to be stored and distributed. Due to high complexity brute force attack will not be feasible. The technique has a good entropy

value near to eight which indicates that brute force attack is not be possible in this technique.

6.5 Discussions

The technique is very simple and easy to implement in various high level language. The test results also show that the performance and security provided by the technique is good and comparable to standard technique. The security provided by the CGTHLPSCT is comparable with other techniques. To enhance the security of the technique, CGTHLPSCT offers changes of some parameters randomly in each session. To generate the secret session key index mask get exchanged between sender and receiver. This technique has a unique ability to construct the secret key at both sides using this exchanged information. Since the encryption and decryption times are much lower, so processing speed is very high. The method takes minimum amount of resources which is greatly handle the resource constraints criteria of wireless communication. This method generates a large number of keys which is the same number of neurons in the map. For ensuring the randomness in every session, some of the parameters get change randomly at each session. CGTHLPSCT outperform than existing TPM, PPM, Diffie-Hellman Key exchange methods and does not suffers from Brute Force or Man-In-The-Middle (MITM) attack. No platform specific optimizations were done in the actual implementation, thus performance should be similar over varied implementation platform. The whole procedure is randomized, thus resulting in a unique process for a unique session, which makes it harder for a cryptanalyst to find a base to start with. This technique is applicable to ensure security in message transmission in any form and in any size in wireless communication.

Some of the salient features of CGTHLPSCT are summarized as follows:

a) *Session key generation and exchange – Identical session key can be generate after the tuning of group CTHLPs with the help of chaos synchronization. So, no need to transfer the whole session key via vulnerable public channel.*

b) *Degree of security – The technique does not suffers from cipher text only attack, known plaintext attack, chosen plaintext attack, chosen cipher text only attack, brute* *force attack and attacks during CGTHLP synchronization process. It offers authentication steps during synchronization.*

c) *Variable block size – Encryption algorithm can work with any block length and thus not require padding, which result identical size of files both in original and encrypted file. So, CGTHLPSCT has no space overhead.*

d) *Variable key –* 128/192/256 *bits CGTHLP based session key and* 128/192/256 *bits PSI based keystream with high key space can be used in different sessions. Since the session key is used only once for each transmission, so there is a minimum time stamp which expires automatically at the end of each transmission of information. Thus the cryptanalyst may not be able guess the session key for that particular session.*

e) *Complexity – The technique has the flexibility to adopt the complexity based on infrastructure, resource and energy available for computing in a node or mesh through wireless communication. So, the proposed technique may be suitable in wireless communication.*

f) *Non-homogeneity – Measures of central tendency, dispersion and Chi-Square value have been performed between the source and corresponding cipher streams generated using proposed technique. All measures indicate that the degree of non-homogeneity of the encrypted stream with respect to the source stream is good. This technique has a better Chi-Square value than technique proposed in chapter 2, 3, 4 and 5.*

g) *Floating frequency – In CGTHLPSCT it is observed that floating frequencies of encrypted characters are indicates the high degree of security of proposed technique. This technique has a better floating frequency than technique proposed in chapter 2, 3 , 4 and 5.*

h) *Entropy – The entropy of encrypted characters is near to eight which indicate the high degree of security of technique. This technique also has a better entropy value than technique proposed in chapter 2, 3 , 4 and 5.*

i) Correlation – The cipher stream generated through CGTHLPSCT is negligibly correlated with the source stream. Therefore the proposed technique may effectively resist data correlation statistical attack.

j) Key sensitivity – The technique generates an entirely different cipher stream with a small change in the key and technique totally fails to decrypt the cipher stream with a slightly different secret session key.

k) Security and performance trade-off – The technique may be ideal for trade-off between security and performance of light weight devices having very low processing capabilities or limited computing power in wireless communication.

Chapter 7

Results and Analysis

7.1 Introduction

In this chapter results of the techniques proposed in different chapters are computed on different types of files with extensive analysis. The comparative study among proposed techniques, Tree Parity Machine (TPM) and Permutation Parity Machine (PPM), RSA, Triple-DES (168 bits), AES (128 bits), RC4 and Vernam Cipher has been done based on twenty files by performing different types of experiment.

All statistical analysis of the NIST test suite have been performed to evaluate randomness of the synchronized session key proposed in different chapters. These tests focused on a variety of different types of non-randomness that could exist in a sequence. Some tests are decomposable into a variety of subtests. All fifteen tests are performed for the proposed techniques along with existing TPM and results of these tests compared and analyzed in section 7.2. Section 7.3 presented the performance comparisons among proposed and existing techniques for generation of session key through tuning. Analysis of the average time (in cycle) needed for generating variable size session key through synchronization between two machines and group of machines, memory heap used, relative time spent in GC and thread required in synchronization phase, trends of average fitness values in different number of generations, length of plan text vs. encryption/decryption key storage has been analyzed and compared with proposed and existing techniques. Twenty files each of four different types (*.dll, .exe, .txt, .doc*) with sizes varying from 1KB to 6.3MB (approx.) have been taken. Results are generated using proposed techniques, RSA, TDES (168 bits) and AES (128 bits) for all files. Using these results, comparison of encryption and decryption time presented in section 7.4. Avalanche, Strict Avalanche effects and Bit independence has been done and presented in section 7.5. Comparison based on Chi-Square values are presented in section 7.6. Nine different file types (*.dll, .com, .exe, .cpp, .txt*) with varying file sizes have been taken to perform character frequency, entropy, floating frequency and autocorrelation test in section 7.7. Section 7.8 presents the analysis based on the results.

7.2 NIST Statistical Test and Analysis

A total of fifteen statistical tests recommended in the NIST test Suite have been performed to evaluate randomness of the synchronized session key proposed in different chapters. These tests focused on a variety of different types of non-randomness that could exist in a sequence. Some tests are decomposable into a variety of subtests. The fifteen tests are performed for the proposed and existing TPM scheme and results of these tests get compared and analyzed. The fifteen tests are following:

1. The Frequency (Monobit) Test
2. The Test for Frequency within a Block
3. The Runs Test
4. The Longest Run of Ones in a Block
5. The Binary Matrix Rank Test
6. The Discrete Fourier Transform Test
7. The Non-overlapping Template Matching Test
8. The Overlapping (Periodic) Template Matching Test
9. Maurer's "Universal Statistical" Test
10. The Linear Complexity Test
11. The Serial Test
12. The Approximate Entropy Test
13. The Cumulative Sums (Cusums) Test
14. The Random Excursions Test
15. The Random Excursions Variant Test

For analysis of the statistical test, a large number of samples of bit sequences in the key has been considered. For m samples of bit sequences obtained from the key of a technique are tested by producing one P-value, a statistical threshold value is defined using equation 7.1.

$$Threshold\ value = (1-\alpha) - 3\sqrt{\left(\frac{\alpha \times (1-\alpha)}{m}\right)} \tag{7.1}$$

In frequency (monobits) test, frequency within a block test, runs test, longest run of ones in a block test, binary matrix rank test, discrete Fourier transom test, non-overlapping (aperiodic) template matching test, overlapping (periodic) template matching test, Maurer's universal statistical test, linear complexity test, approximate entropy test, the value of significance level$(\alpha) = 0.01$. The size of m is grater than inverse of α. If $m = 300$ the $Threshold\ value = 0.972766$. This means that such a test is considered statistically successful, if at least 292 sequences out of the given 300 sequences do pass the test. For a serial and cumulative sums test producing n P-values, for the calculation of $Threshold\ value$ one should consider $(m \times n)$instead of m. With same values of α and m, the $Threshold\ value$ is 0.977814 such a test is considered statistically successful if at least 294 sequences out of the given 300 sequences do pass the test. Random excursions test producing n P-values, for the calculation of $Threshold\ value$ one should consider $(m \times n)$instead of m. With same values of α and m, the $Threshold\ value$ is 0.983907 such a test is considered statistically successful if at least 296 sequences out of the given 300 sequences do pass the test. Random excursions variant test producing n P-values, for the calculation of $Threshold\ value$ one should consider $(m \times n)$instead of m. With same values of α and m, the $Threshold\ value$ is 0.985938 such a test is considered statistically successful if at least 297 sequences out of the given 300 sequences do pass the test. A methodology has been stipulated in NIST document to calculate the P-value of P-values, where it is stated that P-values for a particular test can be considered uniformly distributed, if it's P-value of P-values≥ 0.0001.

7.2.1 Frequency (Monobits) Test

The objective of the test is to find proportion of zeroes and ones for the entire sequence which determine whether the number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence. The test assesses the closeness of the fraction of ones to ½, that is, the number of ones and zeroes in a sequence should be about the same. In this experiment expected proportion for passing the test has been set to 0.972766 using equation 7.1. Table 7.1 and 7.2 shows proportion of passing and uniformity of distribution and counting of P-values lying in the given ranges.

Table: 7.1
Proportion of passing and uniformity of distribution for frequency

Technique	**Expected Proportion**	**Observed Proportion**	**Status for Proportion of passing**	**P-value of P-values**	**Status for Uniform/ Non-uniform distribution**
TPM	0.972766	0.973333	Success	2.781309e-08	Non-uniform
KSOMSCT		0.976329	Success	2.835246e-03	Uniform
DHLPSCT		0.979437	Success	3.122711e-10	Non-uniform
CDHLPSCT		0.983333	Success	3.571386e-01	Uniform
CTHLPSCT		0.984871	Success	3.915294e-07	Non-uniform
CGTHLPSCT		0.986667	Success	4.122711e-10	Non-uniform

Table: 7.2
Counting of P-values lying in the given ranges for frequency

Technique	**0-.01**	**.01-.1**	**.1-.2**	**.2-.3**	**.3-.4**	**.4-.5**	**.5-.6**	**.6-.7**	**.7-.8**	**.8-.9**	**.9-1**
TPM	5	34	29	26	20	30	37	47	0	43	29
KSOMSCT	8	23	42	26	34	31	33	43	0	43	17
DHLPSCT	4	18	25	20	34	37	46	40	0	49	27
CDHLPSCT	6	29	41	21	25	33	38	46	0	47	20
CTHLPSCT	5	20	28	18	35	35	35	42	0	42	23
CGTHLPSCT	8	32	37	27	32	39	40	47	0	43	26

From table 7.1 and 7.2 it is seen that all proposed techniques along with existing TPM based technique passed the frequency (monobits) test successfully because observed proportion values of all the proposed techniques are grater than expected proportion value. It is also noticed that in case of proposed techniques, observed proportion for passing the test are in increasing order in the sequence of KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT. It can be concluded that all proposed techniques outperform than existing TPM based technique.

7.2.2 Test for Frequency within a Block

The focus of the test is to find the proportion of zeroes and ones within M-bit blocks. This test determine whether the frequency of ones in an M-bit block is approximately $\frac{M}{2}$. In this experiment expected proportion for passing the test has been set to 0.972766 using equation 7.1. Table 7.3 and 7.4 shows proportion of passing uniformity of distribution and counting of P-values lying in the given ranges.

Table: 7.3
Proportion of passing and uniformity of distribution for frequency within a block

Technique	**Expected Proportion**	**Observed Proportion**	**Status for Proportion of passing**	**P-value of P-values**	**Status for Uniform/ Non-uniform distribution**
TPM	0.972766	0.963333	Unsuccess	3.393663e+11	Non-uniform
KSOMSCT		0.972818	Success	3.407162e-04	Uniform
DHLPSCT		0.977942	Success	3.529802e-01	Uniform
CDHLPSCT		0.980000	Success	3.639271e-06	Non-uniform
CTHLPSCT		0.984792	Success	3.903719e-03	Uniform
CGTHLPSCT		0.990000	Success	3.949802e-01	Uniform

Table: 7.4
Counting of P-values lying in the given ranges for frequency within a block

Technique	**0-.01**	**.01-.1**	**.1-.2**	**.2-.3**	**.3-.4**	**.4-.5**	**.5-.6**	**.6-.7**	**.7-.8**	**.8-.9**	**.9-1**
TPM	6	48	44	30	35	26	30	22	16	26	17
KSOMSCT	11	44	41	52	32	25	28	19	23	12	13
DHLPSCT	3	22	27	38	32	37	25	33	35	23	25
CDHLPSCT	12	24	23	39	34	37	29	22	15	29	18
CTHLPSCT	10	39	35	48	31	26	23	34	32	17	27
CGTHLPSCT	8	47	45	51	33	29	30	31	37	21	24

From table 7.3 and 7.4 it is seen that all the proposed techniques passed the frequency within a block test successfully because observed proportion values of all the proposed techniques are grater than expected proportion value, whereas existing TPM based technique does not. is also noticed that in case of proposed techniques, observed proportion for passing the test are in increasing order in the sequence of KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT. It can be concluded that all proposed techniques outperform than existing TPM based technique.

7.2.3 Runs Test

The focus of this test is the total number of zero and one runs in the entire sequence, where a run is an uninterrupted sequence of identical bits. A run of length k means that a run consists of exactly k identical bits and is bounded before and after with a bit of the opposite value. The purpose of the runs test is to determine whether the number of runs of ones and zeros of various lengths is as expected for a random sequence. In particular, this test determines whether the oscillation between such substrings is too fast or too slow. In this experiment expected proportion for passing the test has been set to 0.972766 using equation 7.1. Table 7.5 and 7.6 shows proportion of passing and uniformity of distribution and counting of P-values lying in the given ranges.

Table: 7.5
Proportion of passing and uniformity of distribution for runs

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing	P-value of P-values	Status for Uniform/ Non-uniform distribution
TPM	0.972766	0.974275	Success	1.093862e-02	Uniform
KSOMSCT		0.975746	Success	0.321683e-01	Uniform
DHLPSCT		0.977263	Success	0.831790e-01	Uniform
CDHLPSCT		0.986997	Success	1.160128e-01	Uniform
CTHLPSCT		0.990000	Success	1.174101e-01	Uniform
CGTHLPSCT		0.993333	Success	1.191964e-01	Uniform

Table: 7.6
Counting of P-values lying in the given ranges for runs

Technique	0-.01	.01-.1	.1-.2	.2-.3	.3-.4	.4-.5	.5-.6	.6-.7	.7-.8	.8-.9	.9-1
TPM	2	31	24	28	26	35	32	41	21	38	22
KSOMSCT	3	21	46	26	27	22	36	33	28	31	27
DHLPSCT	5	16	36	23	28	44	31	33	30	23	31
CDHLPSCT	7	14	42	26	23	32	28	37	29	37	32
CTHLPSCT	4	18	29	21	27	27	30	32	25	29	30
CGTHLPSCT	6	26	33	27	29	42	36	34	31	22	32

From table 7.5 and 7.2 it is seen that all the proposed techniques along with existing TPM based technique passed the runs test successfully because observed proportion values of all the proposed techniques are grater than expected proportion value. In case of proposed techniques, observed proportion for passing the test are in increasing order in the sequence of KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and all the proposed techniques outperform than existing TPM based technique.

7.2.4 Longest Run of Ones in a Block

This test finds the longest run of ones within M-bit blocks. The purpose of this test is to determine whether the length of the longest run of ones within the tested sequence is consistent with the length of the longest run of ones that would be expected in a random sequence. Note that an irregularity in the expected length of the longest run of ones implies that there is also an irregularity in the expected length of the longest run of zeroes. Long runs of zeroes were not evaluated separately due to a concern about statistical independence among the tests. Expected proportion for passing the test has been set to 0.972766 using equation 7.1.Table 7.7 and 7.8 shows proportion of passing and uniformity of distribution and counting of P-values lying in the given ranges.

Table: 7.7
Proportion of passing and uniformity of distribution for longest run of ones in a block

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing	P-value of P-values	Status for Uniform/ Non-uniform distribution
TPM	0.972766	0.986667	Success	2.197745e-02	Uniform
KSOMSCT		0.97051	Unsuccess	1.491737e+02	Uniform
DHLPSCT		0.988026	Success	2.351830e-02	Uniform
CDHLPSCT		0.990000	Success	2.749211e-03	Uniform
CTHLPSCT		0.993174	Success	2.896945e-03	Uniform
CGTHLPSCT		0.996667	Success	3.100264e-02	Uniform

Table: 7.8
Counting of P-values lying in the given ranges for longest run of ones in a block

Technique	0-.01	.01-.1	.1-.2	.2-.3	.3-.4	.4-.5	.5-.6	.6-.7	.7-.8	.8-.9	.9-1
TPM	4	30	33	37	33	17	42	31	21	32	20
KSOMSCT	3	18	42	23	36	20	39	18	38	35	28
DHLPSCT	1	18	27	35	24	28	43	24	43	31	26
CDHLPSCT	3	19	37	31	35	19	39	28	24	33	27
CTHLPSCT	2	27	29	25	33	29	41	17	26	31	25
CGTHLPSCT	3	17	35	34	28	27	38	24	37	33	21

From table 7.7 and 7.8 it is seen that the proposed techniques except KSOMSCT passed the longest run of ones in a block test successfully because observed proportion values of all the proposed techniques are grater than expected proportion value. Existing TPM based technique has also passed test. In case of proposed techniques, observed proportion for passing the test are in increasing order in the sequence of KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT.

7.2.5 Binary Matrix Rank Test

The purpose of this test is to check for linear dependence among fixed length substrings of the original sequence. In this experiment expected proportion for passing the test has been set to 0.972766 using equation 7.1. Table 7.9 and 7.10 shows proportion of passing and uniformity of distribution and counting of P-values lying in the given ranges.

Table: 7.9
Proportion of passing and uniformity of distribution for binary matrix rank test

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing	P-value of P-values	Status for Uniform/ Non-uniform distribution
TPM	0.972766	0.970173	Unsuccess	7.186328e+03	Uniform
KSOMSCT		0.990000	Success	7.491904e-02	Uniform
DHLPSCT		0.992619	Success	7.571843e-02	Uniform
CDHLPSCT		0.993333	Success	7.194751e-01	Uniform
CTHLPSCT		0.995493	Success	8.281049e-01	Uniform
CGTHLPSCT		0.996667	Success	8.378459e-02	Uniform

Table: 7.10
Counting of P-values lying in the given ranges for binary matrix rank test

Technique	0-.01	.01-.1	.1-.2	.2-.3	.3-.4	.4-.5	.5-.6	.6-.7	.7-.8	.8-.9	.9-1
TPM	3	24	42	24	27	40	20	37	28	29	26
KSOMSCT	2	27	33	31	27	41	30	28	29	24	28
DHLPSCT	1	26	22	32	29	32	37	44	31	19	27
CDHLPSCT	1	23	37	31	28	34	28	41	32	29	28
CTHLPSCT	3	26	29	31	27	39	20	30	29	19	26
CGTHLPSCT	1	28	25	27	26	31	27	42	36	27	29

From table 7.9 and 7.10 it is seen that all the proposed techniques passed the binary matrix rank test successfully because observed proportion values of all the proposed techniques are grater than expected proportion value. But existing TPM based technique does not. In case of proposed techniques, observed proportion for passing the test are in increasing order in the sequence of KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT. This confirms that one is outperform than other. It can be concluded that all the proposed techniques outperform than existing TPM based technique.

7.2.6 Discrete Fourier Transform Test

The purpose of this test is to detect periodic features (i.e., repetitive patterns that are near each other) in the tested sequence that would indicate a deviation from the assumption of randomness. In this experiment expected proportion for passing the test has been set to 0.972766 using equation 7.1. Table 7.11 and 7.12 shows proportion of passing and uniformity of distribution and counting of P-values lying in the given ranges.

Table: 7.11
Proportion of passing and uniformity of distribution for discrete Fourier transform test

Technique	**Expected Proportion**	**Observed Proportion**	**Status for Proportion of passing**	**P-value of P-values**	**Status for Uniform/ Non-uniform distribution**
TPM	0.972766	0.951467	Unsuccess	0.000000e+00	Non-uniform
KSOMSCT		0.968329	Unsuccess	0.000000e+00	Non-uniform
DHLPSCT		1.000000	Success	0.000000e+00	Non-uniform
CDHLPSCT		1.000000	Success	0.000000e+00	Non-uniform
CTHLPSCT		1.000000	Success	0.000000e+00	Non-uniform
CGTHLPSCT		1.000000	Success	0.000000e+00	Non-uniform

Table: 7.12
Counting of P-values lying in the given ranges for discrete Fourier transform test

Technique	**0-.01**	**.01-.1**	**.1-.2**	**.2-.3**	**.3-.4**	**.4-.5**	**.5-.6**	**.6-.7**	**.7-.8**	**.8-.9**	**.9-1**
TPM	1	6	6	23	27	35	52	50	0	69	31
KSOMSCT	0	4	11	18	23	41	37	61	0	58	47
DHLPSCT	0	1	14	24	26	43	34	48	0	75	35
CDHLPSCT	2	2	13	24	28	44	36	52	0	57	37
CTHLPSCT	1	5	14	26	24	39	47	45	0	73	42
CGTHLPSCT	0	2	12	21	27	42	36	56	0	69	38

From table 7.11 and 7.12 it is seen that the proposed techniques except KSOMSCT passed the discrete Fourier transform test successfully because observed proportion values of all the proposed techniques are grater than expected proportion value. Also existing TPM based technique does not passed. It is also noticed that in case of proposed techniques, observed proportion for passing the test are in increasing order in the sequence of, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT.

7.2.7 Non-overlapping (Aperiodic) Template Matching Test

The purpose of this test is to reject sequences that exhibit too many occurrences of a given non-periodic (aperiodic) pattern. For this test and for the overlapping template matching test, an m-bit window is used to search for a specific m-bit pattern. If the pattern is not found, the window slides one bit position. For this test, when the pattern is found, the window is reset to the bit after the found pattern, and the search resumes. In this experiment expected proportion for passing the test has been set to 0.972766 using equation 7.1. Table 7.13 and 7.14 shows proportion of passing and uniformity of distribution and counting of P-values lying in the given ranges.

Table: 7.13
Proportion of passing and uniformity of distribution for non-overlapping (aperiodic) template matching test

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing	P-value of P-values	Status for Uniform/ Non-uniform distribution
TPM	0.972766	0.987164	Success	7.309571e-05	Non-uniform
KSOMSCT		0.988891	Success	7.895104e-03	Uniform
DHLPSCT		0.992275	Success	8.218760e-01	Uniform
CDHLPSCT		0.994872	Success	8.592518e-01	Uniform
CTHLPSCT		0.998941	Success	8.698073e-01	Uniform
CGTHLPSCT		1.000000	Success	8.812965e-01	Uniform

Table: 7.14
Counting of P-values lying in the given ranges for non-overlapping (aperiodic) template matching test

Technique	0-.01	.01-.1	.1-.2	.2-.3	.3-.4	.4-.5	.5-.6	.6-.7	.7-.8	.8-.9	.9-1
TPM	3	29	36	29	29	20	34	32	27	29	32
KSOMSCT	8	36	31	29	25	24	26	30	29	34	28
DHLPSCT	1	13	24	30	29	31	23	30	27	37	55
CDHLPSCT	3	16	36	30	27	30	27	30	29	35	29
CTHLPSCT	1	31	32	28	29	28	32	31	26	31	32
CGTHLPSCT	1	15	26	29	25	30	34	30	28	36	48

From table 7.13 and 7.14 it is seen that all the proposed techniques along with existing TPM based technique passed the non-overlapping (aperiodic) template matching test successfully because observed proportion values of all the proposed techniques are grater than expected proportion value. It is also noticed that in case of proposed techniques, observed proportion for passing the test are in increasing order in the sequence of KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT.

7.2.8 Overlapping (Periodic) Template Matching Test

The focus of this test is the number of pre-defined target substrings. The purpose of this test is to reject sequences that show deviations from the expected number of runs of ones of a given length. Note that when there is a deviation from the expected number of ones of a given length, there is also a deviation in the runs of zeroes. Runs of zeroes were not evaluated separately due to a concern about statistical independence among the tests. For this test and for the non-overlapping template matching test, an m-bit window is used to search for a specific m-bit pattern. If the pattern is not found, the window slides one bit position. For this test, when the pattern is found, the window again slides one bit, and the search is resumed. In this experiment expected proportion for passing the test has been set to 0.972766 using equation 7.1. Table 7.15 and 7.16 shows proportion of passing and uniformity of distribution and counting of P-values lying in the given ranges.

Table: 7.15
Proportion of passing and uniformity of distribution for overlapping (periodic) template matching test

Technique	**Expected Proportion**	**Observed Proportion**	**Status for Proportion of passing**	**P-value of P-values**	**Status for Uniform/ Non-uniform distribution**
TPM	0.972766	0.970173	Unsuccess	7.037531e+03	Uniform
KSOMSCT		0.980201	Success	7.259823e-02	Uniform
DHLPSCT		0.982107	Success	7.573992e-02	Uniform
CDHLPSCT		0.983932	Success	7.729034e-01	Uniform
CTHLPSCT		0.985028	Success	7.750939e-01	Uniform
CGTHLPSCT		0.985739	Success	7.890822e-02	Uniform

Table: 7.16
Counting of P-values lying in the given ranges for overlapping (periodic) template matching test

Technique	0-.01	.01-.1	.1-.2	.2-.3	.3-.4	.4-.5	.5-.6	.6-.7	.7-.8	.8-.9	.9-1
TPM	5	30	30	28	20	28	30	32	21	48	58
KSOMSCT	8	23	49	20	28	20	31	38	26	10	71
DHLPSCT	10	12	31	21	22	39	37	30	31	34	41
CDHLPSCT	14	15	42	27	29	36	22	32	22	62	29
CTHLPSCT	19	19	26	22	23	22	34	31	28	10	34
CGTHLPSCT	21	21	33	24	28	41	38	34	30	28	22

From table 7.15 and 7.16 it is seen that all the proposed techniques passed the overlapping (periodic) template matching test successfully because observed proportion values of all the proposed techniques are grater than expected proportion value. But existing TPM based technique does not. is also noticed that in case of proposed techniques, observed proportion for passing the test are in increasing order in the sequence of KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT. This confirms that one is outperform than other, also from TPM.

7.2.9 Maurer’s “Universal Statistical” Test

The purpose of the test is to detect whether or not the sequence can be significantly compressed without loss of information. An overly compressible sequence is considered to be non-random. In this experiment expected proportion for passing the test has been set to 0.972766 using equation 7.1. Table 7.17 and 7.18 shows proportion of passing and uniformity of distribution and counting of P-values lying in the given ranges.

Table: 7.17
Proportion of passing and uniformity of distribution for Maurer's "universal statistical" test

Technique	**Expected Proportion**	**Observed Proportion**	**Status for Proportion of passing**	**P-value of P-values**	**Status for Uniform/ Non-uniform distribution**
TPM	0.972766	1.000000	Success	7.828473e-01	Uniform
KSOMSCT		1.000000	Success	7.830128e-01	Uniform
DHLPSCT		1.000000	Success	7.830903e-01	Uniform
CDHLPSCT		1.000000	Success	7.831719e-01	Uniform
CTHLPSCT		1.000000	Success	7.839116e-01	Uniform
CGTHLPSCT		1.000000	Success	7.842736e-01	Uniform

Table: 7.18
Counting of P-values lying in the given ranges for Maurer's "Universal Statistical" test

Technique	**0-.01**	**.01-.1**	**.1-.2**	**.2-.3**	**.3-.4**	**.4-.5**	**.5-.6**	**.6-.7**	**.7-.8**	**.8-.9**	**.9-1**
TPM	2	31	37	37	36	32	24	21	29	29	31
KSOMSCT	7	34	31	35	32	21	23	11	22	25	33
DHLPSCT	4	27	29	33	38	24	38	34	23	38	35
CDHLPSCT	7	39	23	32	37	27	27	15	24	39	30
CTHLPSCT	2	32	34	39	36	30	25	27	26	26	32
CGTHLPSCT	1	30	31	30	3	21	32	22	21	24	32

From table 7.17 and 7.18 it is seen that all the proposed techniques along with existing TPM based technique passed the Maurer's "universal statistical" test successfully because observed proportion values of all the proposed techniques are grater than expected proportion value. It is also noticed that in case of proposed techniques, observed proportion for passing the test are in increasing order in the sequence of KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT.

7.2.10 Linear Complexity Test

The focus of this test is to find the length of a generating feedback register. The purpose of this test is to determine whether or not the sequence is complex enough to be considered random. Random sequences are characterized by a longer feedback register. A short feedback register implies non-randomness. In this experiment expected proportion for passing the test has been set to 0.972766 using equation 7.1. Table 7.19 and 7.20 shows proportion of passing and uniformity of distribution and counting of P-values lying in the given ranges.

Table: 7.19
Proportion of passing and uniformity of distribution for linear complexity test

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing	P-value of P-values	Status for Uniform/ Non-uniform distribution
TPM	0.972766	0.973333	Success	2.754287e-01	Uniform
KSOMSCT		0.975318	Success	2.945727e-01	Uniform
DHLPSCT		0.994763	Success	2.978459e-02	Uniform
CDHLPSCT		1.000000	Success	3.866280e-01	Uniform
CTHLPSCT		1.000000	Success	3.871537e-01	Uniform
CGTHLPSCT		1.000000	Success	3.873217e-02	Uniform

Table: 7.20
Counting of P-values lying in the given ranges for linear complexity test

Technique	0-.01	.01-.1	.1-.2	.2-.3	.3-.4	.4-.5	.5-.6	.6-.7	.7-.8	.8-.9	.9-1
TPM	2	23	28	28	37	40	25	29	27	38	23
KSOMSCT	5	35	24	20	33	27	33	27	37	34	25
DHLPSCT	0	32	29	31	29	28	21	28	35	46	21
CDHLPSCT	2	32	28	33	28	29	29	27	33	44	26
CTHLPSCT	4	37	29	30	31	38	35	29	38	43	25
CGTHLPSCT	0	29	26	29	32	42	24	27	31	39	20

From table 7.19 and 7.20 it is seen that all the proposed techniques along with existing TPM passed the linear complexity test successfully because observed proportion values of all the proposed techniques are grater than expected proportion value. It is also noticed that in case of proposed techniques, observed proportion for passing the test are in increasing order in the sequence of KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT. This confirms that one is outperform than other.

7.2.11 Serial Test

The focus of this test is to obtain the frequency of each and every overlapping m-bit pattern across the entire sequence. The purpose of this test is to determine whether the number of occurrences of the 2^m m-bit overlapping patterns is approximately the same as would be expected for a random sequence. The pattern can overlap. In this experiment expected proportion for passing the test has been set to 0.977814 using equation 7.1. Table 7.21 and

7.22 shows proportion of passing and uniformity of distribution and counting of P-values lying in the given ranges.

Table: 7.21
Proportion of passing and uniformity of distribution for serial test

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing	P-value of P-values	Status for Uniform/ Non-uniform distribution
TPM	0.977814	0.971333	Unsuccess	0.000000e+00	Non-uniform
KSOMSCT		0.973903	Unsuccess	0.000000e+00	Non-uniform
DHLPSCT		0.979874	Success	2.049380e-03	Uniform
CDHLPSCT		0.980850	Success	2.160305e-03	Uniform
CTHLPSCT		0.981476	Success	2.142839e-03	Uniform
CGTHLPSCT		0.991667	Success	2.187234e-03	Uniform

Table: 7.22
Counting of P-values lying in the given ranges for serial test

Technique	0-.01	.01-.1	.1-.2	.2-.3	.3-.4	.4-.5	.5-.6	.6-.7	.7-.8	.8-.9	.9-1
TPM	31	55	44	38	21	25	24	21	17	23	19
KSOMSCT	30	27	33	38	29	29	33	24	22	30	34
DHLPSCT	5	35	47	51	74	60	72	67	49	60	80
CDHLPSCT	10	46	43	36	58	57	69	64	48	62	79
CTHLPSCT	17	38	49	39	33	30	38	28	29	56	68
CGTHLPSCT	4	49	38	42	46	37	27	28	43	42	43

From table 7.21 and 7.22 it is seen that proposed techniques except KSOMSCT passed serial test successfully because observed proportion values of the proposed techniques are grater than expected proportion value. Also existing TPM based technique does not passed test. It is also noticed that in case of proposed techniques, observed proportion for passing the test are in increasing order in the sequence of, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT.

7.2.12 Approximate Entropy Test

The focus of this test is to obtain the frequency of each and every overlapping m-bit pattern. The purpose of the test is to compare the frequency of overlapping blocks of two consecutive/adjacent lengths (m and $m+1$) against the expected result for a random sequence. In this experiment expected proportion for passing the test has been set to 0.972766 using equation 7.1. Table 7.23 and 7.24 shows proportion of passing and uniformity of distribution and counting of P-values lying in the given ranges.

Table: 7.23
Proportion of passing and uniformity of distribution for approximate entropy test

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing	P-value of P-values	Status for Uniform/ Non-uniform distribution
TPM	0.972766	0.983333	Success	2.490301e-01	Uniform
KSOMSCT		0.985830	Success	2.837463e-02	Uniform
DHLPSCT		0.987328	Success	2.977321e-01	Uniform
CDHLPSCT		0.991739	Success	3.219583e-01	Uniform
CTHLPSCT		0.9968372	Success	3.335839e-02	Uniform
CGTHLPSCT		0.998174	Success	3.473627e-01	Uniform

Table: 7.24
Counting of P-values lying in the given ranges for approximate entropy test

Technique	0-.01	.01-.1	.1-.2	.2-.3	.3-.4	.4-.5	.5-.6	.6-.7	.7-.8	.8-.9	.9-1
TPM	2	32	30	33	37	36	19	20	29	29	33
KSOMSCT	5	38	36	33	32	23	32	16	25	25	35
DHLPSCT	5	28	22	31	31	28	32	30	27	31	35
CDHLPSCT	4	32	28	30	35	29	25	18	28	32	32
CTHLPSCT	6	35	32	33	34	32	24	28	27	27	38
CGTHLPSCT	3	31	33	32	35	25	32	27	26	28	35

From table 7.23 and 7.24 it is seen that all proposed techniques along with existing TPM based technique passed the approximate entropy test successfully because observed proportion values of all the proposed techniques are grater than expected proportion value. It is also noticed that in case of proposed techniques, observed proportion for passing the test are in increasing order in the sequence of KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT.

7.2.13 Cumulative Sums Test

The focus of this test is the maximal excursion (from zero) of the random walk defined by the cumulative sum of adjusted $(-1,+1)$ digits in the sequence. The purpose of the test is to determine whether the cumulative sum of the partial sequences occurring in the tested sequence is too large or too small relative to the expected behavior of that cumulative sum for random sequences. This cumulative sum may be considered as a random walk. For a random sequence, the random walk should be near zero. For non-random sequences, the excursions of this random walk away from zero will be too large. In this experiment expected proportion for passing the test has been set to 0.977814 using equation 7.1. Table 7.25 and 7.26 shows proportion of passing and uniformity of distribution and counting of P-values lying in the given ranges.

Table: 7.25
Proportion of passing and uniformity of distribution for cumulative sums test

Technique	**Expected Proportion**	**Observed Proportion**	**Status for Proportion of passing**	**P-value of P-values**	**Status for Uniform/ Non-uniform distribution**
TPM	0.977814	0.953762	Unsuccess	0.000000e+00	Non-uniform
KSOMSCT		0.971289	Unsuccess	0.000000e+00	Non-uniform
DHLPSCT		0.980000	Success	1.915204e-06	Uniform
CDHLPSCT		0.987291	Success	2.336568e-04	Uniform
CTHLPSCT		0.995218	Success	2.402179e-01	Uniform
CGTHLPSCT		0.998543	Success	2.526391e-01	Uniform

Table: 7.26
Counting of P-values lying in the given ranges for cumulative Sums test

Technique	**0-.01**	**.01-.1**	**.1-.2**	**.2-.3**	**.3-.4**	**.4-.5**	**.5-.6**	**.6-.7**	**.7-.8**	**.8-.9**	**.9-1**
TPM	6	81	63	59	67	44	61	42	40	87	50
KSOMSCT	12	67	76	58	75	45	45	44	51	73	54
DHLPSCT	5	49	48	56	102	64	41	54	56	75	50
CDHLPSCT	9	75	47	53	94	62	54	49	52	83	54
CTHLPSCT	7	73	43	52	71	58	58	43	59	72	59
CGTHLPSCT	5	49	56	59	95	52	55	53	56	79	54

From table 7.25 and 7.26 it is seen that proposed techniques except KSOMSCT passed the cumulative sums test successfully because observed proportion values of the proposed techniques are grater than expected proportion value. Also existing TPM based technique does not passed test. It is also noticed that in case of proposed techniques, observed proportion for passing the test are in increasing order in the sequence of, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT.

7.2.14 Random Excursions Test

The focus of this test is to find the number of cycles having exactly K visits in a cumulative sum random walk. The cumulative sum random walk is found if partial sums of the $(0,1)$ sequence are adjusted to $(-1,+1)$. A random excursion of a random walk consists of a sequence of n steps of unit length taken at random that begin at and return to the origin. The purpose of this test is to determine if the number of visits to a state within a random walk exceeds what one would expect for a random sequence. In this experiment expected proportion for passing the test has been set to 0.983907 using equation 7.1. Table 7.27 and 7.28 shows proportion of passing and uniformity of distribution and counting of P-values lying in the given ranges.

Table: 7.27
Proportion of passing and uniformity of distribution for random excursions test

Technique	**Expected Proportion**	**Observed Proportion**	**Status for Proportion of passing**	**P-value of P-values**	**Status for Uniform/ Non-uniform distribution**
TPM	0.983907	0.935000	Unsuccess	0.000000e+00	Non-uniform
KSOMSCT		0.942500	Unsuccess	0.000000e+00	Non-uniform
DHLPSCT		0.987359	Success	2.020816e-01	Uniform
CDHLPSCT		0.993964	Success	2.090373e-01	Uniform
CTHLPSCT		0.996667	Success	2.135391e-01	Uniform
CGTHLPSCT		0.997274	Success	2.139863e-01	Uniform

Table: 7.28
Counting of P-values lying in the given ranges for random excursions test

Technique	0-.01	.01-.1	.1-.2	.2-.3	.3-.4	.4-.5	.5-.6	.6-.7	.7-.8	.8-.9	.9-1
TPM	156	275	181	187	170	185	196	200	227	245	378
KSOMSCT	138	274	169	185	167	224	200	202	235	260	346
DHLPSCT	32	246	241	223	255	226	250	244	235	220	228
CDHLPSCT	76	249	195	217	276	228	247	236	231	256	287
CTHLPSCT	80	267	231	201	287	221	243	244	233	249	321
CGTHLPSCT	94	281	223	196	265	227	257	253	235	232	236

From table 7.27 and 7.28 it is seen that the proposed techniques except KSOMSCT passed the random excursions test successfully because observed proportion values of the proposed techniques are grater than expected proportion value. Also existing TPM based technique does not passed test. It is also noticed that in case of proposed techniques, observed proportion for passing the test are in increasing order in the sequence of, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT.

7.2.15 Random Excursions Variant Test

The focus of this test is to find the number of times that a particular state occurs in a cumulative sum random walk. The purpose of this test is to detect deviations from the expected number of occurrences of various states in the random walk. In this experiment expected proportion for passing the test has been set to 0.985938 using equation 7.1. Table 7.29 and 7.30 shows proportion of passing and uniformity of distribution and counting of P-values lying in the given ranges.

Table: 7.29
Proportion of passing and uniformity of distribution for random excursions variant test

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing	P-value of P-values	Status for Uniform/ Non-uniform distribution
TPM	0.985938	0.972593	Unsuccess	0.000000e+00	Non-uniform
KSOMSCT		0.972963	Unsuccess	0.000000e+00	Non-uniform
DHLPSCT		0.986893	Success	1.370847e-01	Uniform
CDHLPSCT		0.988286	Success	1.406195e-01	Uniform
CTHLPSCT		0.989103	Success	1.429043e-01	Uniform
CGTHLPSCT		0.989928	Success	1.430975e-01	Uniform

Table: 7.30
Counting of P-values lying in the given ranges for random excursions variant test

Technique	0-.01	.01-.1	.1-.2	.2-.3	.3-.4	.4-.5	.5-.6	.6-.7	.7-.8	.8-.9	.9-1
TPM	148	296	406	481	520	598	637	640	630	586	458
KSOMSCT	146	268	349	499	596	659	628	613	617	564	461
DHLPSCT	74	435	519	495	527	544	533	578	547	578	570
CDHLPSCT	95	327	403	521	535	634	597	632	619	568	557
CTHLPSCT	75	292	511	493	587	562	604	560	553	583	498
CGTHLPSCT	81	398	428	516	526	549	571	574	569	588	571

From table 7.29 and 7.30 it is seen that the proposed technique except KSOMSCT passed the random excursions variant test successfully because observed proportion values of the proposed techniques are grater than expected proportion value. Also existing TPM based technique does not passed test. It is also noticed that in case of proposed techniques, observed proportion for passing the test are in increasing order in the sequence of, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT.

7.3 Performance Analysis

In this section performance of all the proposed and existing techniques are compared with each other in terms of average synchronization time for generation of session key and grouped session key consisting of variable bit using fixed weight range and different number of neurons in input and hidden layer, different weight range and fixed number of neurons in input and hidden layer, amount of heap used for generating 128 bit session key, amount of relative time spent in GC used for generating 128 bit session key, amount of thread required for generating 128 bit session key, number of generation vs. average fitness value in SA and GA and key storage comparisons. The comparisons and analysis of performance of proposed and existing techniques are made using following attributtes.

- Average synchronization time (in cycle) for generating $128/192/256$ bit session key among proposed and existing techniques
- Average synchronization time (in cycle) for generating $128/192/256$ bit grouped session key (Group size $= 4$) among proposed and existing techniques
- Average synchronization time (in cycle) for generating $128/192/256$ bit session key using fixed weight range ($L = 5$) and different number of neurons in input and hidden layer for the proposed DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT techniques
- Average synchronization time (in cycle) for generating $128/192/256$ bit session key using proposed techniques
- Average synchronization time (in cycle) for generating 128 bit session key using different weight range ($L = 5$ to 50) and fixed number of neurons in input and hidden layer ($2 - 4 - 2$) using DHLPSCT, CDHLPSCT techniques
- Average synchronization time (in cycle) for generating 128 bit session key using different weight range ($L = 5$ to 50) and fixed number of neurons in input and hidden layer ($2 - 2 - 3 - 2$) using CTHLPSCT, CGTHLPSCT techniques

- Average synchronization time (in cycle) for generating 128 bit session key using Hebbian learning rule with different weight range $(L = 5 \text{ to } 50)$ and fixed number of neurons in input and hidden layer $(2 - 4 - 2)$ using DHLPSCT, CDHLPSCT, techniques
- Average synchronization time (in cycle) for generating 128 bit session key using Hebbian learning rule with different weight range $(L = 5 \text{ to } 50)$ and fixed number of neurons in input and hidden layer $(2 - 2 - 3 - 2)$ using CTHLPSCT, CGTHLPSCT techniques
- Memory heap used in proposed and existing techniques for generation of 128 bit session key
- Relative time spent in GC to generate 128 bit session key using proposed and existing techniques
- Thread required to generate 128 bit session key using proposed and existing techniques
- Number of generations vs. average fitness value in Simulated Annealing based encryption/decryption key generation in DHLPSCT technique
- Number of generations vs. average fitness value in Genetic Annealing based encryption/decryption key generation in CDHLPSCT technique
- Length of plaintext vs. encryption/decryption key storage among proposed and existing techniques

7.3.1 Average Synchronization Time (in cycle) for Generating variable bit Session Key

Table: 7.31
Average synchronization time (in cycle) for generating 128 bit session key

Key Length (128 bit)	**Average Synchronization time (in cycle)**
CTHLPSCT (2-2-2-4-25)$_{NS=32}$	2302,83
CDHLPSCT (4-4-2-25)$_{NS=32}$	2397,02
DHLPSCT (4-4-2-25)$_{NS=32}$	2421,71
KSOMSCT	2516,41
TPM (L=25)	2624,27
PPM	2811,04

Here, NS denotes Network Size.

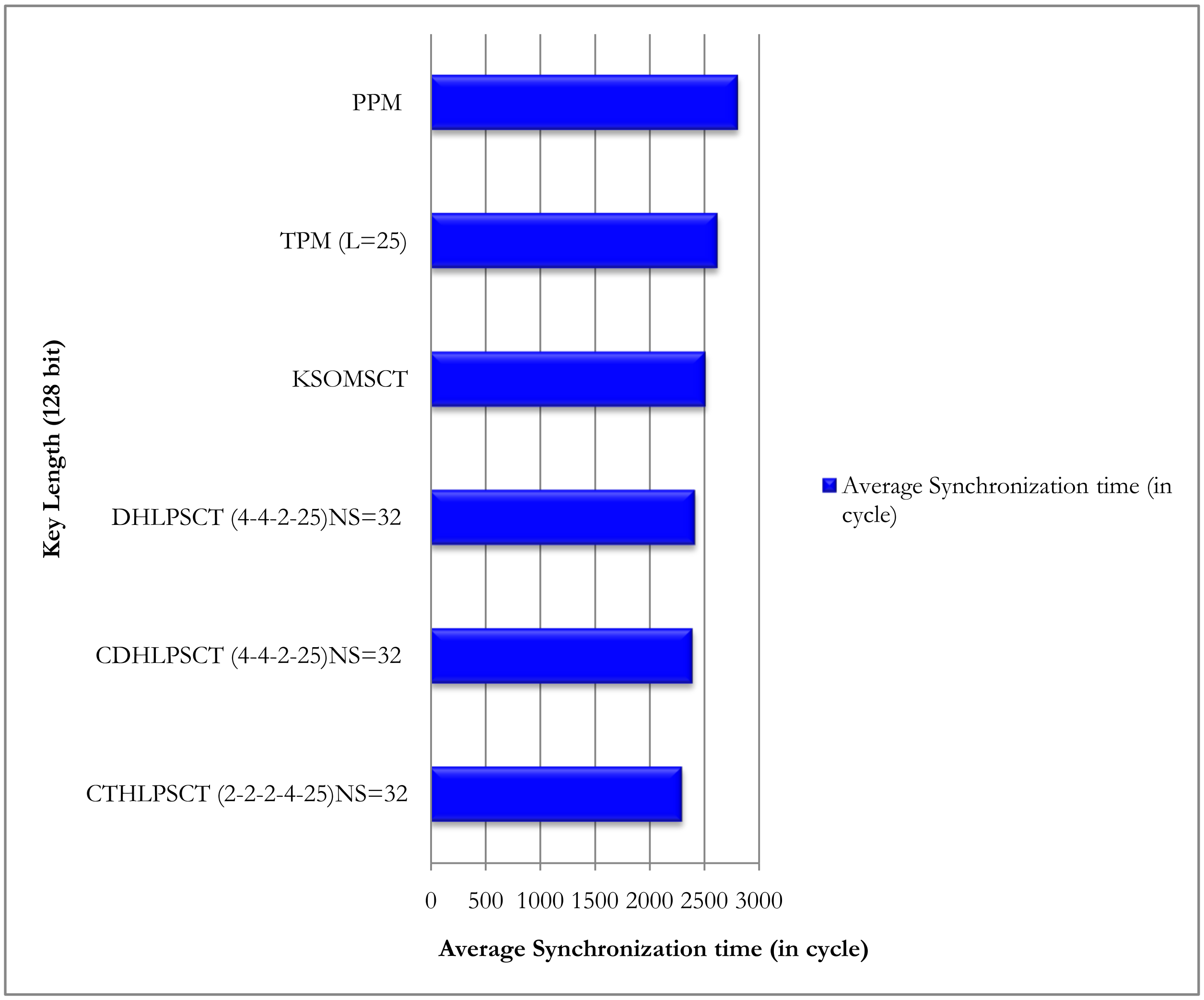


Figure 7.1: 128 bit key length vs. average synchronization time (in cycle)

Table 7.31 and figure 7.1 shows the proposed CTHLPSCT $(2-2-2-2-25)$, CDHLPSCT $(4-4-2-25)$, DHLPSCT $(4-4-2-25)$ and KSOMSCT requires $(2302{,}83), (2397{,}02), (2421{,}71), (2516{,}41)$ cycles respectively in average to generate session key having a length of 128 bit. Whereas existing TPM $(L = 25)$ and PPM needs $2624{,}27$ and $2811{,}04$ cycles respectively, which larger than all the proposed techniques. From Table and figure it has been seen that CTHLPSCT takes minimum amount of time to generate 128 bit session key compared to other proposed techniques and other two existing techniques TPM and PPM. In architecture point of view though DHLPSCT, CDHLPSCT and CTHLPSCT has the same network size i.e. 32 but in CTHLPSCT the input layer contains only two input neurons. So, the overhead of input generation through PRNG for the input layer is much lower than others. Because of this reason CTHLPSCT able to synchronize faster than others CTHLPSCT outperforms over all the proposed techniques (CDHLPSCT, DHLPSCT, and KSOMSCT) and existing TPM and PPM. This is quite affordable in terms of resources available in wireless communication.

Table: 7.32
Average synchronization time (in cycle) for generating 192 bit session key

Key Length (192 bit)	**Average Synchronization time (in cycle)**
CTHLPSCT (2-2-5-2-25)$_{NS=40}$	2463,21
CDHLPSCT (3-3-5-25)$_{NS=45}$	2789,43
DHLPSCT (3-3-5-25)$_{NS=45}$	2807,36
KSOMSCT	3173,41
TPM (L=25)	3347,15
PPM	3571,48

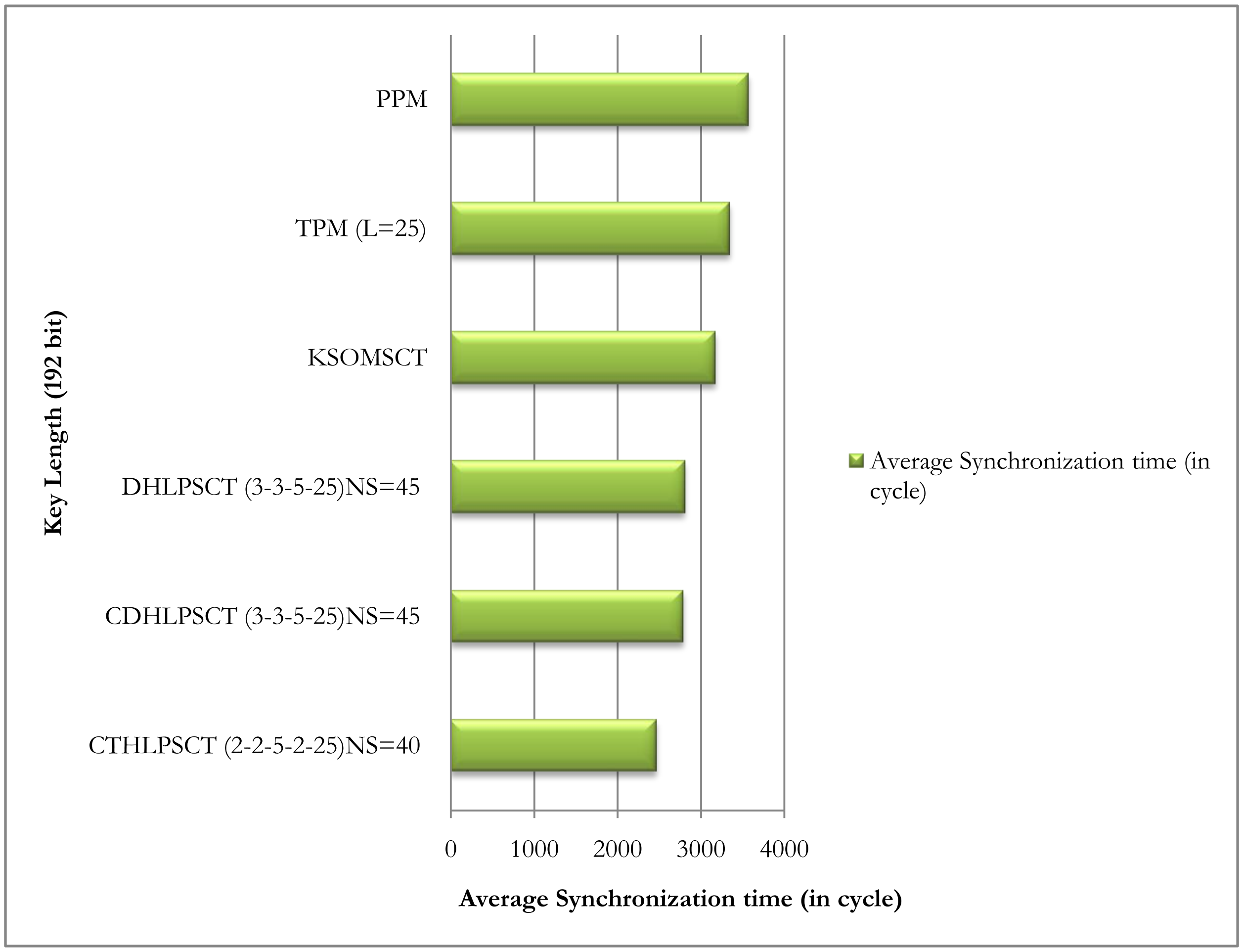


Figure 7.2: 192 bit key length vs. average synchronization time (in cycle)

Table 7.32 and figure 7.2 shows the proposed soft computing based CTHLPSCT $(2-2-5-2-25)$, CDHLPSCT $(3-3- 5-25)$, DHLPSCT $(3-3-5-25)$ and KSOMSCT needs $(2463{,}21), (2789{,}43), (2807{,}36), (3173{,}41)$ cycles respectively in average to generate session key having a length of 128 bit. Whereas existing TPM $(L = 25)$ and PPM needs $3347{,}15$ and $3571{,}48$ cycles respectively, which larger than all the proposed techniques. From Table and figure it has been seen that CTHLPSCT takes minimum amount of time to generate 192 bit session key compared to other proposed techniques and other two existing techniques TPM and PPM. In the architectural point of view, network size of CTHLPSCT is only 40 i.e. smaller than other techniques. So with smaller network size this technique can synchronize faster than others. Also it has smaller number of neurons in its input layer than others. This significantly reduces the overhead of input generations compared to others. For this reason, CTHLPSCT outperforms over all the proposed techniques (CDHLPSCT, DHLPSCT, and KSOMSCT) and existing TPM and PPM. This is quite affordable in terms of resources available in wireless communication.

Table: 7.33
Average synchronization time (in cycle) for generating 256 bit session key

Key Length (256 bit)	Average Synchronization time (in cycle)
CTHLPSCT (2-2-7-2-25)$_{NS=56}$	3682,18
CDHLPSCT (4-4-4-25)$_{NS=64}$	4208,42
DHLPSCT (4-4-4-25)$_{NS=64}$	4233,62
KSOMSCT	4719,72
TPM (L=25)	4851,86
PPM	5193,03

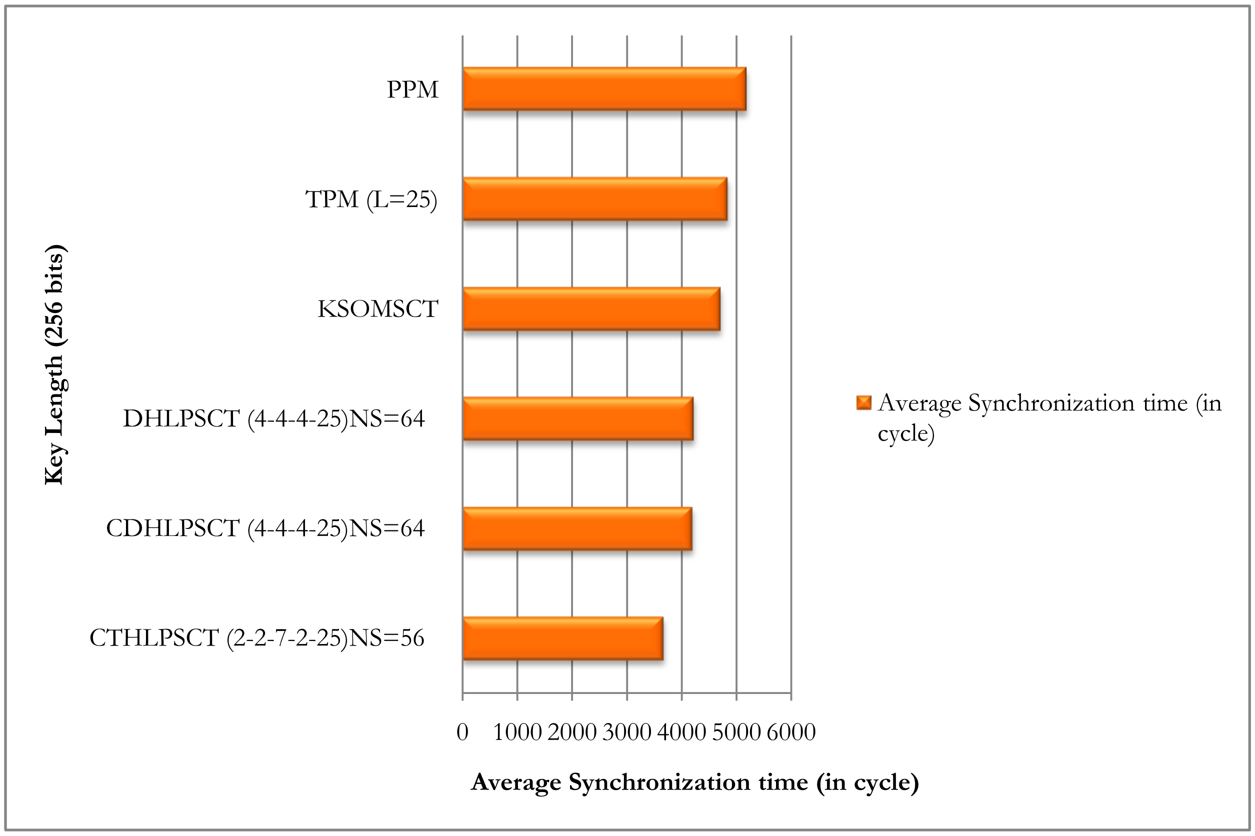


Figure 7.3: 256 bit key length vs. average synchronization time (in cycle)

Table 7.33 and figure 7.3 shows CTHLPSCT $(2-2-7-2-25)$, CDHLPSCT $(4-4-4{-}25$, DHLPSCT $4{-}4{-}4{-}25$ and KSOMSCT needs $3682{,}18$, $4208{,}42$, $4233{,}62$, $(4719{,}72)$ cycles respectively in average to generate session key having a length of 128 bit. Whereas existing TPM $(L = 25)$ and PPM needs $4851{,}86$ and $5193{,}03$ cycles respectively, which larger than all the proposed techniques. From Table and figure it has been seen that CTHLPSCT takes minimum amount of time to generate 256 bit session key compared to other proposed techniques and other two existing techniques TPM and PPM. In architecture point of view though DHLPSCT, CDHLPSCT and CTHLPSCT has the network size 64 but in CTHLPSCT the network size is only 56 also the input layer contains only two input neurons in case of CTHLPSCT. So, the overhead of input generation through PRNG for the input layer is much lower than others. Because of this reason CTHLPSCT able to synchronize faster than others CTHLPSCT outperforms over all the proposed techniques (CDHLPSCT, DHLPSCT, and KSOMSCT) and existing TPM and PPM. This is quite affordable in terms of resources available in wireless communication.

7.3.2 Average Synchronization Time (in cycle) for Generating variable bit Grouped Session (Group size = 4) Key

Table: 7.34
Average synchronization time (in cycle) for generating 128 bit grouped session (Group size = 4) key

Key Length (128 bit)	**No. of Parties participating in the Group Synchronization**	**Average Synchronization time (in cycle)**
CGTHLPSCT $(2\text{-}2\text{-}2\text{-}4\text{-}25)_{NS=32}$	4	4394,91
CTHLPSCT $(2\text{-}2\text{-}2\text{-}4\text{-}25)_{NS=32}$	4	13816,98
CDHLPSCT $(4\text{-}4\text{-}2\text{-}25)_{NS=32}$	4	14382,12
DHLPSCT $(4\text{-}4\text{-}2\text{-}25)_{NS=32}$	4	14530,26
KSOMSCT	4	15098,46
TPM (L=25)	4	15745,62
PPM	4	16866,24

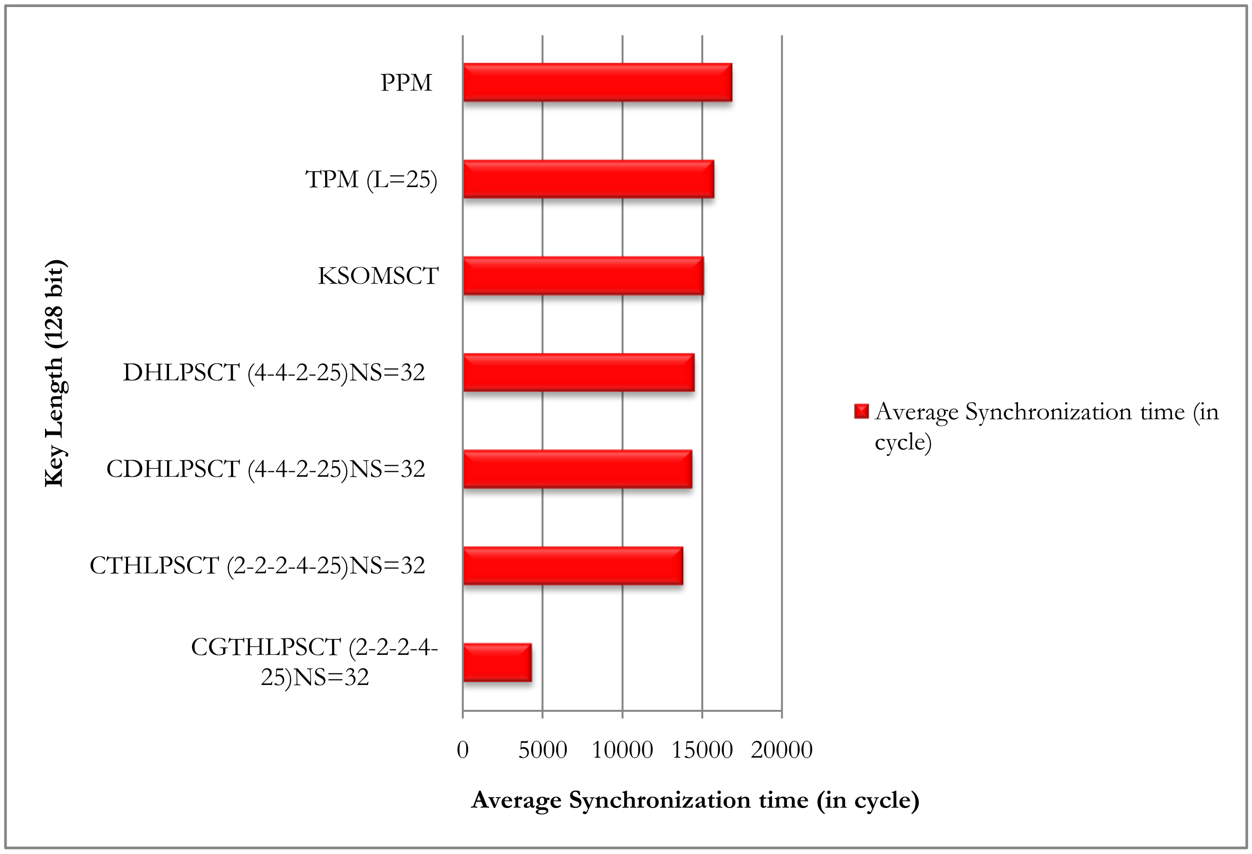


Figure 7.4: 128 bit key length vs. average synchronization time (in cycle) for grouped synchronization (Group size $= 4$)

Table 7.34 and figure 7.4 shows CGTHLPSCT $(2-2-2-4-25)$, CTHLPSCT $(2-2-2-4-25)$, CDHLPSCT $(4-4-2-25)$, DHLPSCT $(4-4-2-25)$ and KSOMSCT needs $(4394{,}91), (13816{,}98), (14382{,}12), (14530{,}26), (15098{,}46)$ cycles respectively in an average to generate session key having a length of 128 bit for synchronize group of four parties. CGTHLPSCT needs $nlog(n-1)$ number of synchronizations because this technique use complete binary tree based framework for synchronizing n parties. Whereas other proposed and existing techniques needs $\frac{n(n-1)}{2}$ number of synchronizations for synchronizing n parties. If $n=4$ then CGTHLPSCT needs only $4log(4-1)$ number of synchronizations. Whereas other techniques needs $\frac{4(4-1)}{2}=6$ synchronizations steps. This clearly indicates that proposed CGTHLPSCT outperforms than all other proposed and existing techniques at the time of group synchronization.

Table: 7.35
Average synchronization time (in cycle) for generating 192 bit grouped session (Group size = 4) key

Key Length (192 bit)	**No. of Parties participating in the Group Synchronization**	**Average Synchronization time (in cycle)**
CGTHLPSCT (2-2-5-2-25)$_{NS=40}$	4	4700,99
CTHLPSCT (2-2-5-2-25)$_{NS=40}$	4	14779,26
CDHLPSCT (3-3-5-25)$_{NS=45}$	4	16736,58
DHLPSCT (3-3-5-25)$_{NS=45}$	4	16844,16
KSOMSCT	4	19040,46
TPM (L=25)	4	20082,90
PPM	4	21428,88

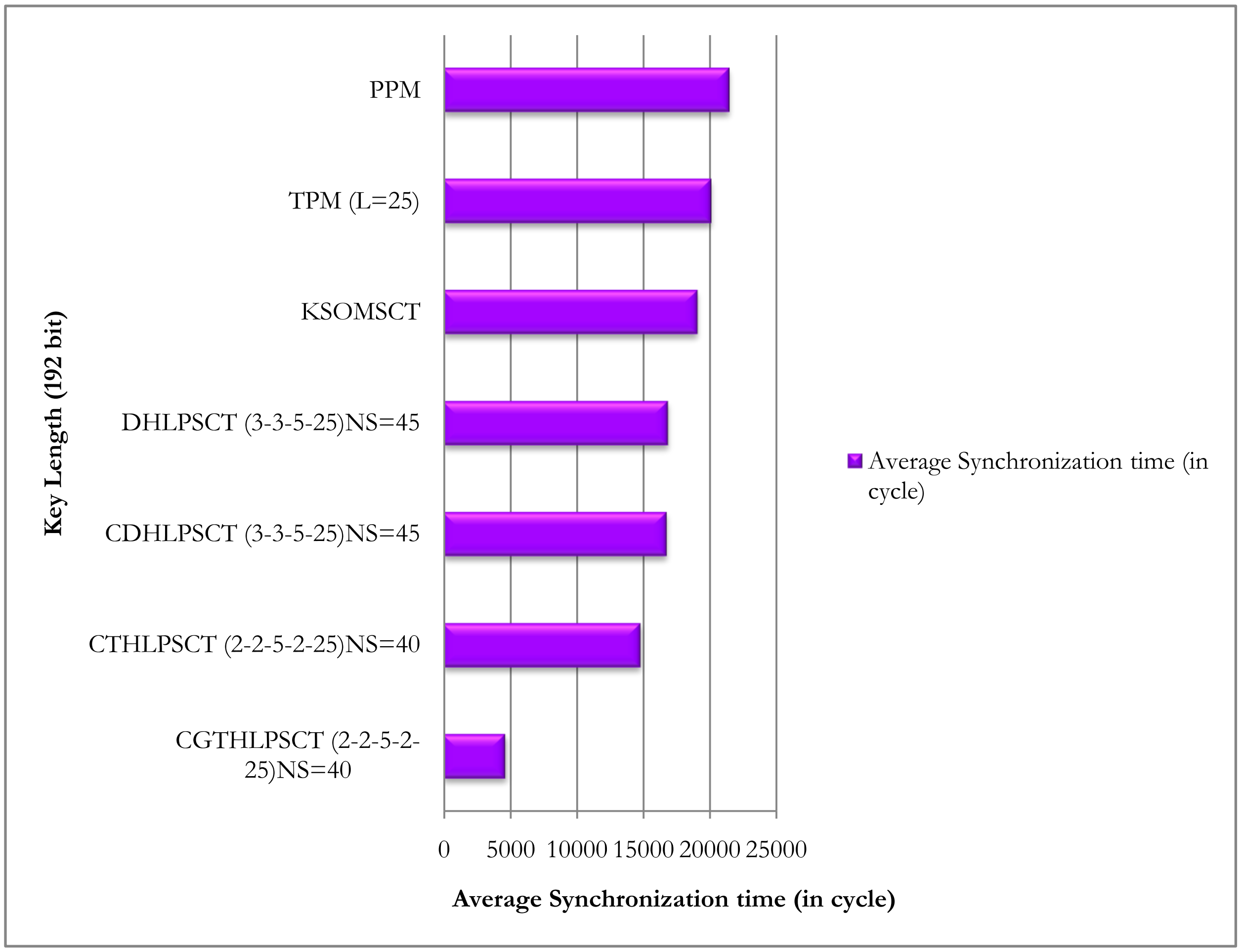


Figure 7.5: 192 bit key length vs. average synchronization time (in cycle) for grouped synchronization (Group size= 4)

Table 7.35 and figure 7.5 shows CGTHLPSCT $(2-2-5-2-25)$, CTHLPSCT $(2-2-5-2-25)$, CDHLPSCT$(3-3-5-25)$, DHLPSCT $(3-3-5-25)$ and KSOMSCT needs $(4700{,}99), (14779{,}26), (16736{,}58), (16844{,}16), (19040{,}46)$ cycles respectively in average to generate session key having a length of 192 bit for synchronize group of four parties. CGTHLPSCT needs $nlog(n-1)$ number of synchronizations because this technique use complete binary tree based framework for synchronizing n parties. Whereas other proposed and existing techniques needs $\frac{n(n-1)}{2}$ number of synchronizations for synchronizing n parties. If $n = 4$ then CGTHLPSCT needs only $4log(4-1)$ number of synchronizations. Whereas other techniques needs $\frac{4(4-1)}{2} = 6$ synchronizations steps. This clearly indicates that proposed CGTHLPSCT outperforms than all other proposed and existing techniques at the time of group synchronization.

Table: 7.36
Average synchronization time (in cycle) for generating 256 bit grouped session (Group size $=$ 4) key

Key Length (256 bit)	**No. of Parties participating in the Group Synchronization**	**Average Synchronization time (in cycle)**
CGTHLPSCT (2-2-7-2-25)$_{NS=56}$	4	7027,38
CTHLPSCT (2-2-7-2-25)$_{NS=56}$	4	22093,08
CDHLPSCT (4-4-4-25)$_{NS=64}$	4	25250,52
DHLPSCT (4-4-4-25)$_{NS=64}$	4	25401,72
KSOMSCT	4	28318,32
TPM (L=25)	4	29111,16
PPM	4	31158,18

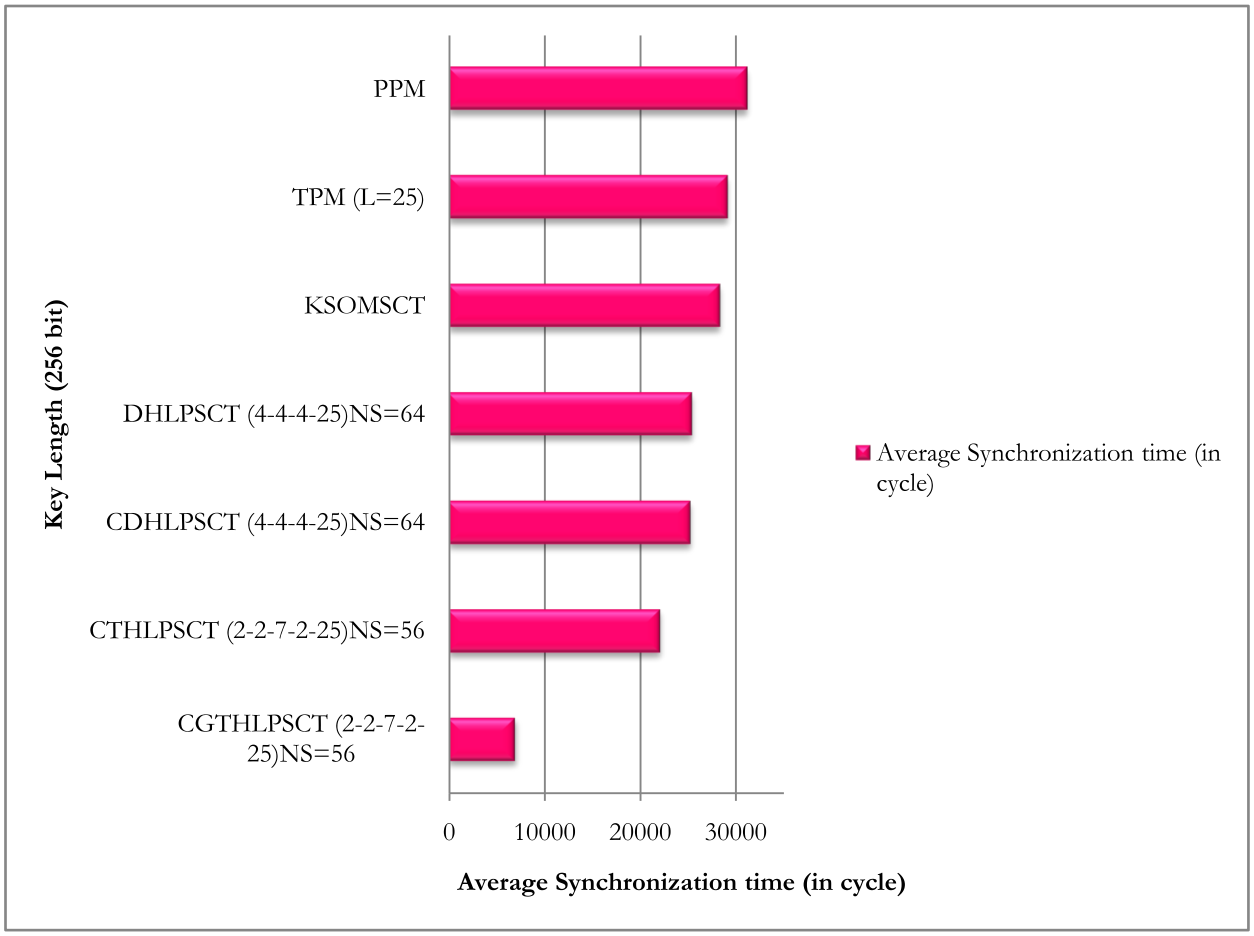


Figure 7.6: 256 bit key length vs. average synchronization time (in cycle) for grouped synchronization (Group size$=$ 4)

Table 7.36 and figure 7.6 shows CGTHLPSCT $(2-2-5-2-25)$, CTHLPSCT $(2-2-5-2-25)$, CDHLPSCT $(3-3-5-25)$, DHLPSCT $(3-3-5-25)$ and KSOMSCT needs $(7027{,}38), (22093{,}08), (25250{,}52), (25401{,}72), (28318{,}32)$ cycles respectively in average to generate session key having a length of 192 bit for synchronize group of four parties. CGTHLPSCT needs $nlog(n-1)$ number of synchronizations because this technique use complete binary tree based framework for synchronizing n parties. Whereas other proposed and existing techniques needs $\frac{n(n-1)}{2}$ number of synchronizations for synchronizing n parties. If $n = 4$ then CGTHLPSCT needs only $4log(4-1)$ number of synchronizations. Whereas other techniques needs $\frac{4(4-1)}{2} = 6$ synchronizations steps. This clearly indicates that proposed CGTHLPSCT outperforms than all other proposed and existing techniques at the time of group synchronization.

7.3.3 Average Synchronization Time (in cycle) for Generating 128 bit Session Key using fixed Weight range ($L = 5$) with variable Neurons

Table: 7.37
Generation of 128 bit session key using fixed weight range ($L = 5$) with variable neurons in DHLPSCT

DHLP Size	**N-K1-K2-L**	**Average Synchronization time in cycle**		
		Hebbian	**Anti-Hebbian**	**Random Walk**
8	1-8-1-5	112,73	139,18	148,62
16	2-4-2-5	209,94	233,36	241,49
24	2-2-6-5	306,97	329,19	348,07
24	6-2-2-5	309,49	337,45	356,32
32	4-2-4-5	410,13	432,69	451,28

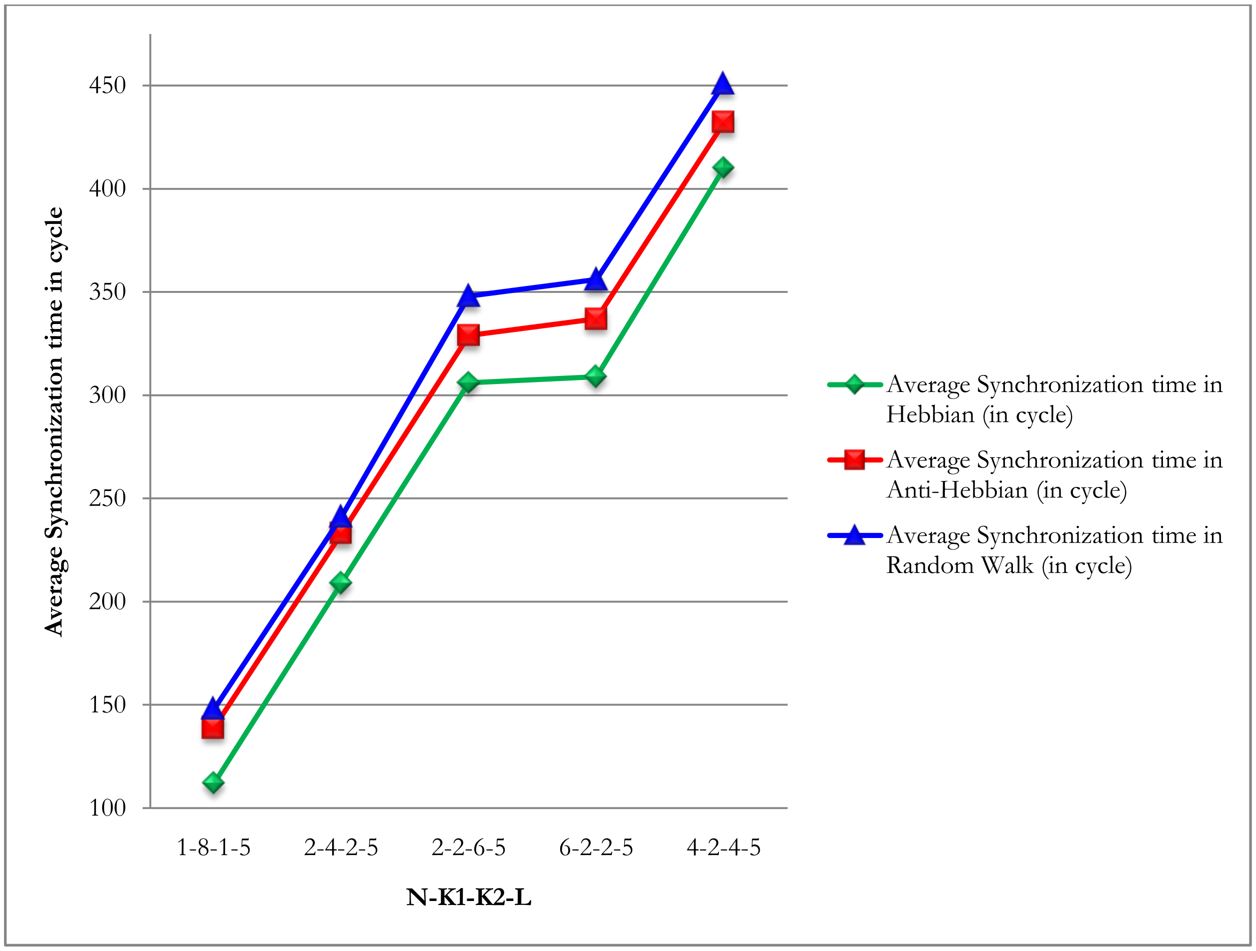


Figure 7.7: Generation of 128 bit session key using fixed weight range ($L = 5$) with variable neurons in DHLPSCT

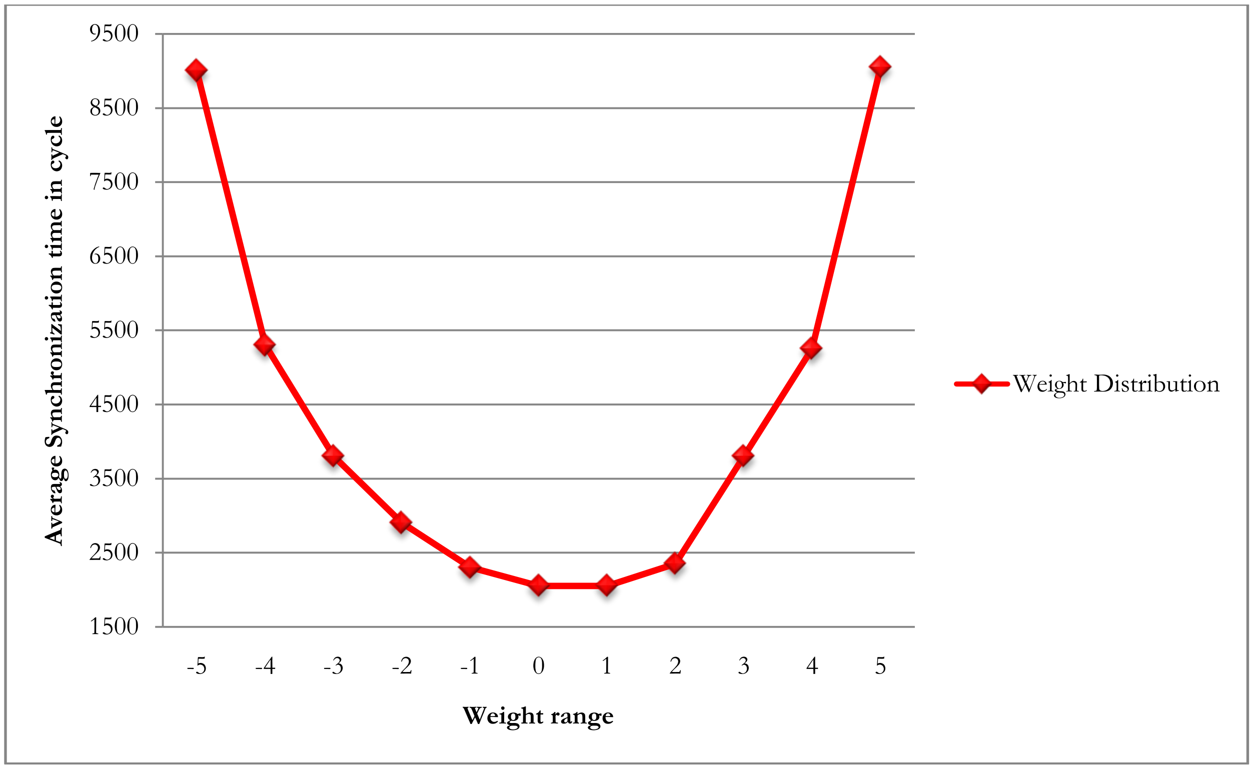


Figure 7.8: Weight distribution in Hebbian learning rule with weight range $(L) = 5$ in DHLPSCT

From the table 7.37 and figure 7.7 it has been observed that several DHLPSCT configuration (in terms of different neurons in different layers) can be use to generate 128 bit session key with fixed weight range $L = 5$. DHLPSCT size is the $N \times K1 \times K2$, where $N \times K1$ is number of input units, $K1$ is the number of hidden units in layer 1 and $K2$ is the number of hidden units in layer 2. For the first row in the table DHLPSCT size is eight where $N = 1$, $K1 = 8$, $K2 = 1$, $L = 5$. Total numbers of weights generated by the DHLPSCT are $(N \times K1 + K1 \times K2)$. Each weight decimal value can be represented in eight bit binary. So, total $((N \times K1 + K1 \times K2) \times 8)$ numbers of bits present in a weight (length of a session key). If $N = 1$, $K1 = 8$, $K2 = 1$ then $\left((1 \times 8 + 8 \times 1) \times 8\right) = 128$ bits weight value act as a session key. Among three learning rules Hebbian rules outperform over other two rules (Anti-Hebbian and Random Walk). Hebbian rules perform better where network size is comparatively small because weights are not getting well distributed in the Hebbian rules shown in figure 7.8. So, small network with 128 bit session key Hebbian makes the synchronization faster but for the small network Anti-Hebbian and Random Walk takes much more amount of synchronization time due to the weight distribution process.

Table: 7.38
Generation of 128 bit session key using fixed weight range ($L = 5$) with variable neurons in CDHLPSCT

CDHLP Size	N-K1-K2-L	Average Synchronization time in cycle		
		Hebbian	Anti-Hebbian	Random Walk
8	1-8-1-5	95,42	118,73	127,12
16	2-4-2-5	192,06	212,58	223,86
24	2-2-6-5	288,13	310,79	331,14
24	6-2-2-5	289,72	311,18	332,94
32	4-2-4-5	383,96	407,05	422,89

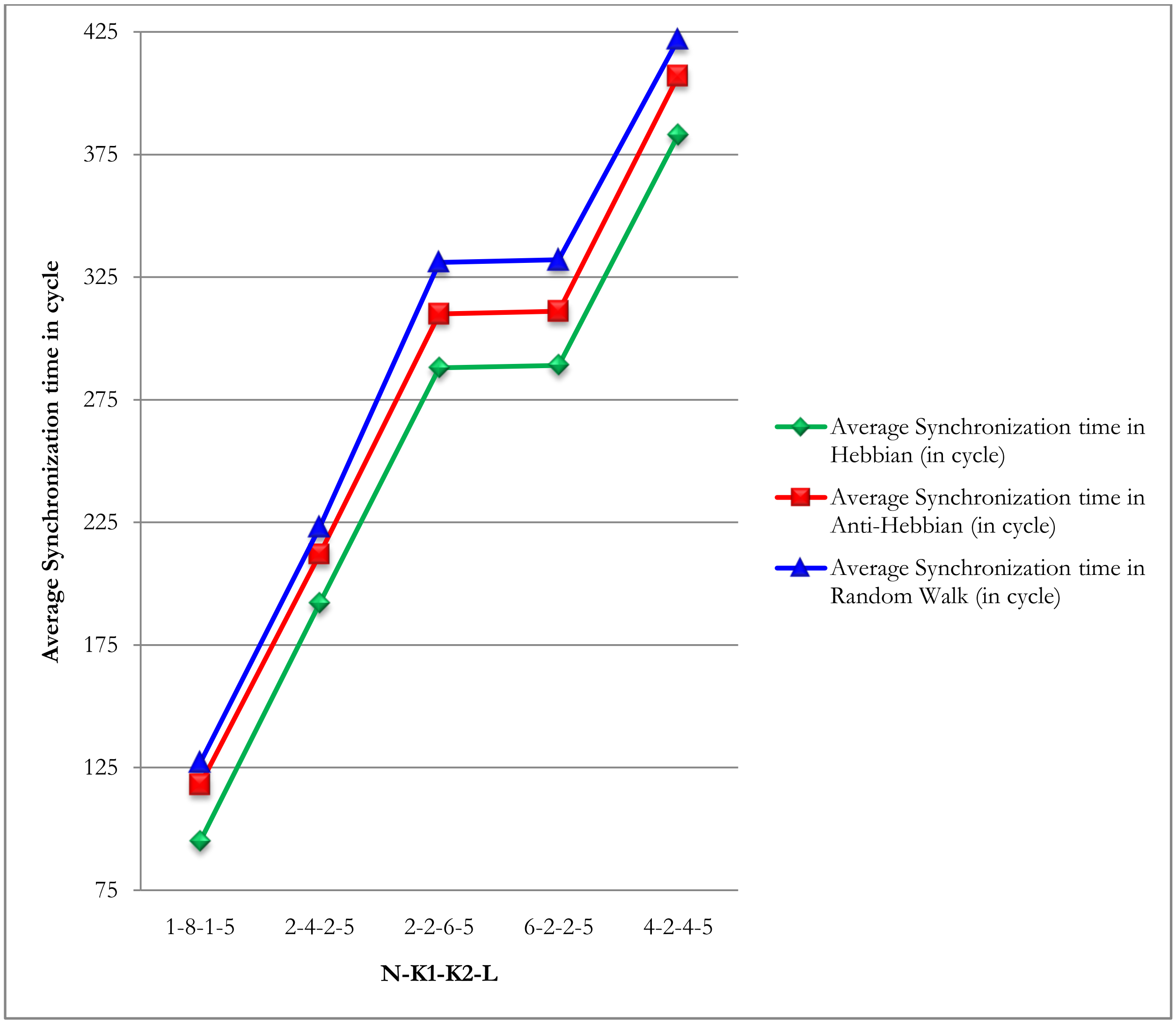


Figure 7.9: Generation of 128 bit session key using fixed weight range ($L = 5$) with variable neurons in CDHLPSCT

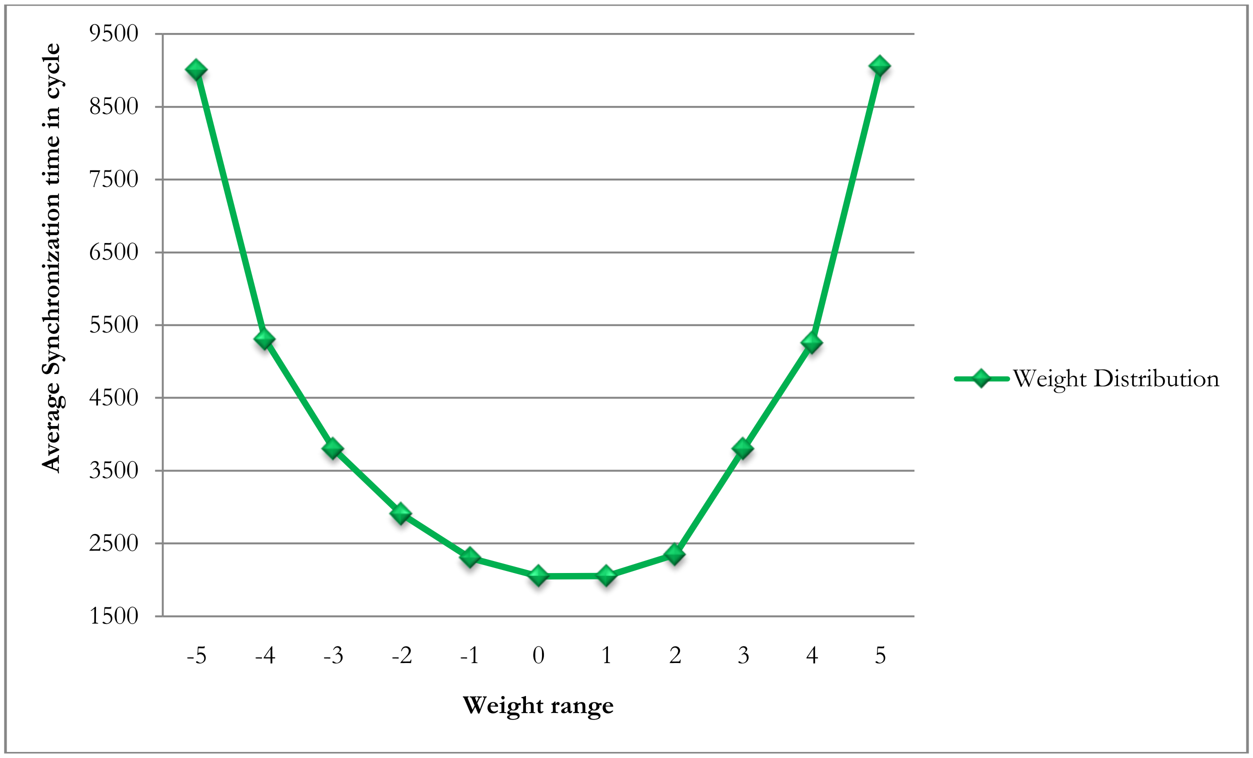


Figure 7.10: Weight distribution in Hebbian learning rule with weight range $(L) = 5$ in CDHLPSCT

From the table 7.38 and figure 7.9 it has been observed that several CDHLPSCT configuration (in terms of different neurons in different layers) can be use to generate 128 bit session key with fixed weight range $L = 5$. CDHLPSCT size is the $N \times K1 \times K2$, where $N \times K1$ is number of input units, $K1$ is the number of hidden units in layer 1 and $K2$ is the number of hidden units in layer 2. For the first row DHLPSCT size is eight, where $N = 1$, $K1 = 8$, $K2 = 1, L = 5$. Total numbers of weights generated by the DHLPSCT are $(N \times K1 + K1 \times K2)$. Each weight decimal value can be represented in eight bit binary. So, total $((N \times K1 + K1 \times K2) \times 8)$ numbers of bits present in a weight (length of a session key). If $N = 1$, $K1 = 8$, $K2 = 1$ then $\left((1 \times 8 + 8 \times 1) \times 8\right) = 128$ bits weight value act as a session key. Among three learning rules Hebbian rules outperform over other two rules (Anti-Hebbian and Random Walk). Hebbian rules perform better where network size is comparatively small because weights are not getting well distributed in the Hebbian rules shown in figure 7.10. So, small network with 128 bit session key Hebbian makes the synchronization faster but for the small network Anti-Hebbian and Random Walk takes much more amount of synchronization time due to the weight distribution process.

Table: 7.39
Generation of 128 bit session key using fixed weight range $(L = 5)$ with variable neurons in CTHLPSCT

CTHLP Size	N-K1-K2-K3-L	Average Synchronization time in cycle		
		Hebbian	Anti-Hebbian	Random Walk
24	2-2-3-2-5	287,81	309,23	320,52
24	2-3-2-2-5	288,77	310,35	323,15
32	2-2-2-4-5	382,93	406,87	421,36
32	4-2-2-2-5	383,18	406,91	421,89

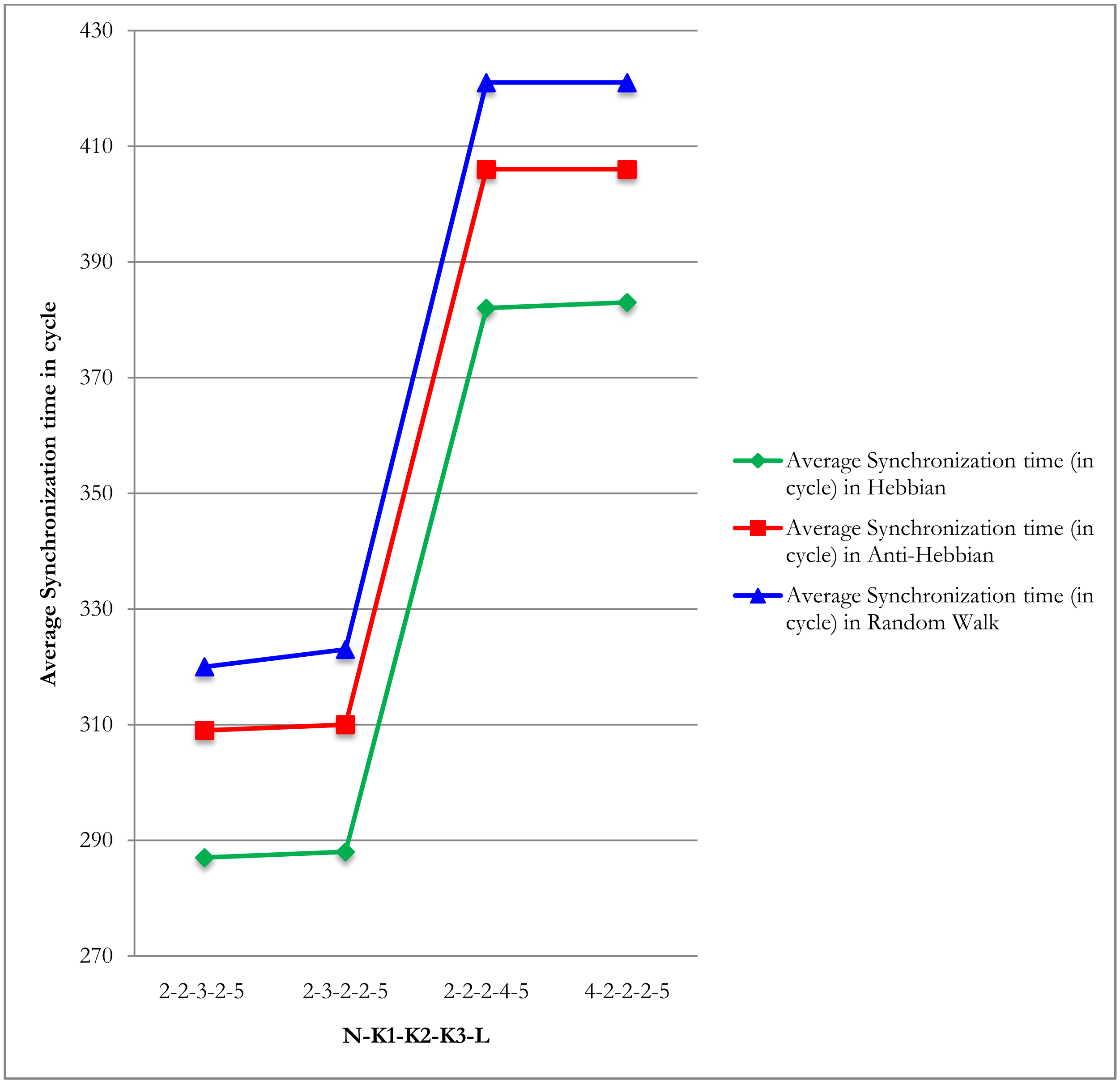


Figure 7.11: Generation of 128 bit session key using fixed weight range $(L = 5)$ with variable neurons in CTHLPSCT

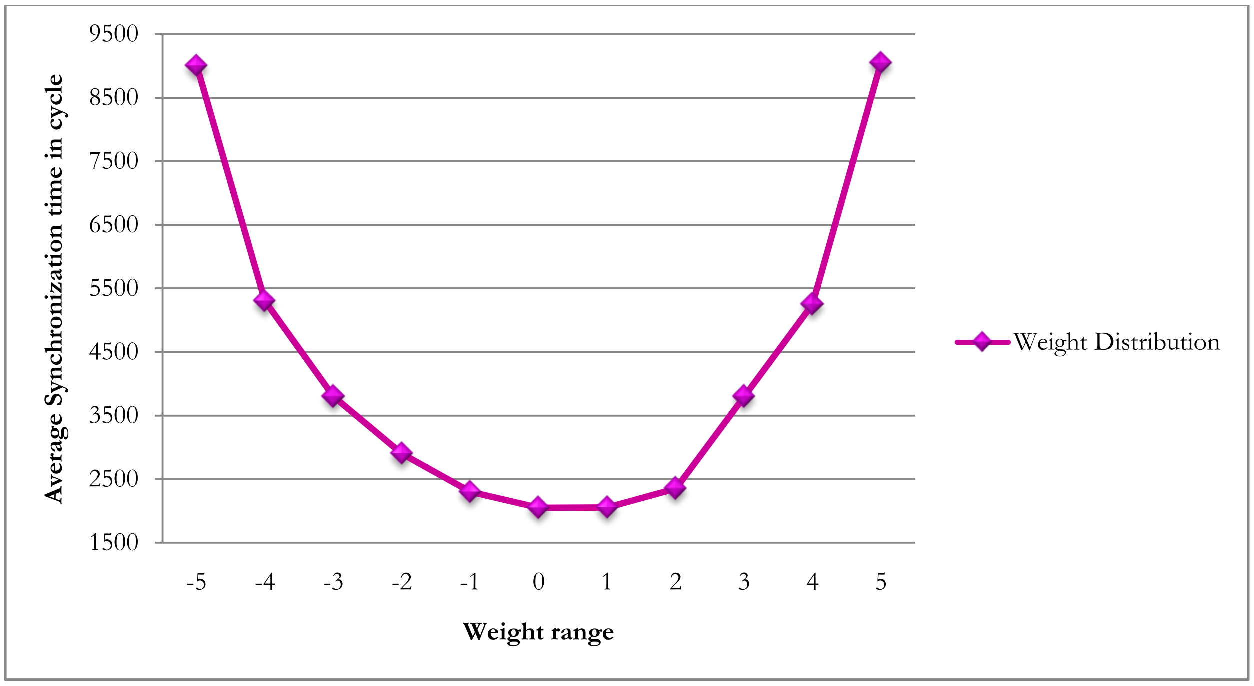


Figure 7.12: Weight distribution in Hebbian learning rule with weight range $(L) = 5$ in CTHLPSCT

From the table 7.39 and figure 7.11 it has been observed that several CTHLPSCT configuration (in terms of different neurons in different layers) can be use to generate 128 bit session key with fixed weight range $L = 5$. CTHLPSCT size is the $N \times K1 \times K2 \times K3$, where $N \times K1$ is number of input, $K1$ is the number of hidden unit in layer 1, $K2$ is the number of hidden unit in layer 2 and $K3$ is the number of hidden unit in layer 3. For the first row CTHLPSCT size is 24, where $N = 2,\ K1 = 2,\ K2 = 3,\ K3 = 2,\ L = 5$. Total number of weights generated by the CTHLPSCT are $(N \times K1 + K1 \times K2 + K2 \times K3)$. Each weight value represented in eight bit binary. So, total $\left((N \times K1 + K1 \times K2 + K2 \times K3) \times 8\right)$ numbers of bits present in a weight (length of a session key). If $N = 2,\ K1 = 2,\ K2 = 3,$ $K3 = 2$ then $((2 \times 2 + 2 \times 3 + 3 \times 2) \times 8) = 128$ bits weight value act as a session key. Among three learning rules Hebbian rules outperform over other two rules (Anti-Hebbian and Random Walk). Hebbian rules perform better where network size is comparatively small because weights are not getting well distributed in the Hebbian rules shown in figure 7.12. So, small network with 128 bit session key Hebbian makes the synchronization faster but for the small network Anti-Hebbian and Random Walk takes much more amount of synchronization time due to the weight distribution process.

Table: 7.40
Generation of 128 bit session key using fixed weight range ($L = 5$) with variable neurons in CGTHLPSCT

CGTHLP Size	N-K1-K2-K3-L	No. of CGTHLP Participated at Group Session Key Generation	Average Synchronization Steps		
			Hebbian	Anti-Hebbian	Random Walk
24	2-2-3-2-5	4	549,28	590,16	611,70
32	2-2-2-4-5	4	730,81	776,50	804,15
24	2-3-2-2-5	8	1952,31	2098,20	2184,74
32	4-2-2-2-5	8	2590,59	2751,03	2852,30

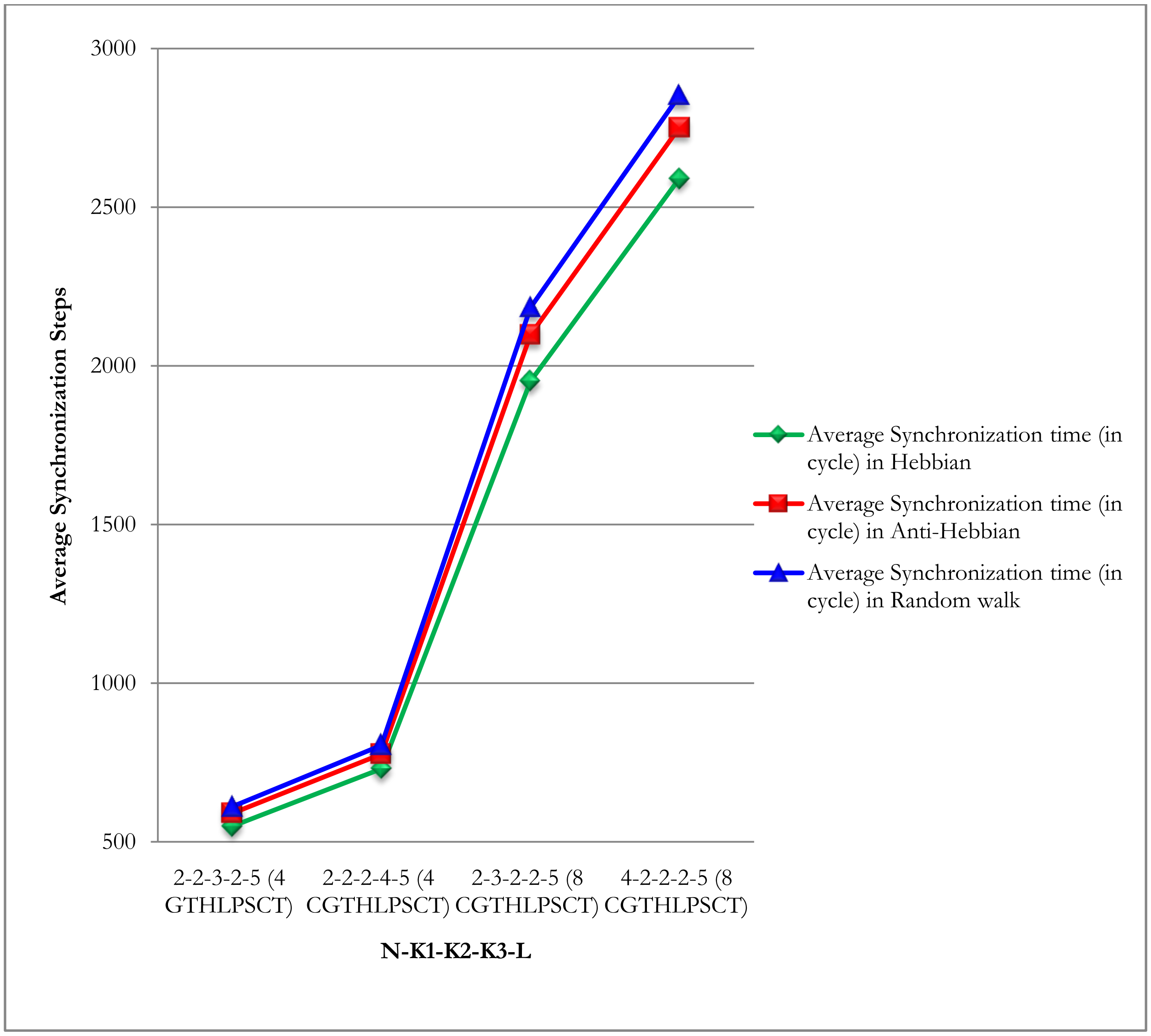


Figure 7.13: Generation of 128 bit session key using fixed weight range ($L = 5$) with variable neurons in CGTHLPSCT

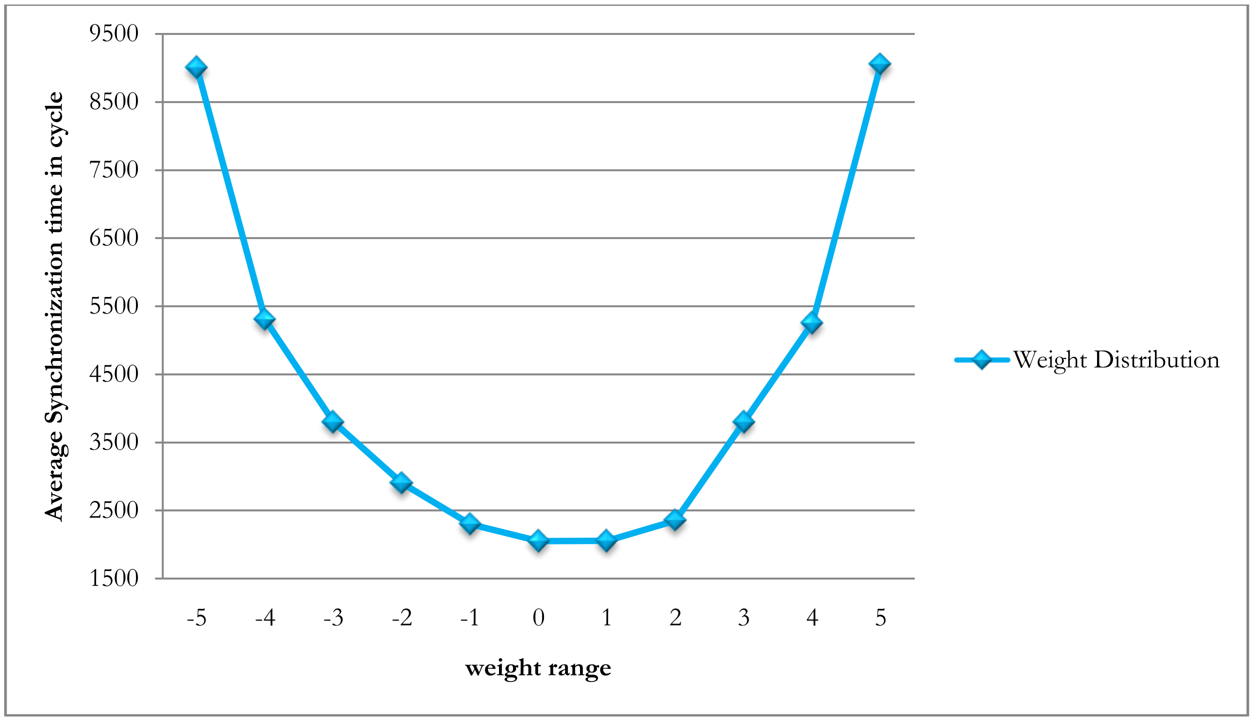


Figure 7.14: Weight distribution in Hebbian learning rule with weight range $(L) = 5$ in CGTHLPSCT

From the table 7.40 and figure 7.13 it has been observed that group of CTHLPSCT with configuration (in terms of different neurons in different layers) can be use to generate 128 bit session key with fixed weight range $L = 5$. CGTHLPSCT needs $nlog(n-1)$ number of synchronizations because this technique use complete binary tree based framework for synchronizing n parties. Whereas other proposed and existing techniques needs $\frac{n(n-1)}{2}$ number of synchronizations for synchronizing n parties. If $n = 4$ then CGTHLPSCT needs only $4log(4-1)$ number of synchronizations. Each CTHLPSCT in the group have the size of $N \times K1 \times K2 \times K3$, where $N \times K1$ is number of input, $K1$ is the number of hidden unit in layer 1 , $K2$ is the number of hidden unit in layer 2 and $K3$ is the number of hidden unit in layer 3. For the first row CTHLPSCT size is 24, where $N = 2, K1 = 2, K2 = 3, K3 = 2, L = 5$. Total number of weights generated by the CTHLPSCT are $(N \times K1 + K1 \times K2 + K2 \times K3)$. Each weight value represented in eight bit binary. So, total $((N \times K1 + K1 \times K2 + K2 \times K3) \times 8)$ numbers of bits present in a weight (length of a session key). If $N = 2, K1 = 2, K2 = 3, K3 = 2$ then $\big((2 \times 2 + 2 \times 3 + 3 \times 2) \times 8\big) = 128$ bits weight value act as a session key. Among three learning rules Hebbian rules outperform over other two rules (Anti-Hebbian and Random Walk). Hebbian rules perform better where network size is comparatively small because weights are not getting well distributed in the Hebbian rules shown in figure 7.14. So, small network with 128 bit session key Hebbian makes the synchronization faster but for the small network Anti-Hebbian and Random Walk takes much more amount of synchronization time due to the weight distribution process.

7.3.4 Average Synchronization Time (in cycle) for Generating 192 bit Session Key using fixed Weight range ($L = 5$) with variable Neurons

Table: 7.41
Generation of 192 bit session key using fixed weight range ($L = 5$) with variable neurons in DHLPSCT

DHLP Size	**N-K1-K2-L**	**Average synchronization time in cycle**		
		Hebbian	**Anti-Hebbian**	**Random Walk**
24	2-6-2-5	328,73	346,23	374,89
32	2-4-4-5	419,26	441,64	457,08
32	4-4-2-5	417,39	438,48	453,11
36	3-4-3-5	452,73	469,91	481,03
45	3-3-5-5	598,27	573,62	588,18
45	5-3-3-5	607,42	581,83	597,39
48	4-3-4-5	643,61	609,10	623,27
72	6-2-6-5	956,71	904,28	909,37

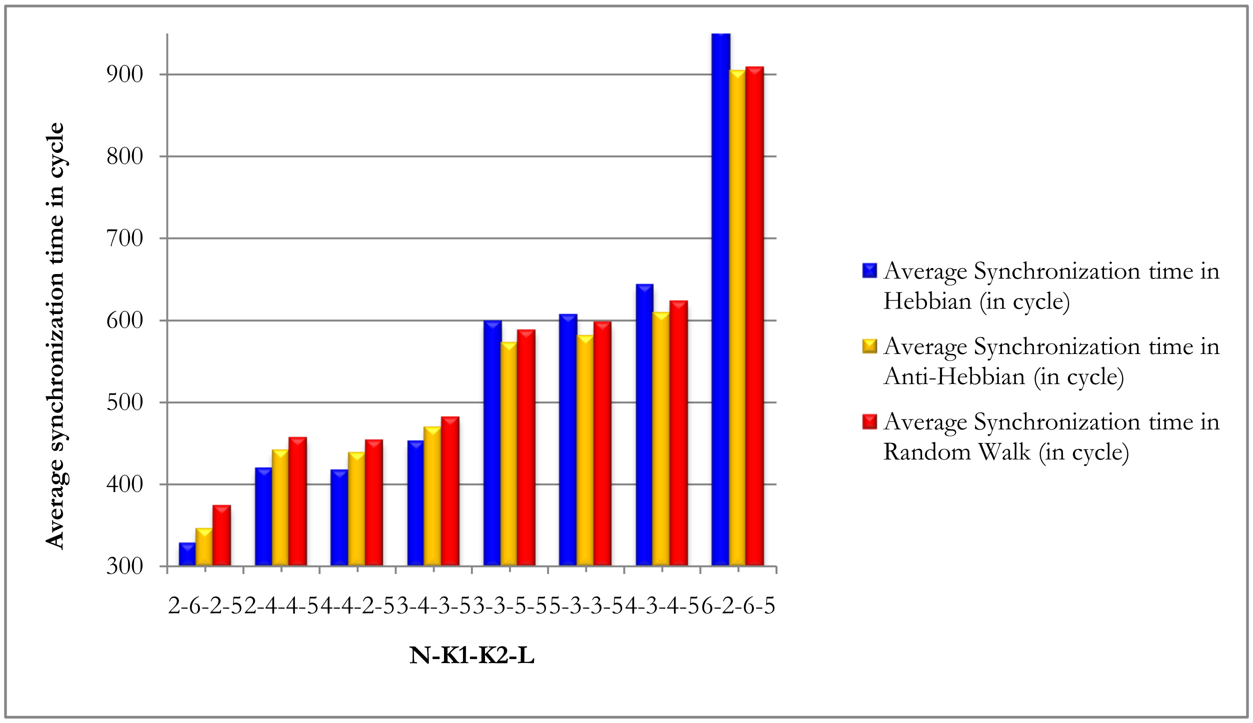


Figure 15: Generation of 192 bit session key using fixed weight range $(L = 5)$ with variable neurons in DHLPSCT

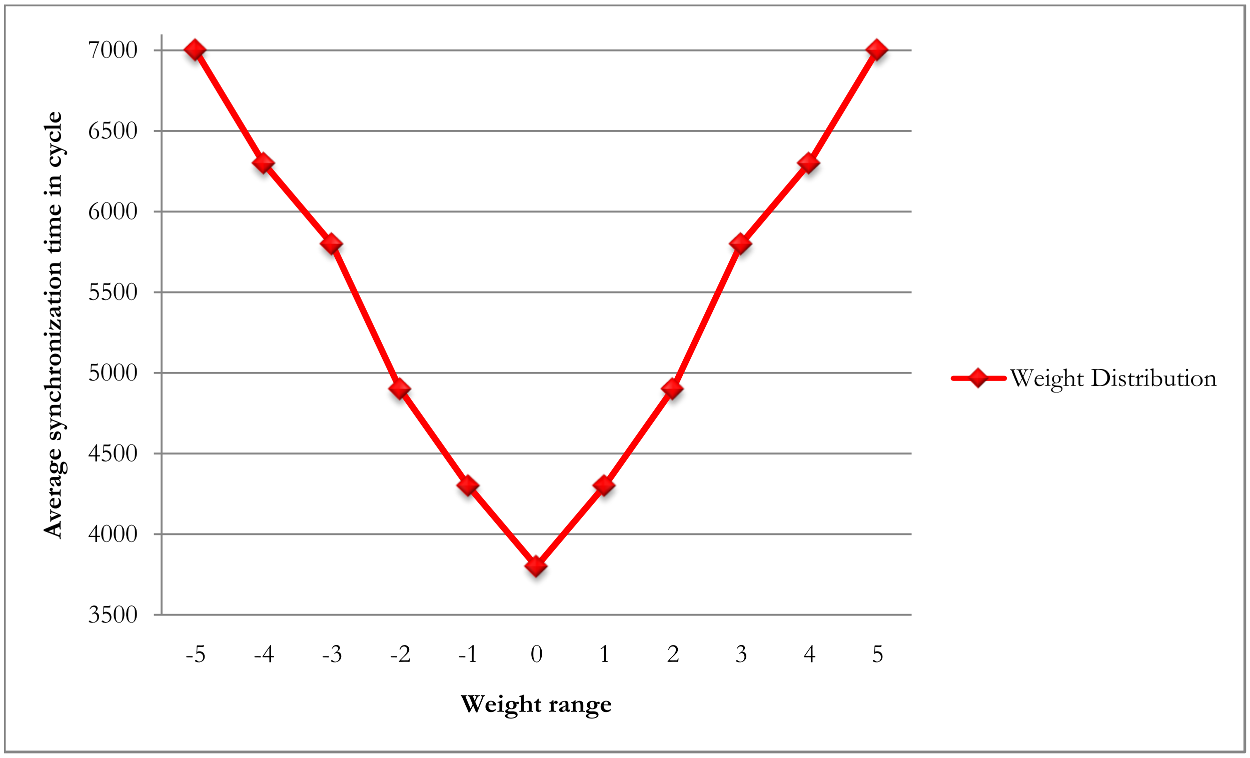


Figure 7.16: Weight distribution in Anti-Hebbian learning rule with weight range $(L) = 5$ in DHLPSCT

From the table 7.41 and figure 7.15 it has been observed that several DHLPSCT configuration (in terms of different neurons in different layers) can be use to generate 192 bit session key with fixed weight range $L = 5$. DHLPSCT size is the $N \times K1 \times K2$, where $N \times K1$ is number of input units, $K1$ is the number of hidden units in layer 1 and $K2$ is the number of hidden units in layer 2. For the first row DHLPSCT size is 24, where $N = 2$, $K1 = 6$, $K2 = 2$, $L = 5$. Total number of weights generated by the DHLPSCT are $(N \times K1 + K1 \times K2)$. Each weight value represented in eight bit binary. So, total $((N \times K1 + K1 \times K2) \times 8)$ numbers of bits present in a weight (length of a session key). If $N = 2$, $K1 = 6$, $K2 = 2$ then $\left((2 \times 6 + 6 \times 2) \times 8\right) = 192$ bits weight value act as a session key. Among three learning rules Anti-Hebbian rules outperform over other two rules when network size is medium (45 to 72). Hebbian rules perform better where network size is small (less than 45). In Anti-Hebbian rule weights are getting well distributed than Hebbian rules shown in figure 7.16. So, network having medium size with 192 bit session key Anti-Hebbian makes the synchronization faster but for this range Hebbian and Random Walk takes much more amount of synchronization time.

Table: 7.42
Generation of 192 bit session key using fixed weight range $(L = 5)$ with variable neurons in CDHLPSCT

CDHLP Size	**N-K1-K2-L**	**Average Synchronization time in cycle**		
		Hebbian	**Anti-Hebbian**	**Random Walk**
24	2-6-2-5	289,35	311,62	332,14
32	2-4-4-5	383,78	406,89	422,17
32	4-4-2-5	384,10	406,15	421,08
36	3-4-3-5	418,69	427,83	435,86
45	3-3-5-5	559,18	538,59	546,13
45	5-3-3-5	560,23	539,26	547.64
48	4-3-4-5	602,76	574,35	581,08
72	6-2-6-5	923,85	862,19	866,11

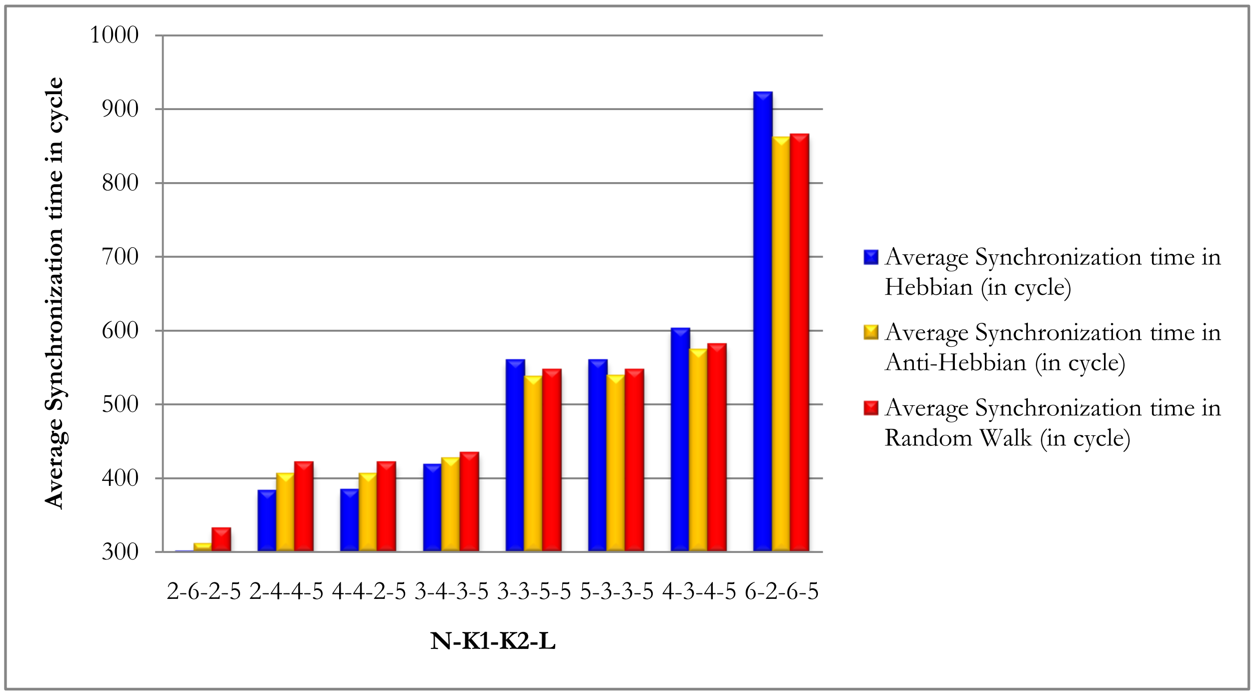


Figure 7.17: Generation of 192 bit session key using fixed weight range $(L = 5)$ with variable neurons in CDHLPSCT

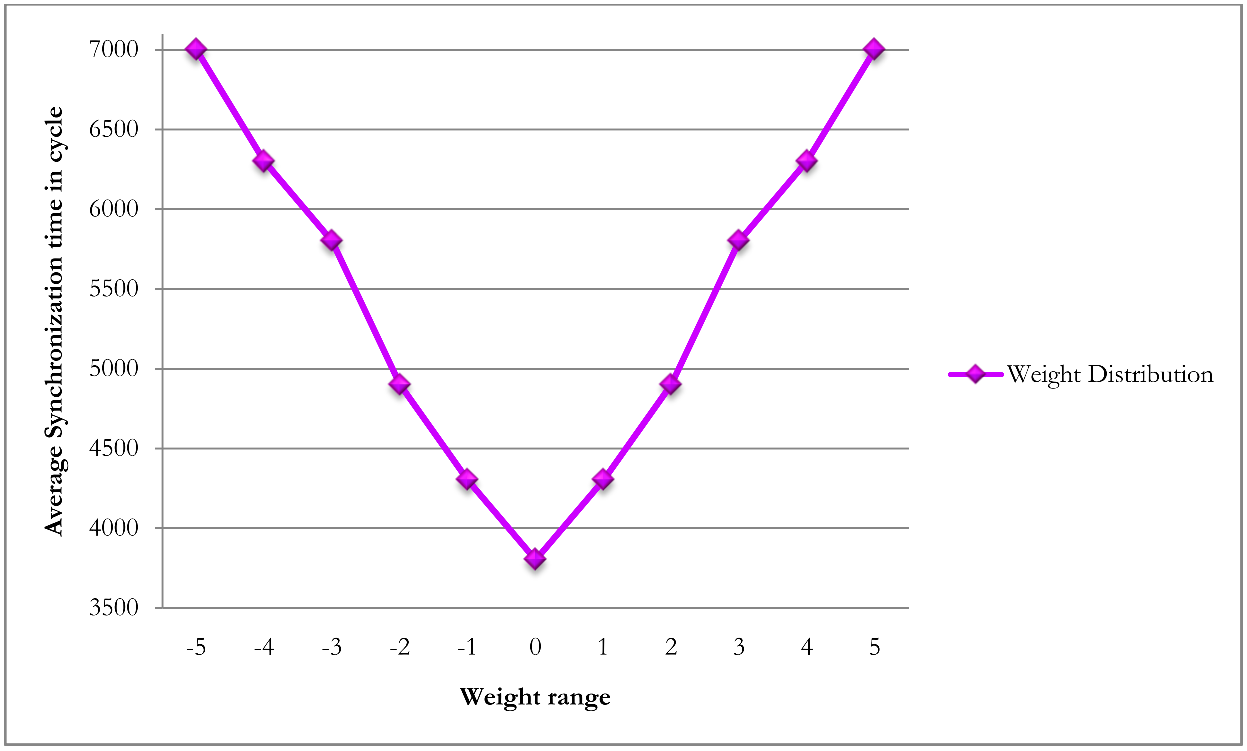


Figure 7.18: Weight distribution in Anti-Hebbian learning rule with weight range $(L) = 5$ in CDHLPSCT

From the table 7.42 and figure 7.17 it has been observed that several CDHLPSCT configuration (in terms of different neurons in different layers) can be use to generate 192 bit session key with fixed weight range $L = 5$. DHLPSCT size is the $N \times K1 \times K2$, where $N \times K1$ is number of input units, $K1$ is the number of hidden units in layer 1 and $K2$ is the number of hidden units in layer 2. For the first row CDHLPSCT size is 24, where $N = 2,\ K1 = 6,\ K2 = 2,\ L = 5$. Total numbers of weights generated by the CDHLPSCT are $(N \times K1 + K1 \times K2)$. Each weight value represented in eight bit binary. So, total $((N \times K1 + K1 \times K2) \times 8)$ numbers of bits present in a weight (length of a session key). If $N = 2,\ K1 = 6,\ K2 = 2$ then $\left((2 \times 6 + 6 \times 2) \times 8\right) = 192$ bits weight value act as a session key. Among three learning rules Anti-Hebbian rules outperform over other two rules when network size is medium $(45$ to $72)$. Hebbian rules perform better where network size is small (less than 45). In Anti-Hebbian rule weights are getting well distributed than Hebbian rules shown in figure 7.18. So, network having medium size with 192 bit session key Anti-Hebbian makes the synchronization faster but for this range Hebbian and Random Walk takes much more amount of synchronization time.

Table: 7.43
Generation of 192 bit session key using fixed weight range $(L = 5)$ with variable neurons in CTHLPSCT

CTHLP Size	N-K1-K2-K3-L	Average Synchronization time in cycle		
		Hebbian	Anti-Hebbian	Random Walk
40	2-2-5-2-5	451,17	434,83	443,02
40	2-5-2-2-5	454,37	436,11	447,19
54	2-3-3-3-5	677,76	645,89	653,92
54	3-3-3-2-5	679,23	646,05	655,11
64	2-2-2-8-5	805.71	765,53	776,84
64	4-1-4-4-5	806.16	766,10	777,61
64	4-4-1-4-5	806.98	766,87	778,41
64	8-2-2-2-5	807.24	767,63	779,12

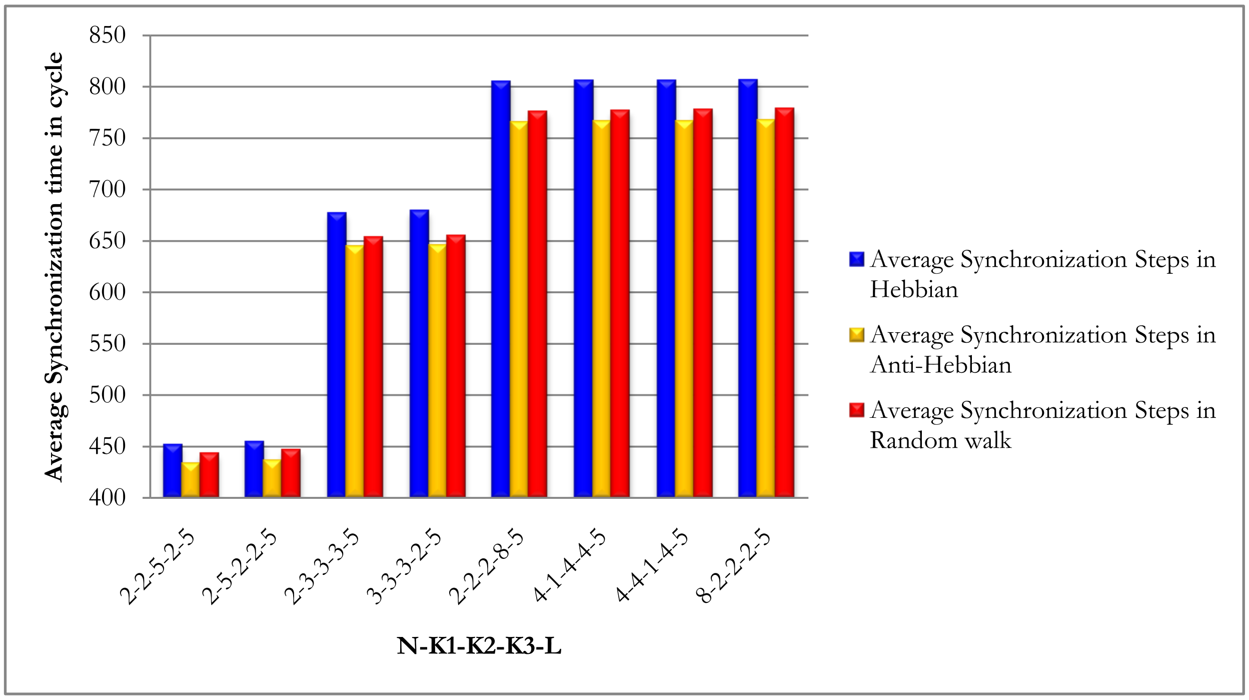


Figure 7.19: Generation of 192 bit session key using fixed weight range $(L = 5)$ with variable neurons in CTHLPSCT

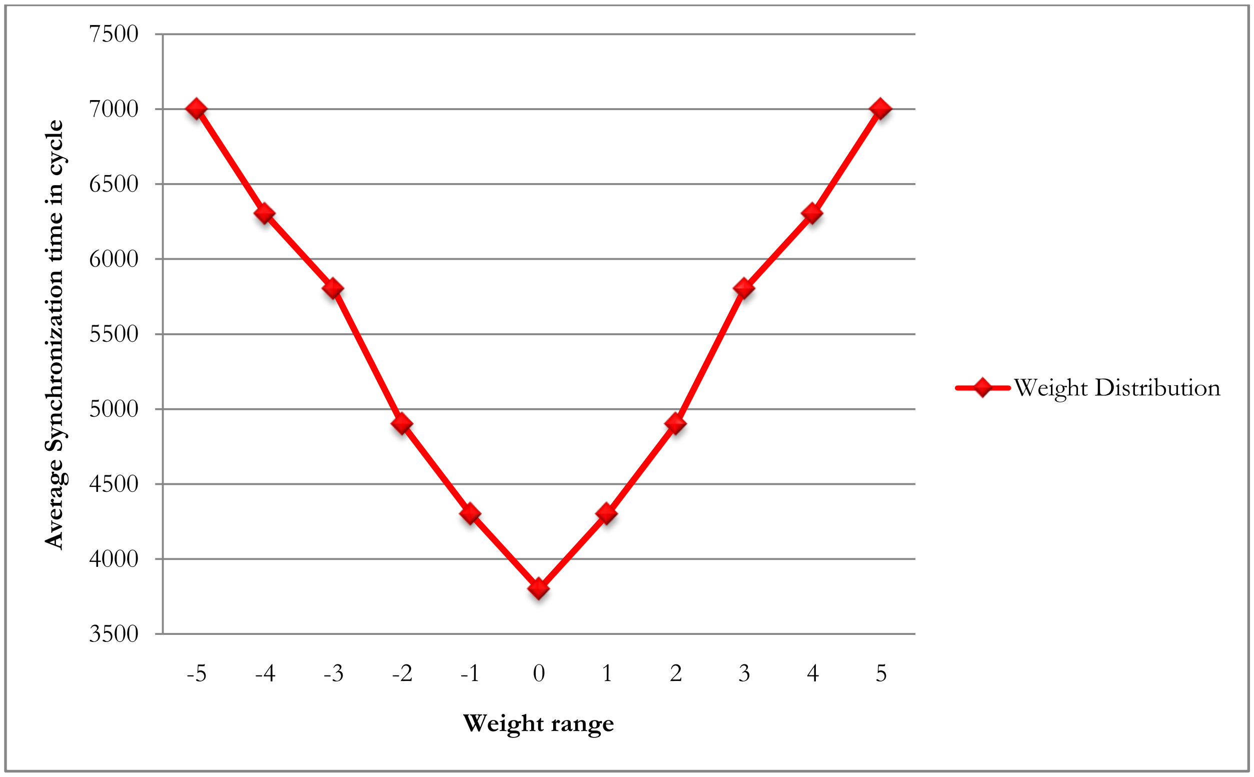


Figure 7.20: Weight distribution in Anti-Hebbian learning rule with weight range $(L) = 5$ in CTHLPSCT

From the table 7.43 and figure 7.19 it has been observed that several CTHLPSCT configuration (in terms of different neurons in different layers) can be use to generate 128 bit session key with fixed weight range $L = 5$. CTHLPSCT size is the $N \times K1 \times K2 \times K3$, where N is number of input, $K1$ is the number of hidden unit in layer 1 , $K2$ is the number of hidden unit in layer 2 and $K3$ is the number of hidden unit in layer 3. For the first row CTHLPSCT size is 40, where $N = 2,\ K1 = 5,\ K2 = 2,\ K3 = 2,\ L = 5$. Total number of weights generated by the CTHLPSCT are $(N \times K1 + K1 \times K2 + K2 \times K3)$. Each weight value represented in eight bit binary. So, total $((N \times K1 + K1 \times K2 + K2 \times K3) \times 8)$ numbers of bits present in a weight (length of a session key). If $N = 2,\ K1 = 2,\ K2 = 3,$ $K3 = 2$ then $\big((2 \times 5 + 5 \times 2 + 2 \times 2) \times 8\big) = 192$ bits weight value act as a session key. Among three learning rules Anti-Hebbian rules outperform over other two rules (Hebbian and Random walk). Anti-Hebbian rules perform better where network size is medium. In Anti-Hebbian rule weights are getting well distributed than Hebbian rules shown in 7.20. So, network having medium size with 192 bit session key Anti-Hebbian makes the synchronization faster but for this range Hebbian and Random walk takes much more amount of synchronization time.

Table: 7.44
Generation of 192 bit session key using fixed weight range $(L = 5)$ with variable neurons in CGTHLPSCT

CTHLP Size	**N-K1-K2-K3-L**	**No. of CTHLP Participated at Group Session Key Generation**	**Average Synchronization steps in cycle**		
			Hebbian	**Anti-Hebbian**	**Random Walk**
40	2-2-5-2-5	4	861,05	829,86	845,49
40	2-5-2-2-5	8	3071,89	2948,44	3023,35
54	2-3-3-3-5	4	1293,49	1232,67	1247,99
54	3-3-3-2-5	8	4592,12	4367,80	4429,05
64	2-2-2-8-5	4	1537,68	1461,00	1482,58
64	4-1-4-4-5	8	5450,27	5179,43	5257,25
64	4-4-1-4-5	10	7700,54	7317,79	7427,91
64	8-2-2-2-5	12	10087,84	9592,84	9736,43

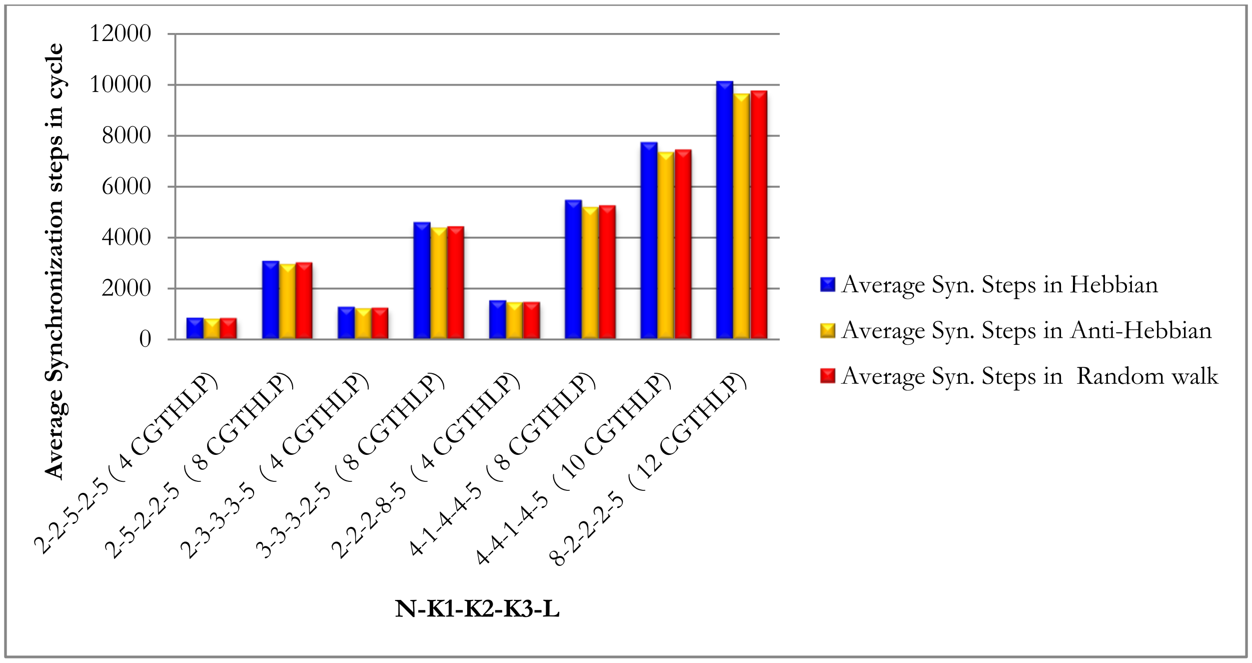


Figure 7.21: Generation of 192 bit session key using fixed weight range $(L = 5)$ with variable neurons in CGTHLPSCT

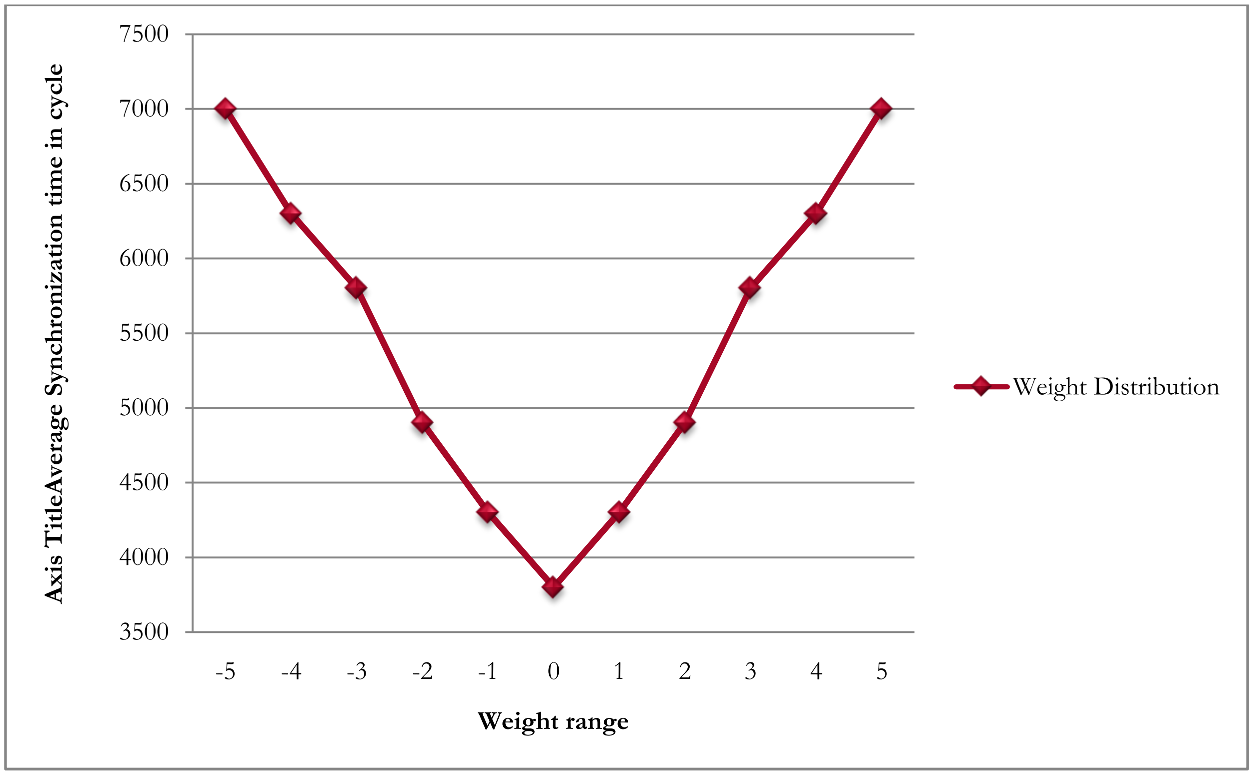


Figure 7.22: Weight distribution in Anti-Hebbian learning rule with weight range $(L) = 5$ in CGTHLPSCT

From the table 7.44 and figure 7.21 it has been observed that several CTHLPSCT makes a group and different or same configuration (in terms of different neurons in different layers) of each CTHLPSCT can be use to generate 192 bit session key with fixed weight range $L = 5$. CTHLPSCT size is the $N \times K1 \times K2 \times K3$, where N is number of input, $K1$ is the number of hidden unit in layer 1, $K2$ is the number of hidden unit in layer 2 and $K3$ is the number of hidden unit in layer 3. For the first row CTHLPSCT size is 40, where $N = 2,\ K1 = 5,\ K2 = 2,\ K3 = 2,\ L = 5$. Total numbers of weights generated by the CTHLPSCT are $(N \times K1 + K1 \times K2 + K2 \times K3)$. Each weight value represented in eight bit binary. So, total $\big((N \times K1 + K1 \times K2 + K2 \times K3) \times 8\big)$ numbers of bits present in a weight (length of a session key). If $N = 2,\ K1 = 2,\ K2 = 3,\ K3 = 2$ then $\big((2 \times 5 + 5 \times 2 + 2 \times 2) \times 8\big) = 192$ bits weight value act as a session key. CGTHLPSCT needs $nlog(n-1)$ number of synchronizations because this technique use complete binary tree based framework for synchronizing n parties. Whereas other proposed and existing techniques needs $\frac{n(n-1)}{2}$ number of synchronizations for synchronizing n parties. If $n = 4$ then CGTHLPSCT needs only $4log(4-1)$ number of synchronizations.Among three learning rules Anti-Hebbian rules outperform over other two rules (Hebbian and Random Walk). Anti-Hebbian rules perform better where network size is medium. In Anti-Hebbian rule weights are getting well distributed than Hebbian rules shown in figure 7.22. So, network having medium size with 192 bit session key Anti-Hebbian makes the synchronization faster but for this range Hebbian and Random Walk takes much more amount of synchronization time.

7.3.5 Average Synchronization Time (in cycle) for Generating 256 bit Session Key using fixed Weight range ($L = 5$) with variable Neurons

Table: 7.45
Generation of 256 bit session key using fixed weight range ($L = 5$) with variable neurons in DHLPSCT

DHLPSCT Size	**N-K1-K2-L**	**Average Synchronization time in cycle**		
		Hebbian	**Anti-Hebbian**	**Random Walk**
32	2-8-2-5	406,39	432,07	459,28
64	4-4-4-5	861,47	797,29	803,12
128	8-2-8-5	1752,83	1632,94	1573,48
256	16-1-16-5	3517,29	3339,08	3154,61

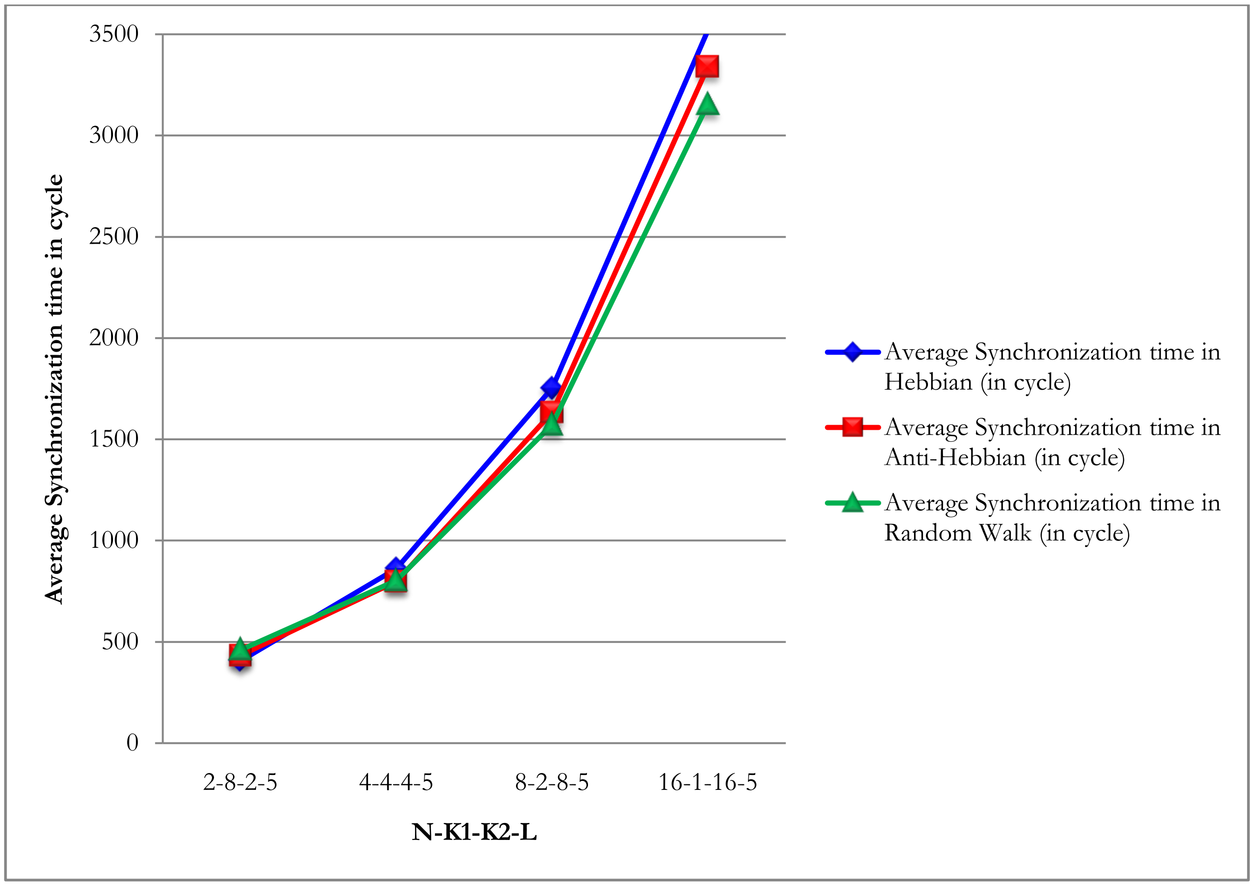


Figure 7.23: Generation of 256 bit session key using fixed weight range ($L = 5$) with variable neurons in DHLPSCT

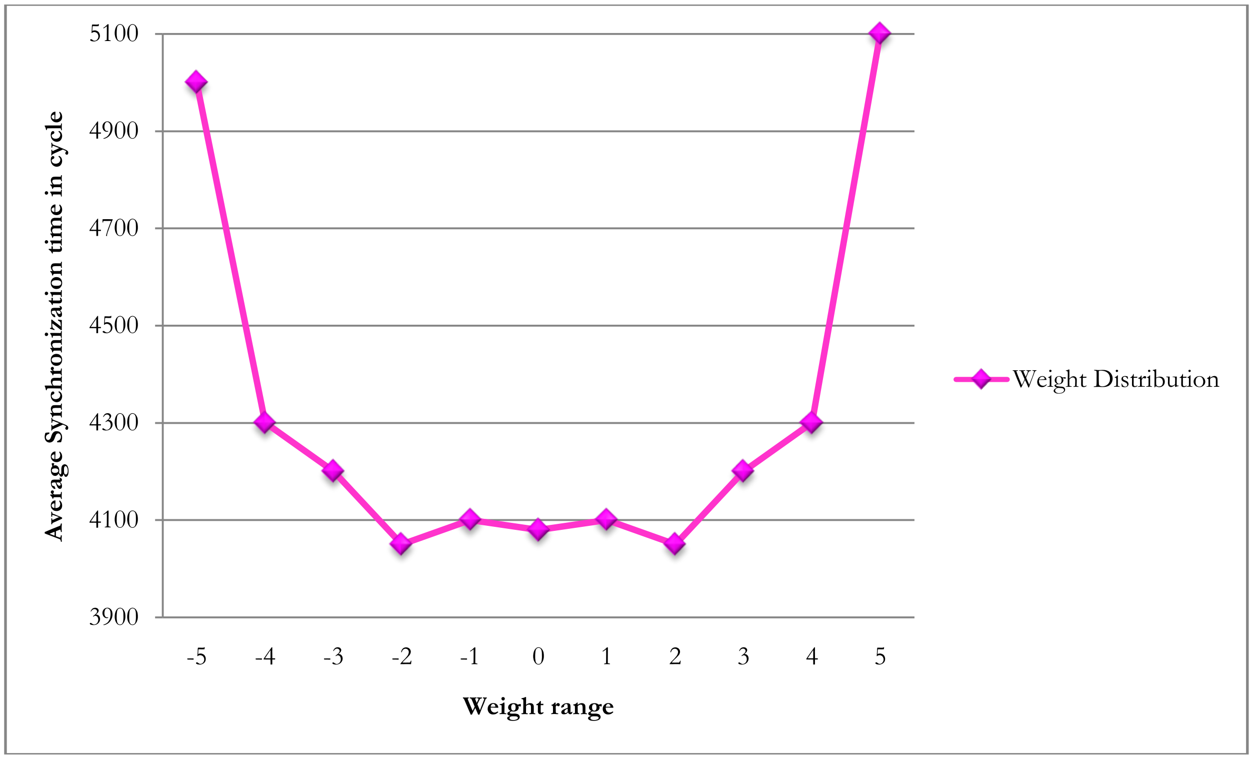


Figure 7.24: Weight distribution in Random Walk learning rule with weight range $(L) = 5$ in DHLPSCT

From the table 7.45 and graph 7.23 it has been observed that several DHLPSCT configuration (in terms of different neurons in different layers) can be use to generate 256 bit session key with fixed weight range $L = 5$. DHLPSCT size is the $N \times K1 \times K2$, where $N \times K1$ is number of input units, $K1$ is the number of hidden units in layer 1 and $K2$ is the number of hidden units in layer 2. For the first row DHLPSCT size is 32, where $N = 2$, $K1 = 8$, $K2 = 2$, $L = 5$. Total numbers of weights generated by the DHLPSCT are $(N \times K1 + K1 \times K2)$. Each weight value represented in eight bit binary. So, total $((N \times K1 + K1 \times K2) \times 8)$ numbers of bits present in a weight (length of a session key). $N = 2$, $K1 = 8$, $K2 = 2$ then $\left((2 \times 8 + 8 \times 2) \times 8\right) = 256$ bits weight value act as a session key. Among three learning rules Random Walk rules outperform over other two rules (Hebbian and Anti-Hebbian) when network sixe is large (128 and more). In Random Walk rule weights are getting well distributed than Hebbian and Anti-Hebbian rules shown in figure 7.24. So, network having size grater than equal to 128 Random Walk makes the synchronization faster but for this range Hebbian and Anti-Hebbian takes much more amount of synchronization step.

Table: 7.46
Generation of 256 bit session key using fixed weight range ($L = 5$) with variable neurons in CDHLPSCT

CDHLP Size	**N-K1-K2-L**	**Average Synchronization time in cycle**		
		Hebbian	**Anti-Hebbian**	**Random Walk**
32	2-8-2-5	383,64	407,14	427,33
64	4-4-4-5	826,16	765,85	770,32
128	8-2-8-5	1687,28	1582,13	1531,42
256	16-1-16-5	3429,73	3248,29	3062,74

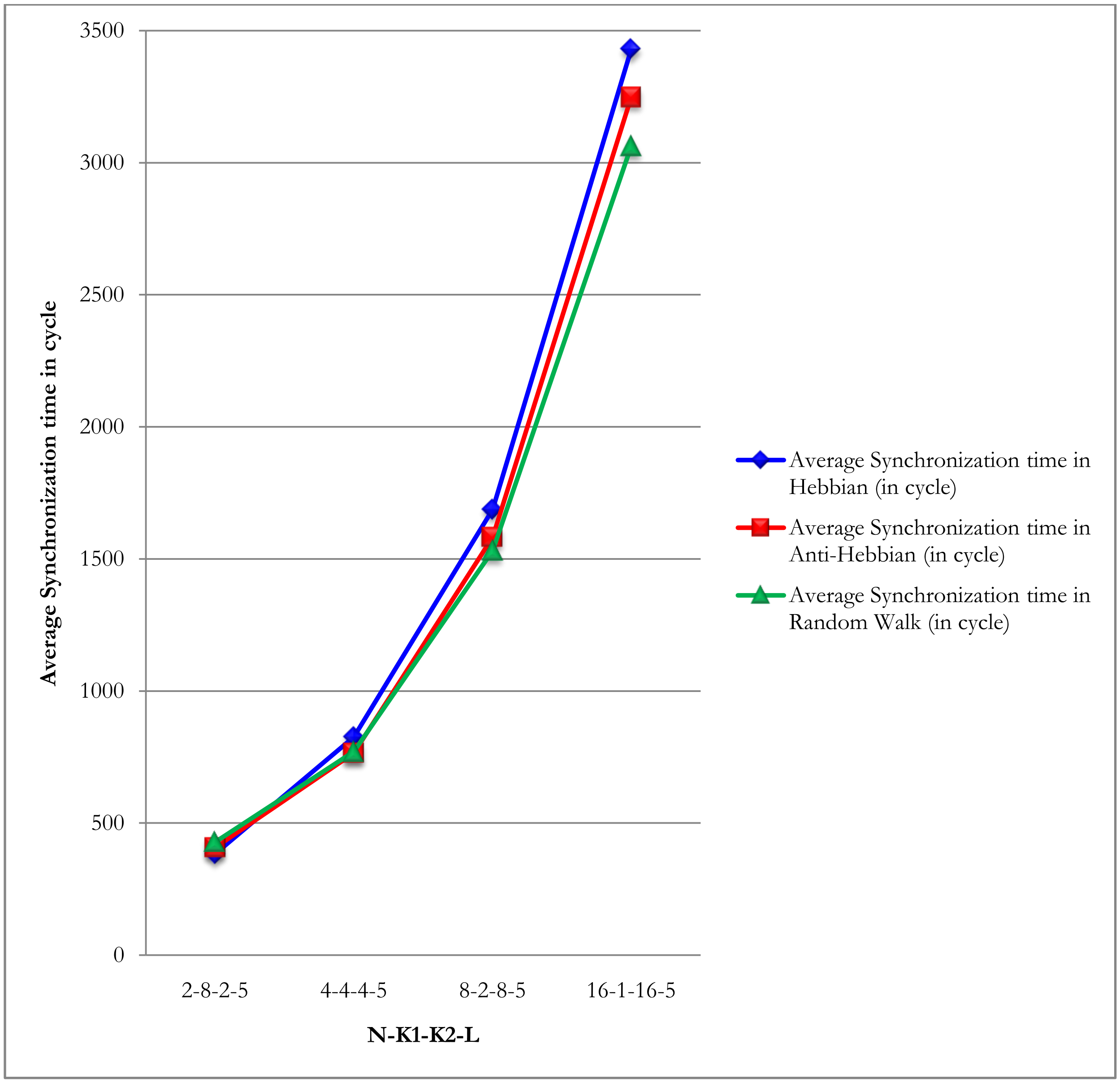


Figure 7.25: Generation of 256 bit session key using fixed weight range ($L = 5$) with variable neurons in CDHLPSCT

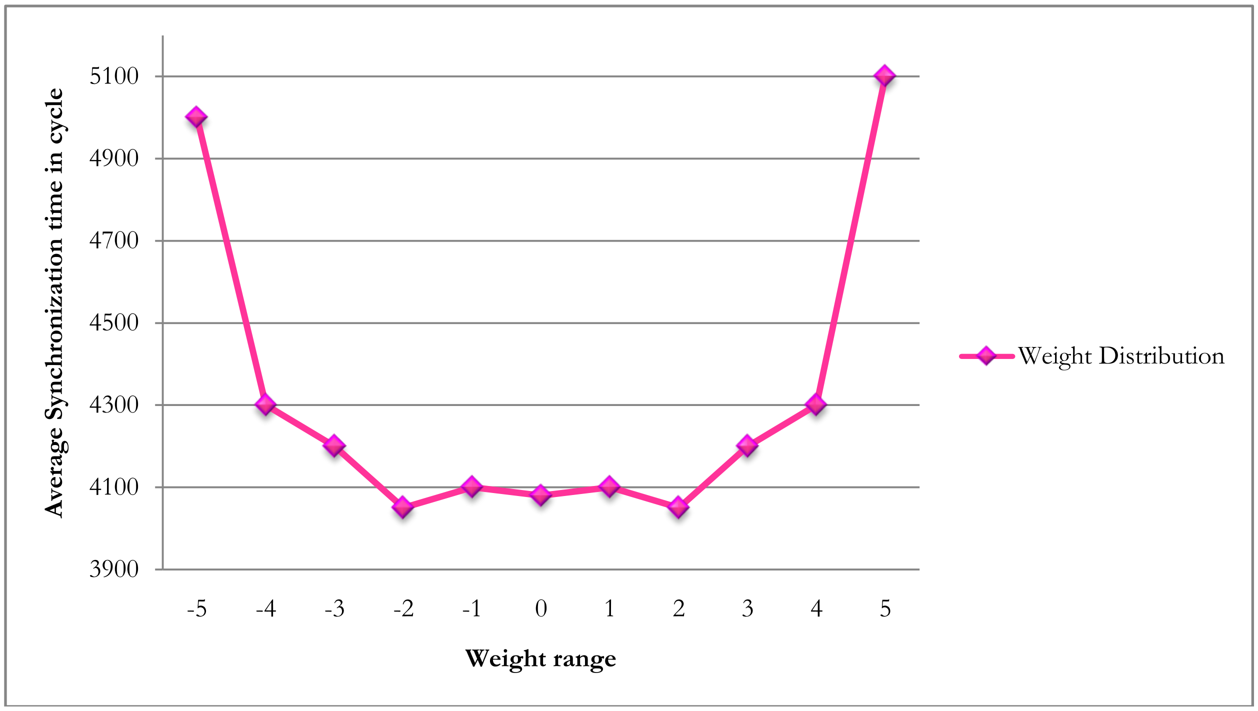


Figure 7.26: Weight distribution in Random Walk learning rule with weight range $(L) = 5$ in CDHLPSCT

From the table 7.46 and graph 7.25 it has been observed that several CDHLPSCT configuration (in terms of different neurons in different layers) can be use to generate 256 bit session key with fixed weight range $L = 5$. CDHLPSCT size is the $N \times K1 \times K2$, where $N \times K1$ is number of input units, $K1$ is the number of hidden units in layer 1 and $K2$ is the number of hidden units in layer 2. For the first row CDHLPSCT size is 32, where $N = 2$, $K1 = 8$, $K2 = 2$, $L = 5$. Total number of weights generated by the CDHLPSCT are $(N \times K1 + K1 \times K2)$. Each weight value represented in eight bit binary. So, total $\left((N \times K1 + K1 \times K2) \times 8\right)$ numbers of bits present in a weight (length of a session key). $N = 2$, $K1 = 8$, $K2 = 2$ then $\left((2 \times 8 + 8 \times 2) \times 8\right) = 256$ bits weight value act as a session key. Among three learning rules Random Walk rules outperform over other two rules (Hebbian and Anti-Hebbian) when network sixe is large (128 and more). In Random Walk rule weights are getting well distributed than Hebbian and Anti-Hebbian rules shown in figure 7.26. So, network having size greater than equal to 128 Random Walk makes the synchronization faster but for this range Hebbian and Anti-Hebbian takes much more amount of synchronization step.

Table: 7.47
Generation of 256 bit session key using fixed weight range $(L = 5)$ with variable neurons in CTHLPSCT

CTHLP Size	N-K1-K2-K3-L	Average Synchronization time in cycle		
		Hebbian	Anti-Hebbian	Random Walk
56	2-2-7-2-5	722,16	669,34	673,71
56	2-7-2-2-5	724,03	670,19	674,25
128	4-2-4-4-5	1686,93	1581,87	1519,18
128	4-4-2-4-5	1687,32	1582,17	1521,38

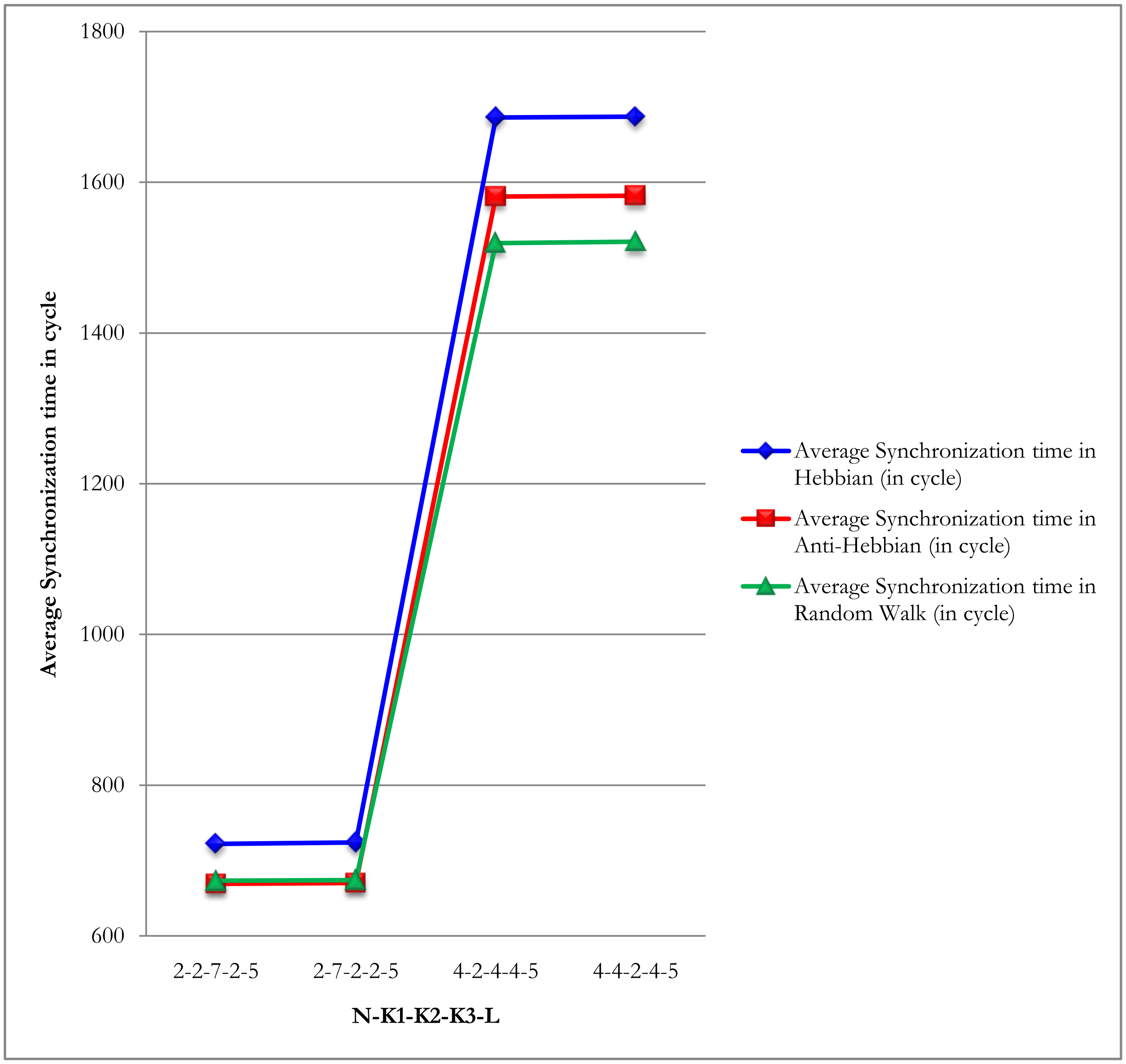


Figure 7.27: Generation of 256 bit session key using fixed weight range $(L = 5)$ with variable neurons in CTHLPSCT

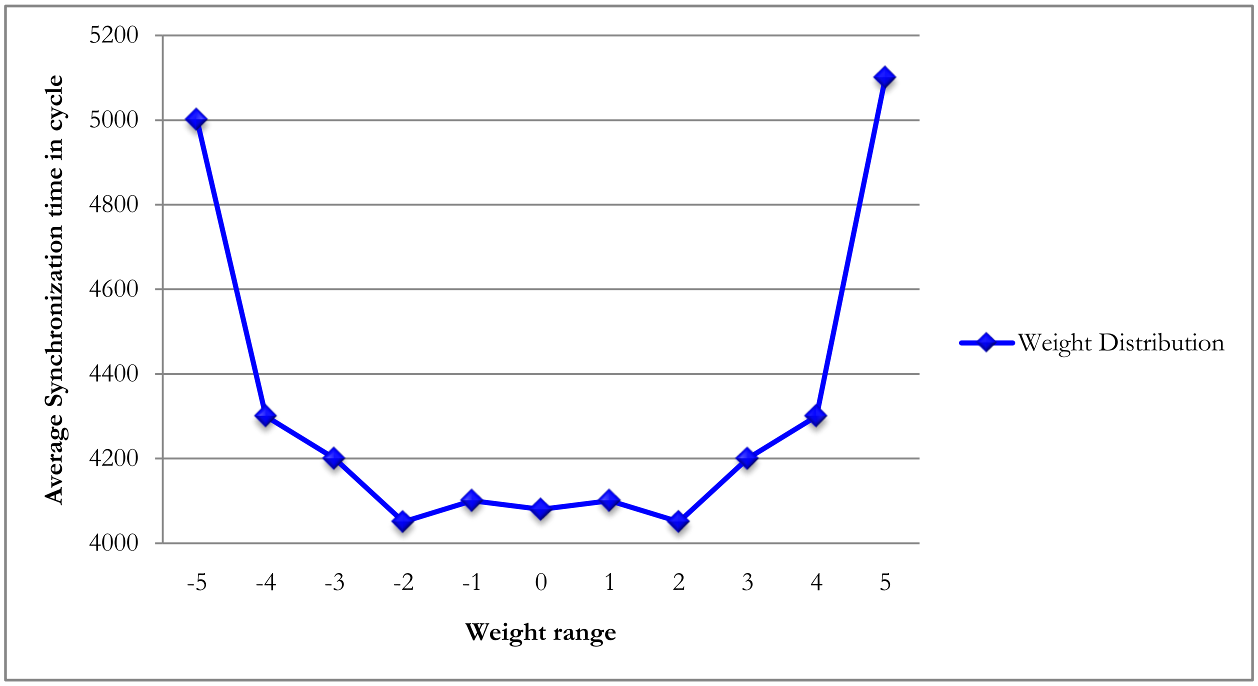


Figure 7.28: Weight distribution in Random Walk learning rule with weight range $(L) = 5$ in CTHLPSCT

From the table 7.47 and graph 7.27 it has been observed that several CTHLPSCT configuration (in terms of different neurons in different layers) can be use to generate 128 bit session key with fixed weight range $L = 5$. CTHLP size is the $N \times K1 \times K2 \times K3$, where N is number of input, $K1$ is the number of hidden unit in layer 1, $K2$ is the number of hidden unit in layer 2 and $K3$ is the number of hidden unit in layer 3. For the first row CTHLPSCT size is 40, where $N = 2,\ K1 = 2,\ K2 = 7,\ K3 = 2,\ L = 5$. Total number of weights generated by the CTHLPSCT are $(N \times K1 + K1 \times K2 + K2 \times K3)$. Each weight value represented in eight bit binary. So, total $((N \times K1 + K1 \times K2 + K2 \times K3) \times 8)$ numbers of bits present in a weight (length of a session key). If $N = 2,\ K1 = 2,\ K2 = 7,\ K3 = 2$ then $\big((2 \times 2 + 2 \times 7 + 7 \times 2) \times 8\big) = 256$ bits weight value act as a session key. Among three learning rules Random Walk rules outperform over other two rules (Hebbian and Anti-Hebbian). Random Walk rules perform better where network size is big. In Random Walk rule weights are getting well distributed than Hebbian and Anti-Hebbian rules shown in figure 7.28. So, network having size grater than equal to 128 Random Walk makes the synchronization faster but for this range Hebbian and Anti-Hebbian takes much more amount of synchronization time.

Table: 7.48
Generation of 256 bit session key using fixed weight range ($L = 5$) with variable neurons in CGTHLPSCT

CTHLP Size	N-K1-K2-K3-L	No. of CTHLP Participated at Group Session Key Generation	Average Synchronization steps in cycle		
			Hebbian	Anti-Hebbian	Random walk
56	2-2-7-2-5	4	1378,23	1277,42	1285,76
128	4-2-4-4-5	4	3219,48	3018,97	2899,33
56	2-7-2-2-5	8	4895,01	4531,01	4558,45
128	4-4-2-4-5	8	11407,61	10696,71	10285,72

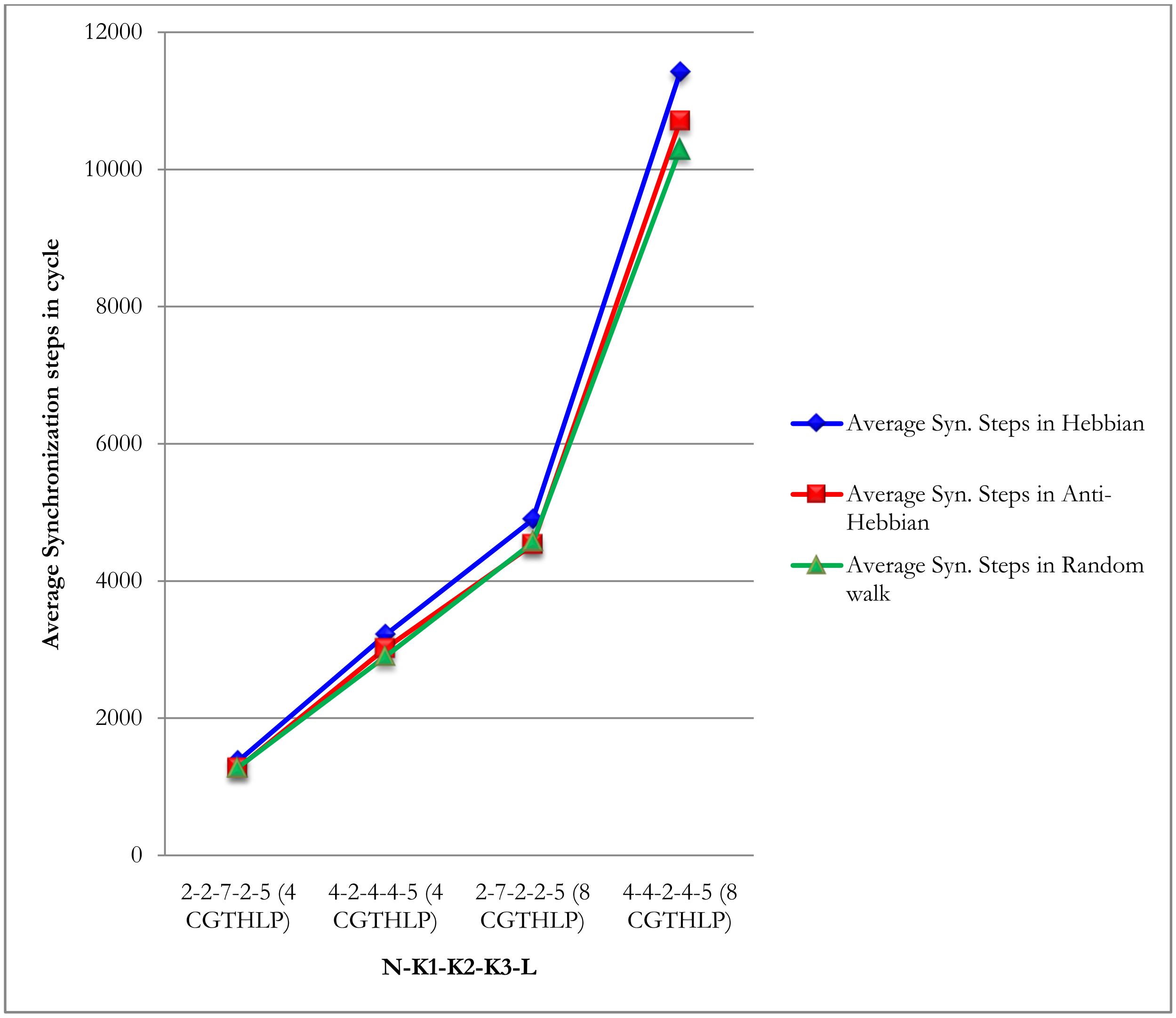


Figure 7.29: Generation of 256 bit session key using fixed weight range ($L = 5$) with variable neurons in CGTHLPSCT

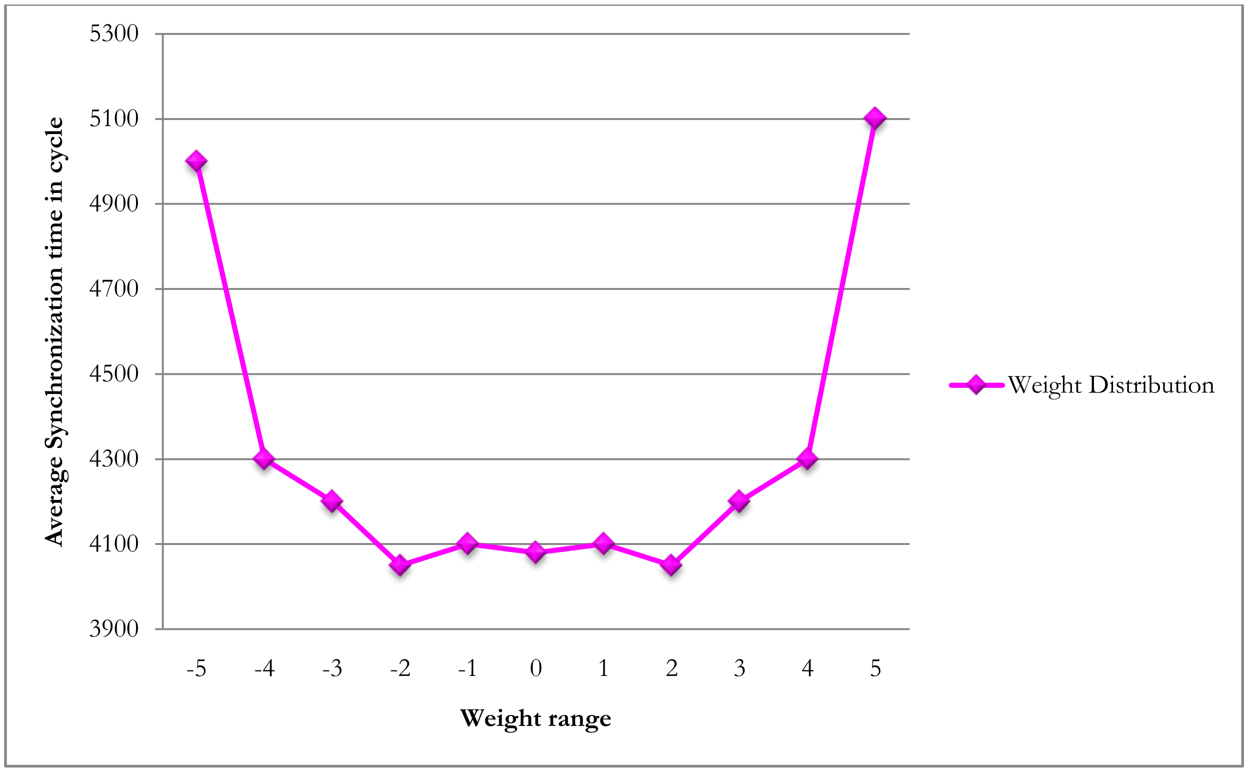


Figure 7.30: Weight distribution in Random Walk learning rule with weight range $(L) = 5$ in CGTHLPSCT

From the table 7.48 and graph 7.29 it has been observed that a group of CTHLPSCT can synchronize together. Each CTHLPSCT configuration (in terms of different neurons in different layers) can be use to generate 256 bit session key with fixed weight range $L = 5$ in the group. CGTHLPSCT needs $nlog(n-1)$ number of synchronizations because this technique use complete binary tree based framework for synchronizing n parties. Whereas other proposed and existing techniques needs $\frac{n(n-1)}{2}$ number of synchronizations for synchronizing n parties. If $n = 4$ then CGTHLPSCT needs only $4log(4-1)$ number of synchronizations. CTHLPSCT size is the $N \times K1 \times K2 \times K3$, where N is number of input, $K1$ is the number of hidden unit in layer 1, $K2$ is the number of hidden unit in layer 2 and $K3$ is the number of hidden unit in layer 3. For the first row CTHLPSCT size is 40, where $N = 2,\ K1 = 2,\ K2 = 7,\ K3 = 2,\ L = 5$. Total number of weights generated by the CTHLPSCT are $(N \times K1 + K1 \times K2 + K2 \times K3)$. Each weight value represented in eight bit binary. So, total $((N \times K1 + K1 \times K2 + K2 \times K3) \times 8)$ numbers of bits present in a weight (length of a session key). If $N = 2,\ K1 = 2,\ K2 = 7,\ K3 = 2$ then $\big((2 \times 2 + 2 \times$

$7 + 7 \times 2) \times 8) = 256$ bits weight value act as a session key. Among three learning rules Random Walk rules outperform over other two rules (Hebbian and Anti-Hebbian). Random Walk rules perform better where network size is big. In Random Walk rule weights are getting well distributed than Hebbian and Anti-Hebbian rules shown in figure 7.30. So, network having size greater than equal to 128 Random Walk makes the synchronization faster but for this range Hebbian and Anti-Hebbian takes much more amount of synchronization time.

7.3.6 Average synchronization time (in cycle) for generating variable session key

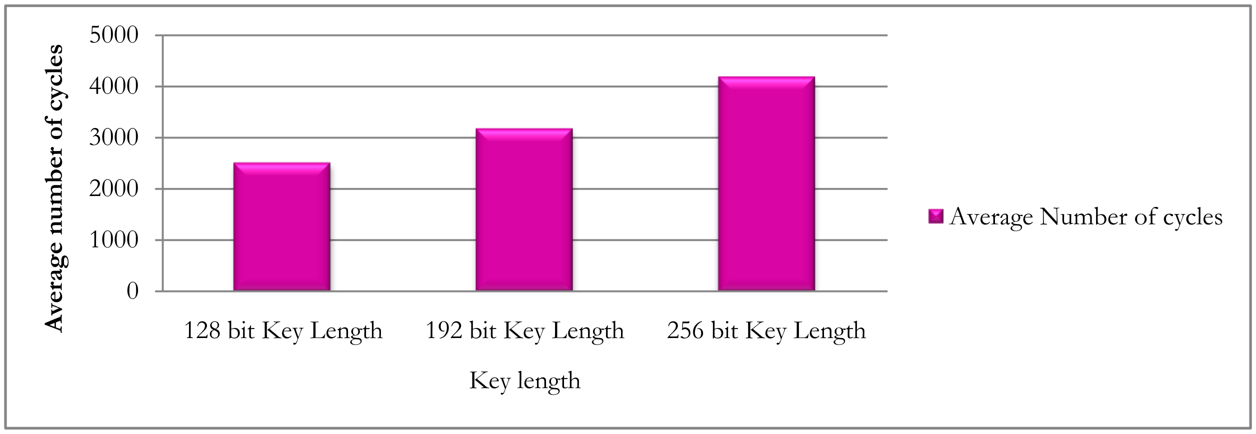


Figure 7.31: Average synchronization time (in cycle) for generating variable session key in KSOMSCT

From figure 7.31 it has been observed that if the length of the session key get increased then the increased of average synchronization steps is linear.

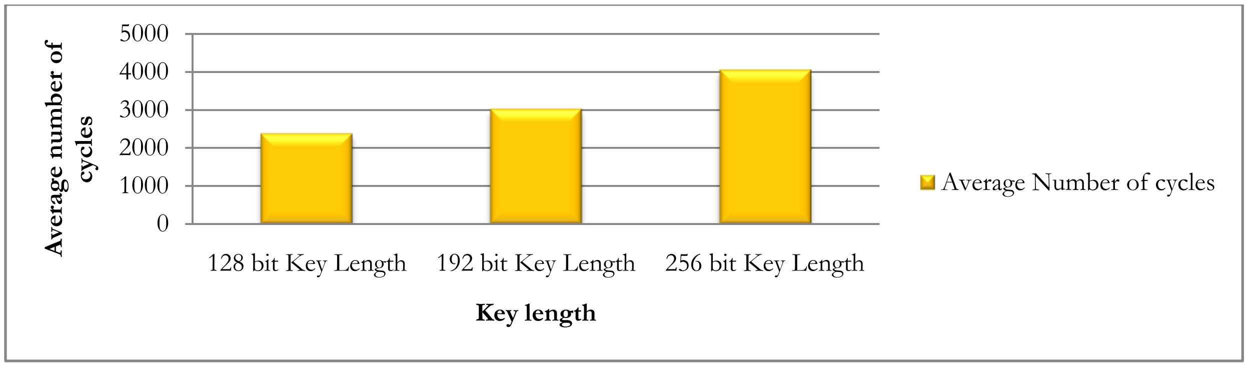


Figure 7.32: Average synchronization time (in cycle) for generating variable session key in DHLPSCT

From figure 7.32 it has been observed that if the length of the session key get increased then the increased of average synchronization steps is linear. This technique needs less amount of iterations than KSOMSCT.

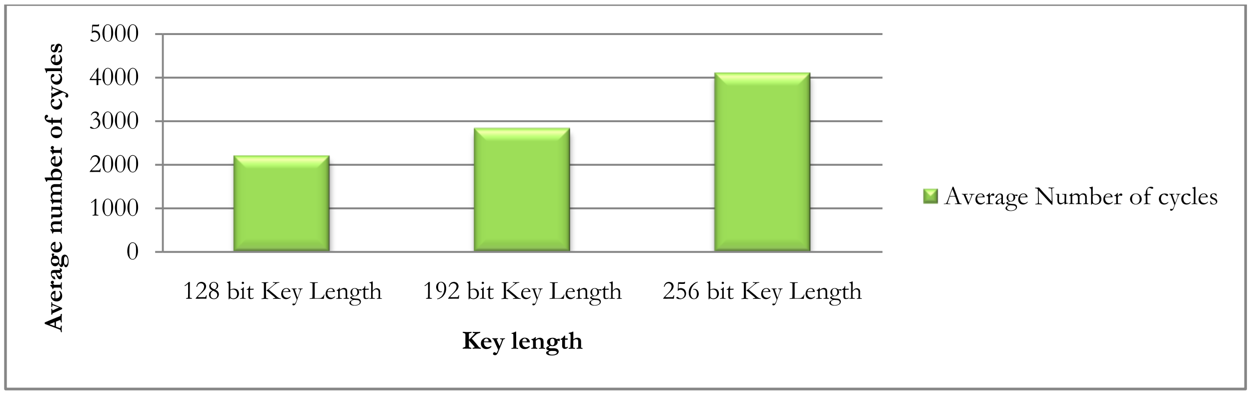


Figure 7.33: Average synchronization time (in cycle) for generating variable session key in CDHLPSCT

From figure 7.33 it has been observed that if the length of the session key get increased then the increased of average synchronization steps is linear. This technique needs less amount of iterations than DHLPSCT and KSOMSCT.

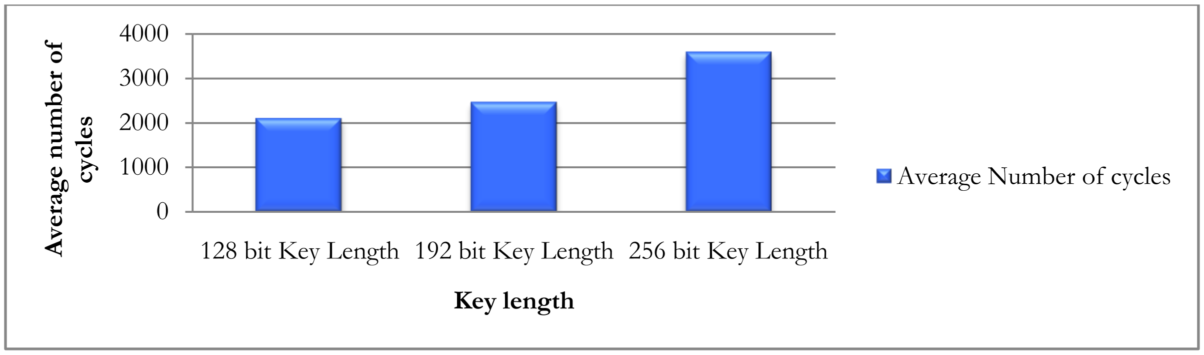


Figure 7.34: Average synchronization time (in cycle) for generating variable session key in CTHLPSCT

From figure 7.34 it has been observed that if the length of the session key get increased then the increased of average synchronization steps is linear. This technique needs less amount of iterations than CDHLPSCT, DHLPSCT and KSOMSCT.

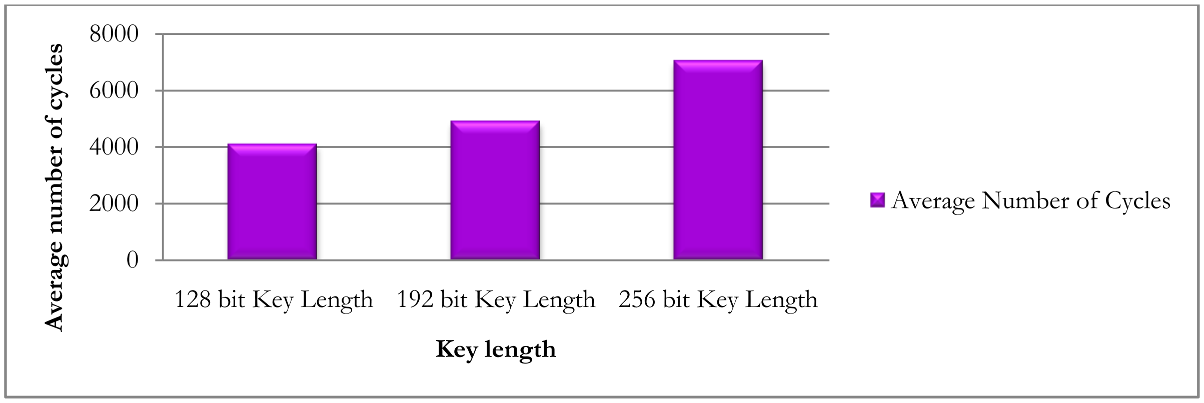


Figure 7.35: Average synchronization time (in cycle) for generating variable session key in CGTHLPSCT

From figure 7.35 it has been observed that if the length of the session key get increased then the increased of average synchronization steps is linear. This technique needs less amount of iterations than CTHLPSCT, CDHLPSCT, DHLPSCT and KSOMSCT.

7.3.7 Average Synchronization Time (in cycle) for Generating 128 bit Session Key using variable Weight range ($L = 5$ to 50) with fixed Neurons ($2-4-2$) in DHLPSCT, CDHLPSCT

Table: 7.49

Generation of 128 bit session key using variable weight range ($L = 5$ to 50) with fixed neurons ($2-4-2$) in DHLPSCT

L value	**Synchronization time in cycle**		
	Hebbian	**Anti-Hebbian**	**Random Walk**
5	209,94	233,36	241,49
10	620,08	612,74	602,29
15	1267,79	1058,23	1025,02
20	2178,19	1635,79	1563,84
25	3307,86	2322,13	2214,71
30	4723,51	3145,35	2962,94
35	6198,27	4126,45	3872,46
40	7857,09	5278,97	4387,29
45	9619,53	6594,18	6017,38
50	12798,72	8073,05	7277,61

In the above table 7.49 and following figure 7.36 the graph shows a trend towards increase in the synchronization steps as the range for weight values (L) increases in all three learning rules. For small L values Hebbian takes less synchronization steps than other two learning rules in the range of $2-4-2-5$ to $2-4-2-15$ but as the L value increases Hebbian rule takes more steps to synchronize than other two learning rules. Here, Anti-Hebbian rules takes less time than the other two learning rules in the range of $2-4-2-20$ to $2-4-2-30$. Random Walk outperform from $2-4-2-35$ and beyond that. The most vital findings is that if the synaptic depth i.e. weight range (L) is increased, the complexity of a successful attack grows exponentially, but there is only a polynomial increase of the effort needed to generate a key. So, increasing the L value security of the system can be increased.

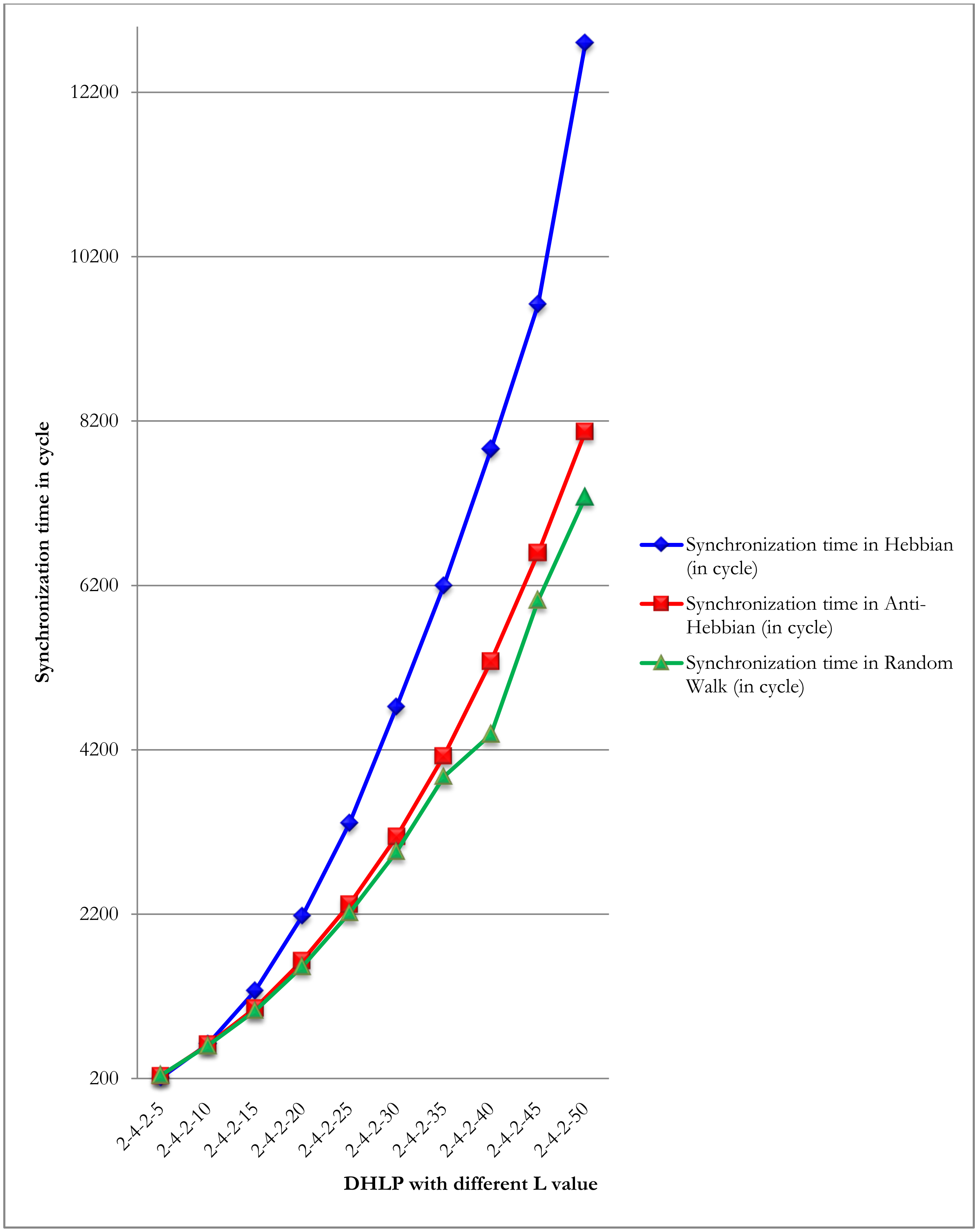


Figure 7.36: Generation of 128 bit session key using variable weight range ($L = 5$ to 50) with fixed neurons ($2 - 4 - 2$) in DHLPSCT

Table: 7.50

Generation of 128 bit session key using variable weight range ($L = 5$ to 50) with fixed neurons ($2 - 4 - 2$) in CDHLPSCT

L value	**Synchronization time in cycle**		
	Hebbian	**Anti-Hebbian**	**Random Walk**
5	192,06	212,58	223,86
10	602,52	589,27	581,63
15	1251,17	1038,50	1006,91
20	2160,21	1612,13	1547,52
25	3288,74	2305,84	2193,02
30	4702,97	3121,68	2945,38
35	6179,08	4107,34	3834,17
40	7842,35	5257,17	4839,62
45	9601,01	6572,03	5996,09
50	12776,16	8051,38	7261,10

In the above table 7.51 and following figure 7.37 the graph shows a trend towards increase in the synchronization steps as the range for weight values (L) increases in all three learning rules. For small L values Hebbian takes less synchronization steps than other two learning rules in the range of $2 - 4 - 2 - 5$ to $2 - 4 - 2 - 15$ but as the L value increases Hebbian rule takes more steps to synchronize than other two learning rules. Here, Anti-Hebbian rules takes less time than the other two learning rules in the range of $2 - 4 - 2 - 20$ to $2 - 4 - 2 - 30$. Random Walk outperform from $2 - 4 - 2 - 35$ and beyond that. The most vital findings is that if the synaptic depth i.e. weight range (L) is increased, the complexity of a successful attack grows exponentially, but there is only a polynomial increase of the effort needed to generate a key. So, increasing the L value security of the system can be increased.

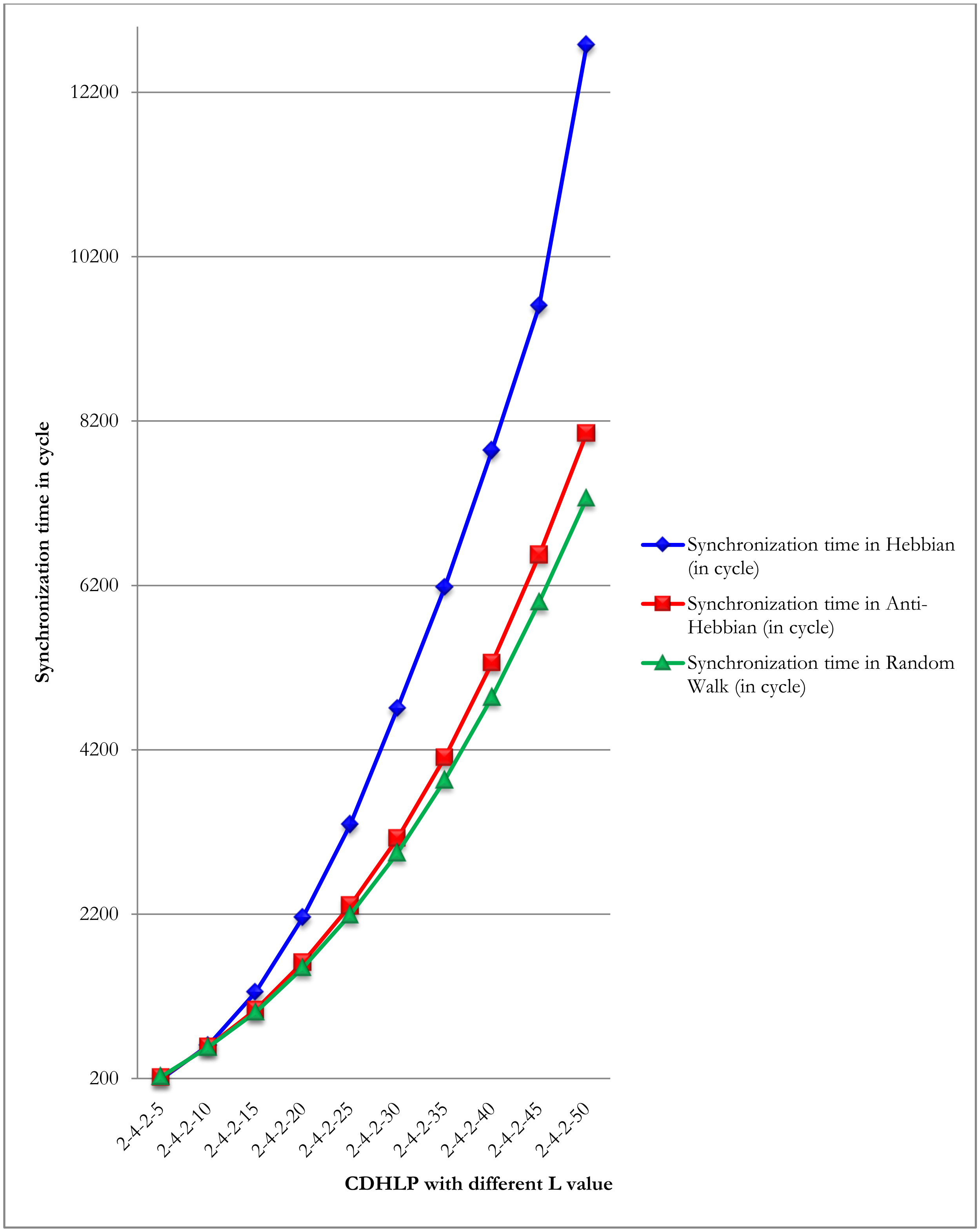


Figure 7.37: Generation of 128 bit session key using variable weight range ($L = 5$ to 50) with fixed neurons ($2 - 4 - 2$) in CDHLPSCT

7.3.8 Average Synchronization Time (in cycle) for Generating 128 bit Session Key using variable Weight range ($L = 5$ to 50) with fixed Neurons ($2 - 2 - 3 - 2$) in CTHLPSCT, CGTHLPSCT

Table: 7.51
Generation of 128 bit session key using variable weight range ($L = 5$ to 50) with fixed neurons ($2 - 2 - 3 - 2$) in CTHLPSCT

L value	**Average Synchronization time in cycle**		
	Hebbian	**Anti-Hebbian**	**Random Walk**
5	287,81	309,23	320,52
10	664,59	682,25	675.47
15	1356,06	1136,92	1104,92
20	2271,32	1703,26	1648,13
25	3407,18	2411,15	2302,83
30	4836,97	3220,86	3074,10
35	6319,72	4192,53	3959,65
40	7987,63	5341,28	4978,72
45	9753,02	6662,07	6147,49
50	12980,17	8137,47	7423,34

In the above table 7.51 and following figure 7.38 the graph shows a trend towards increase in the synchronization steps as the range for weight values (L) increases in all three learning rules. For small L values Hebbian takes less synchronization steps than other two learning rules in the range of $2 - 2 - 3 - 2 - 5$ to $2 - 2 - 3 - 2 - 15$ but as the L value increases Hebbian rule takes more steps to synchronize than other two learning rules. Here, Anti-Hebbian rules take fewer steps than the other two learning rules in the range of $2 - 2 - 3 - 2 - 8 - 20$ to $2 - 2 - 3 - 2 - 30$. Random Walk outperform from $3 - 2 - 2 - 8 - 35$ and beyond that. The most vital findings is that if the synaptic depth i.e. weight range (L) is increased, the complexity of a successful attack grows exponentially, but there is only a polynomial increase of the effort needed to generate a key. So, increasing the L value security of the system can be increased.

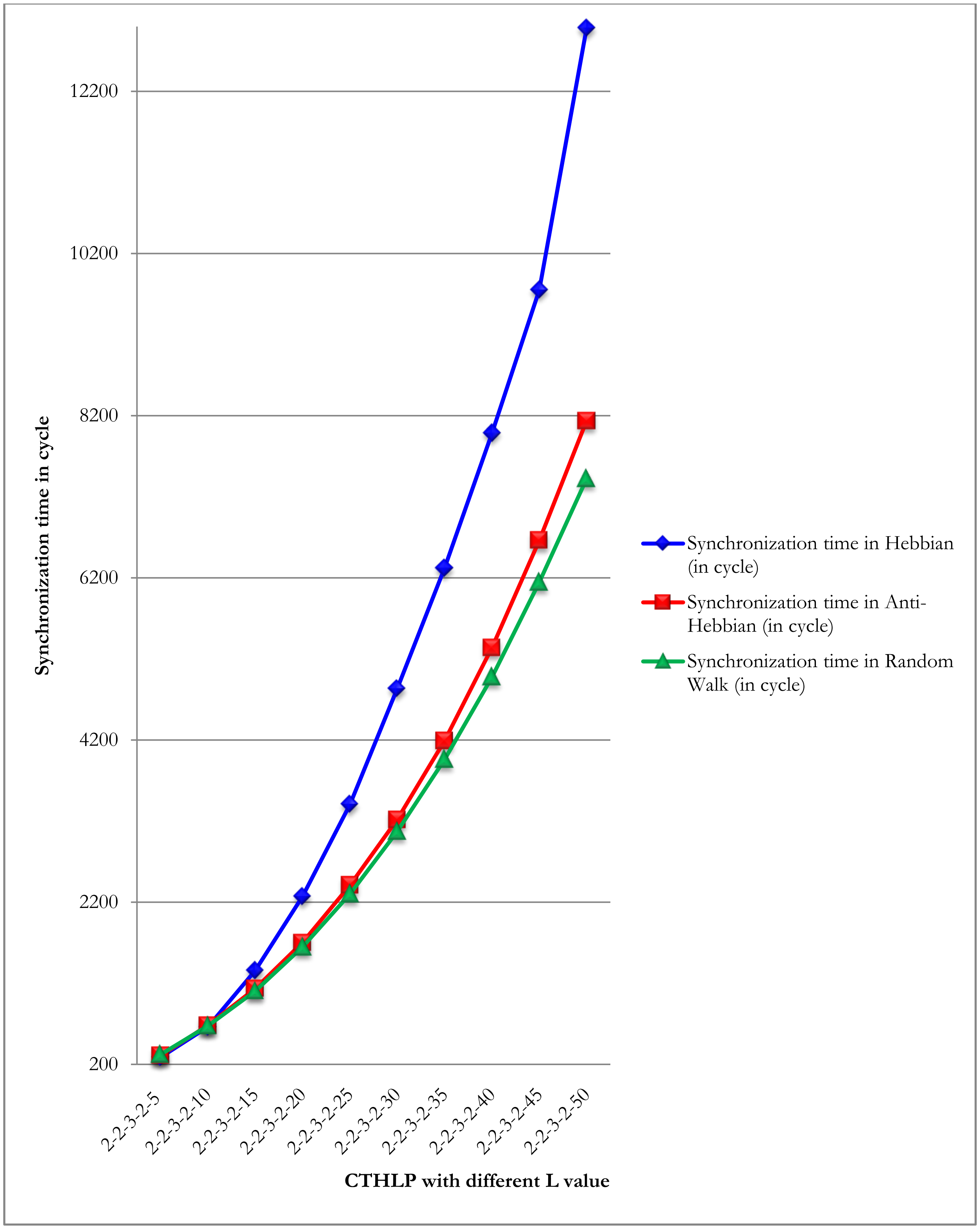


Figure: 7.38 Generation of 128 bit session key using variable weight range ($L = 5$ to 50) with fixed neurons ($2 - 2 - 3 - 2$) in CTHLPSCT

Table: 7.52
Generation of 128 bit session key using variable weight range ($L = 5$ to 50) with fixed neurons ($2 - 2 - 3 - 2$) in CGTHLPSCT

L value	**No. of CTHLP Participated at Group Session Key Generation**	**Average Synchronization time in cycle**		
		Hebbian	**Anti-Hebbian**	**Random Walk**
5	3	259,91	279,26	289,45
10	4	1325,61	1302,06	1289,12
15	5	1356,06	1136,92	1104,92
20	6	2271,32	1703,26	1648,13
25	7	3407,18	2411,15	2302,83
30	8	4836,97	3220,86	3074,10
35	9	6319,72	4192,53	3959,65
40	10	7987,63	5341,28	4978,72
45	11	9753,02	6662,07	6147,49
50	12	12980,17	8137,47	7423,34

In the above table 7.52 and following figure 7.39 the graph shows a trend towards increase in the synchronization steps as the range for weight values (L) increases in all three learning rules. For small L values Hebbian takes less synchronization steps than other two learning rules in the range of $2 - 2 - 3 - 2 - 5$ to $2 - 2 - 3 - 2 - 15$ but as the L value increases Hebbian rule takes more steps to synchronize than other two learning rules. Here, Anti-Hebbian rules take fewer steps than the other two learning rules in the range of $2 - 2 - 3 - 2 - 8 - 20$ to $2 - 2 - 3 - 2 - 30$. Random Walk outperform from $3 - 2 - 2 - 8 - 35$ and beyond that. The most vital findings is that if the synaptic depth i.e. weight range (L) is increased, the complexity of a successful attack grows exponentially, but there is only a polynomial increase of the effort needed to generate a key. So, increasing the L value security of the system can be increased.

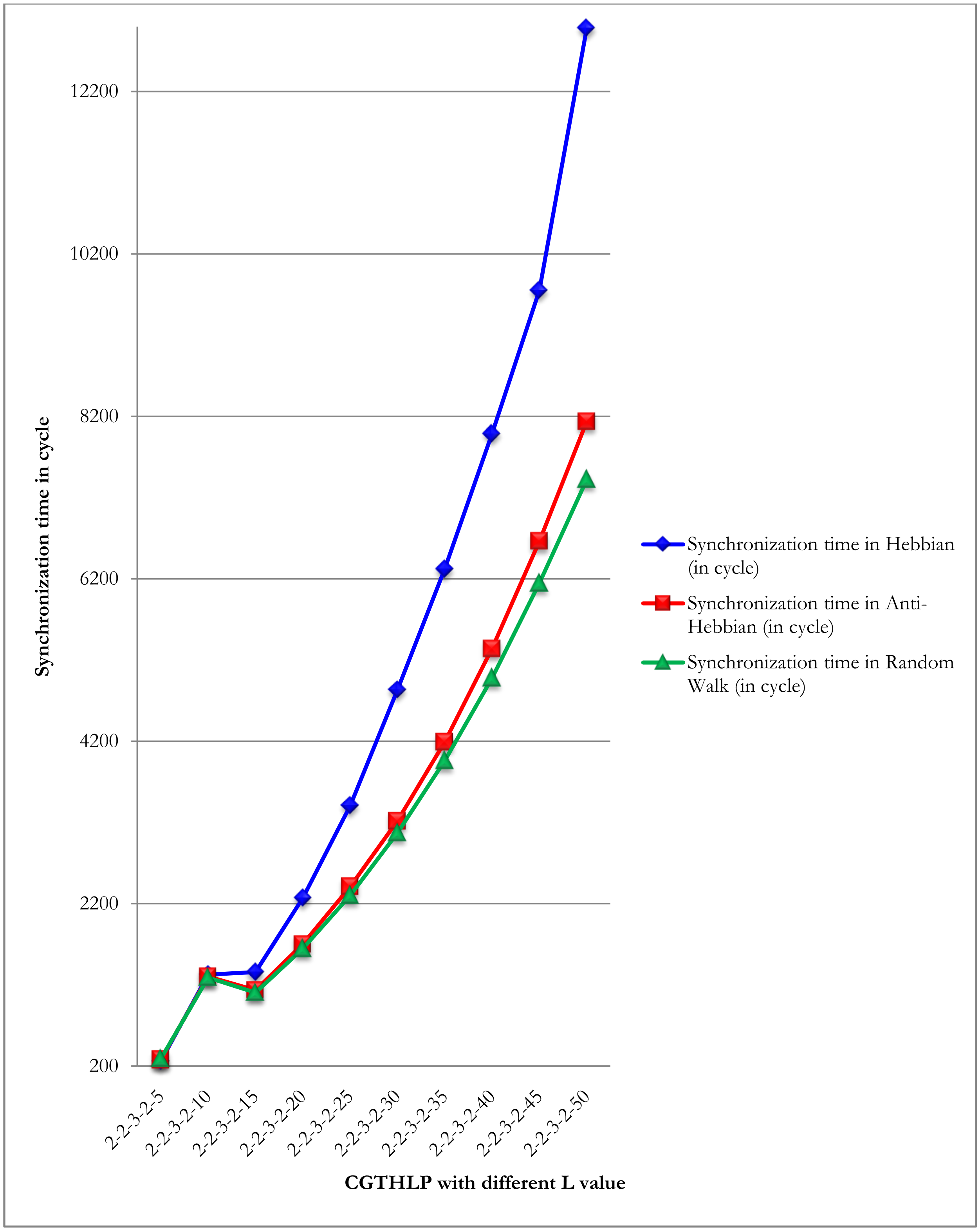


Figure 7.39: Generation of 128 bit session key using variable weight range ($L = 5$ to 50) with fixed neurons ($2 - 2 - 3 - 2$) in CGTHLPSCT

7.3.9 Average Synchronization Time (in cycle) for Generating 128 bit Session Key using Hebbian learning rule with variable Weight range ($L = 5$ to 50) and fixed Neurons ($2 - 4 - 2$) in DHLPSCT, CDHLPSCT

Table: 7.53

Generation of 128 bit session key using Hebbian learning rule with variable weight range ($L = 5$ to 50) and fixed neurons ($2 - 4 - 2$) in DHLPSCT

L value	**Synchronization time in cycle in 1000 runs**		
	Min	**Max**	**Average**
5	176,28	243,06	209,94
10	541,07	699,09	620,08
15	1217,16	1318,42	1267,79
20	2137,08	2219,30	2178,19
25	3229,98	3385,74	3307,86
30	4706,05	4740,97	4723,51
35	6176,41	6220,13	6198,27
40	7814,52	7899,66	7857,09
45	9605,48	9633,58	9619,53
50	12793,26	12804,18	12798,72

Table 7.53 and figure 7.40 shows the minimum, maximum and average synchronization steps of $2 - 4 - 2$ DHLPSCT using different weight range and Hebbian learning rule. From the graph presented in the figure 7.40 it has been conclude that for the higher value of L synchronization steps also get increased.

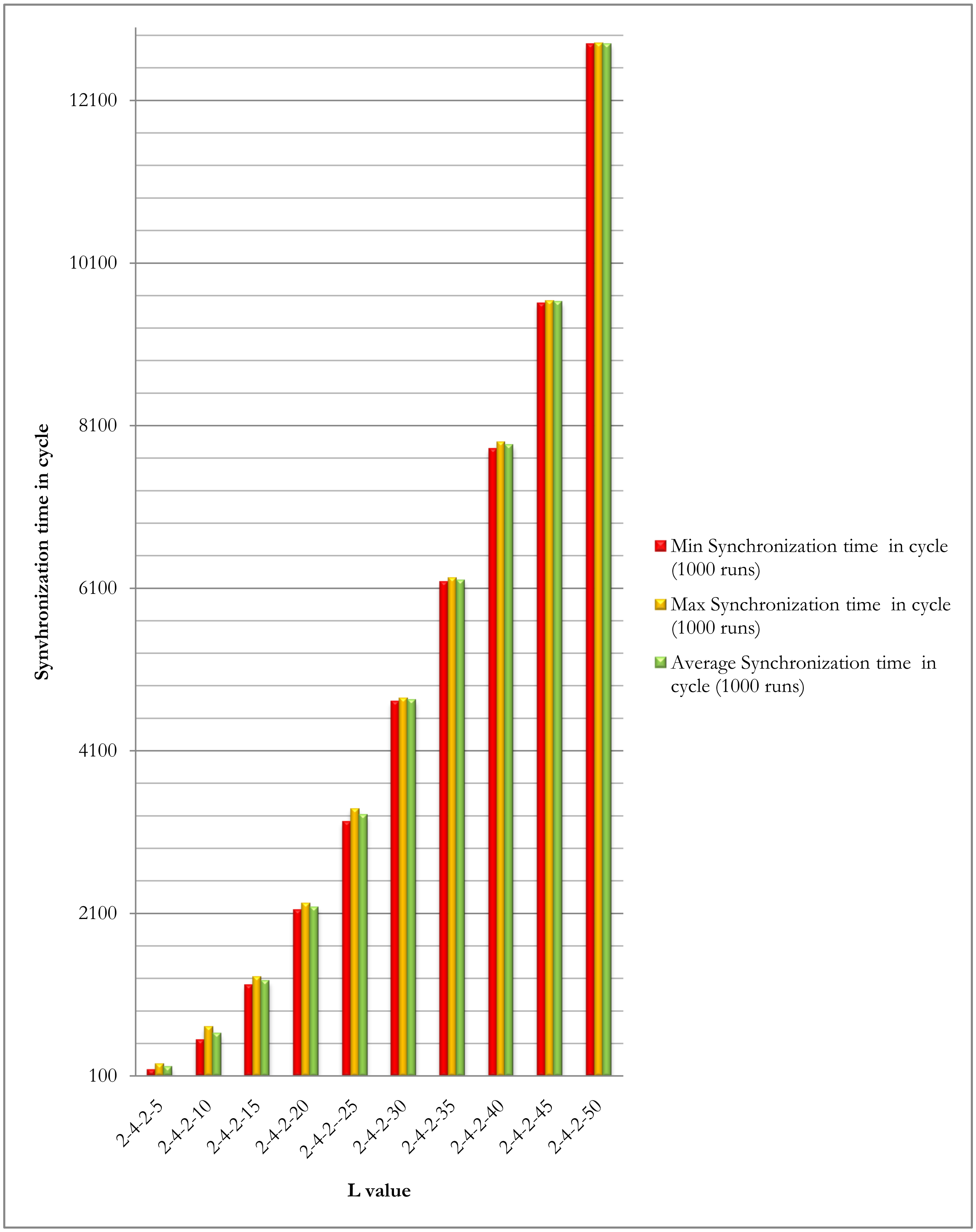


Figure 7.40: Generation of 128 bit session key using Hebbian learning rule with variable weight range ($L = 5$ to 50) and fixed neurons ($2 - 4 - 2$) in DHLPSCT

Table: 7.54

Generation of 128 bit session key using Hebbian learning rule with variable weight range $(L = 5$ to $50)$ and fixed neurons $(2-4-2)$ in CDHLPSCT

L value	**Synchronization time in cycle in 1000 runs**		
	Min	**Max**	**Average**
5	158,04	226,08	192,06
10	527,81	677,23	602,52
15	1193,20	1309,14	1251,17
20	2112,05	2208,37	2160,21
25	3208,87	3368,61	3288,74
30	4675,98	4729,96	4702,97
35	6153,09	6205,07	6179,08
40	7791,17	7893,53	7842,35
45	9584,01	9618,01	9601,01
50	12776,03	12785,29	12776,16

Table 7.54 and figure 7.41 shows the minimum, maximum and average synchronization steps of $2-4-2$ CDHLPSCT using different weight range and Hebbian learning rule. From the graph presented in the figure 7.41 it has been conclude that for the higher value of L synchronization steps also get increased.

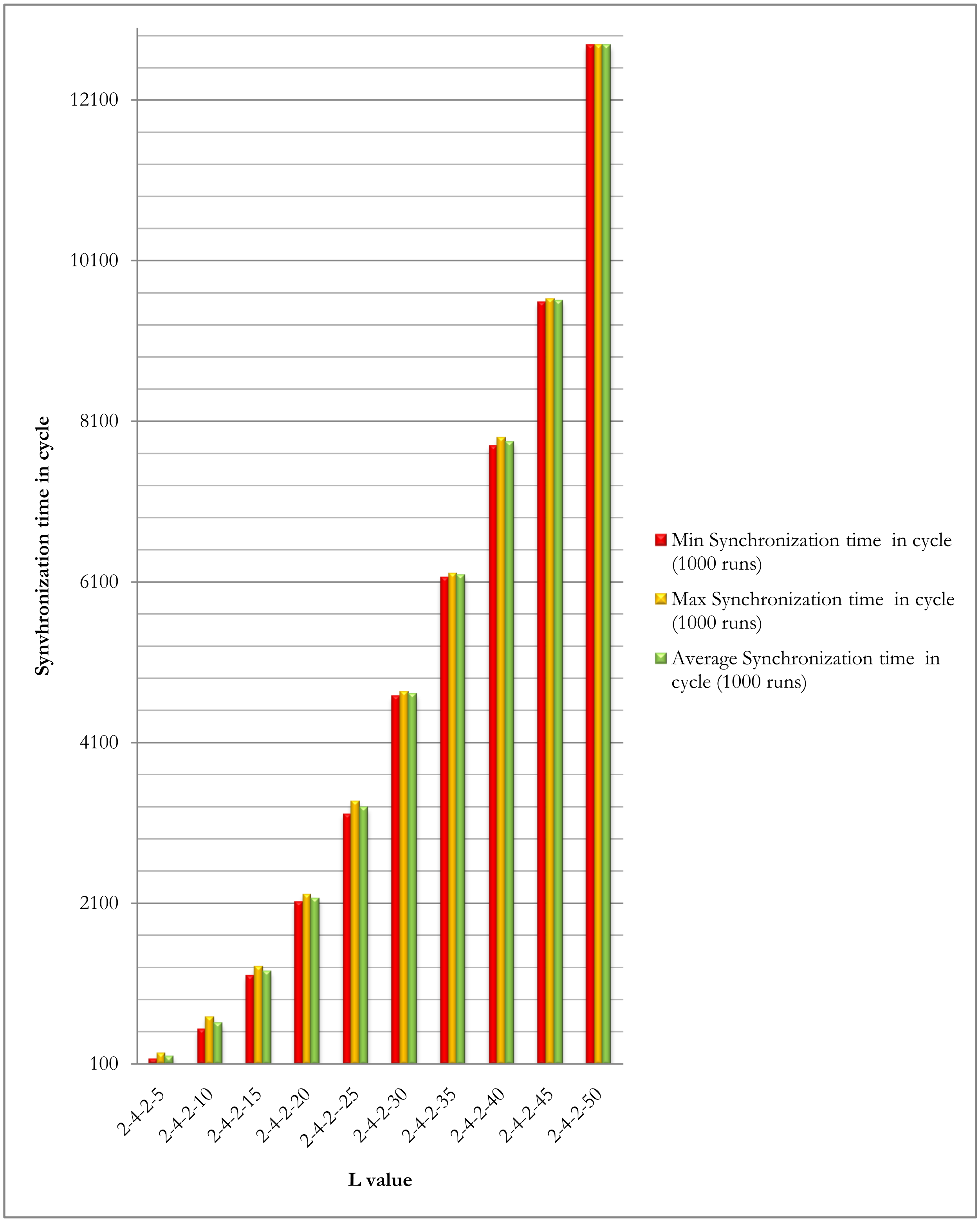


Figure 7.41: Generation of 128 bit session key using Hebbian learning rule with variable weight range ($L = 5$ to 50) and fixed neurons $(2 - 4 - 2)$ in CDHLPSCT

7.3.10 Average Synchronization Time (in cycle) for Generating 128 bit Session Key using Hebbian learning rule with variable Weight range $(L = 5 \text{ to } 50)$ and fixed Neurons $(2-2-3-2)$ in CTHLPSCT, CGTHLPSCT

Table: 7.55

Generation of 128 bit session key using Hebbian learning rule with variable weight range $(L = 5 \text{ to } 50)$ and fixed neurons $(2-2-3-2)$ in CTHLPSCT

CTHLP Size (128 bit Key)	**Synchronization time in cycle (1000 runs)**		
	Min	**Max**	**Average**
2-2-3-2-5	262,87	312,75	287,81
2-2-3-2-10	512,26	816,92	664,59
2-2-3-2-15	1163,03	1549,09	1356,06
2-2-3-2-20	2065,45	2477,19	2271,32
2-2-3-2-25	3147,07	3647,29	3407,18
2-2-3-2-30	4465,98	5207,96	4836,97
2-2-3-2-35	6041,86	6597,58	6319,72
2-2-3-2-40	7655,71	8319,55	7987,63
2-2-3-2-45	8479,03	11027,01	9753,02
2-2-3-2-50	11792,11	14168,23	12980,17

Table 7.55 and figure 7.42 shows the minimum, maximum and average synchronization steps of $2-2-3-2$ CTHLPSCT using different weight range and Hebbian learning rule. From the graph presented in the figure 7.42 it has been conclude that for the higher value of L synchronization steps also get increased.

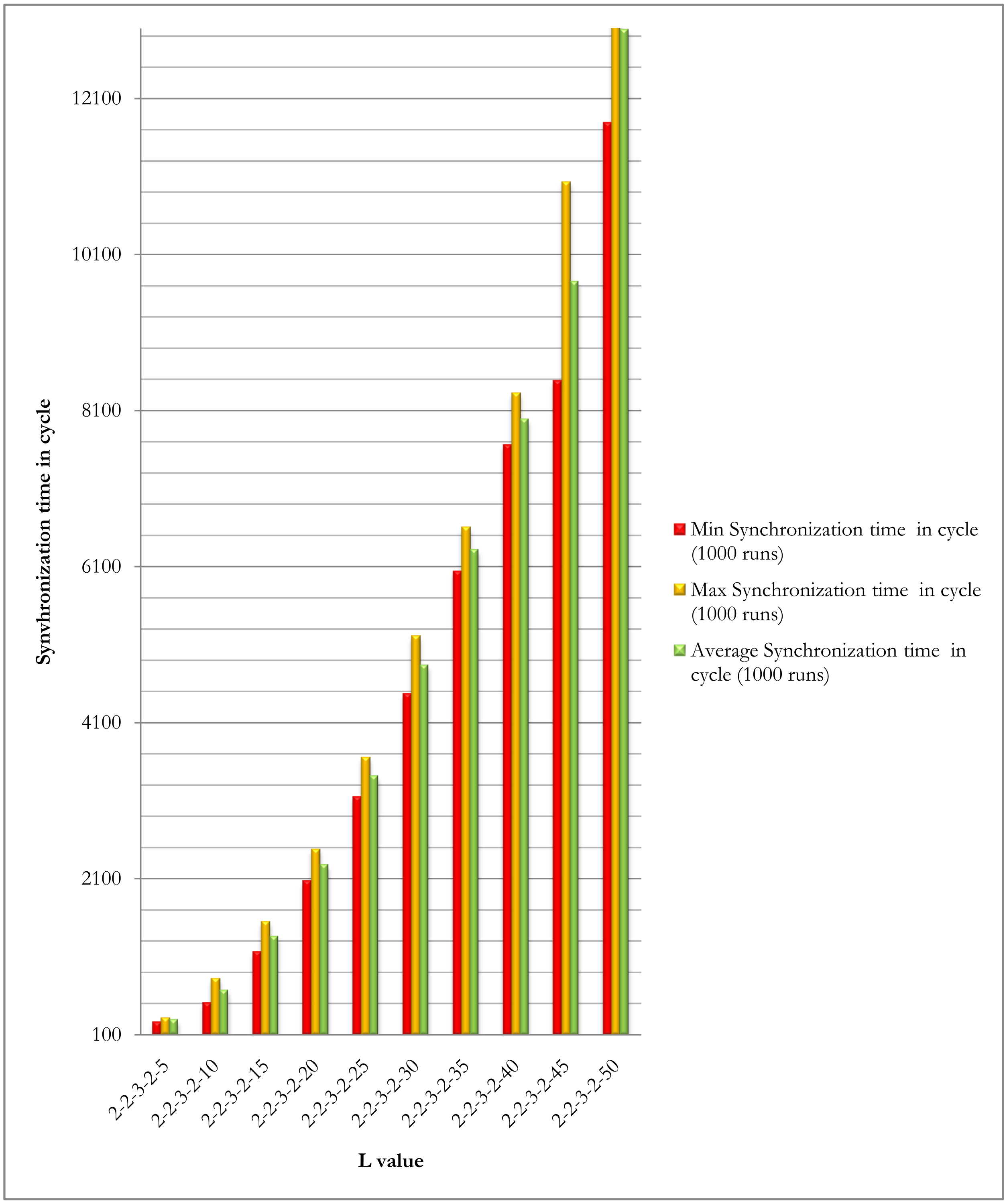


Figure 7.42: Generation of 128 bit session key using Hebbian learning rule with variable weight range ($L = 5$ to 50) and fixed neurons ($2 - 2 - 3 - 2$) in CTHLPSCT

Table: 7.56

Generation of 128 bit session key using Hebbian learning rule with variable weight range $(L = 5 \text{ to } 50)$, variable group size with fixed neurons $(2 - 2 - 3 - 2)$ in CGTHLPSCT

L value	**No. of CTHLP Participated at Group Session Key Generation**	**Synchronization time in cycle (1000 runs)**		
		Min	**Max**	**Average**
5	4	237,39	282,44	259,91
10	5	1542,05	2450,76	1996,40
15	6	4877,53	6496,60	5856,76
20	7	11250,63	13493,40	12372,02
25	8	21276,66	19925,99	20601,32
30	9	36298,64	42329,31	39313,98
35	10	57654,00	62956,91	60305,45
40	11	84212,81	91515,05	87863,95
45	12	105959,98	137801,35	121880,60
50	13	165435,76	198771,20	182103,50

Table 7.56 and figure 7.43 shows the minimum, maximum and average synchronization steps of $2 - 2 - 3 - 2$ CGTHLPSCT using different weight range and Hebbian learning rule. From the graph presented in the figure 7.43 it has been conclude that for the higher value of L synchronization steps also get increased.

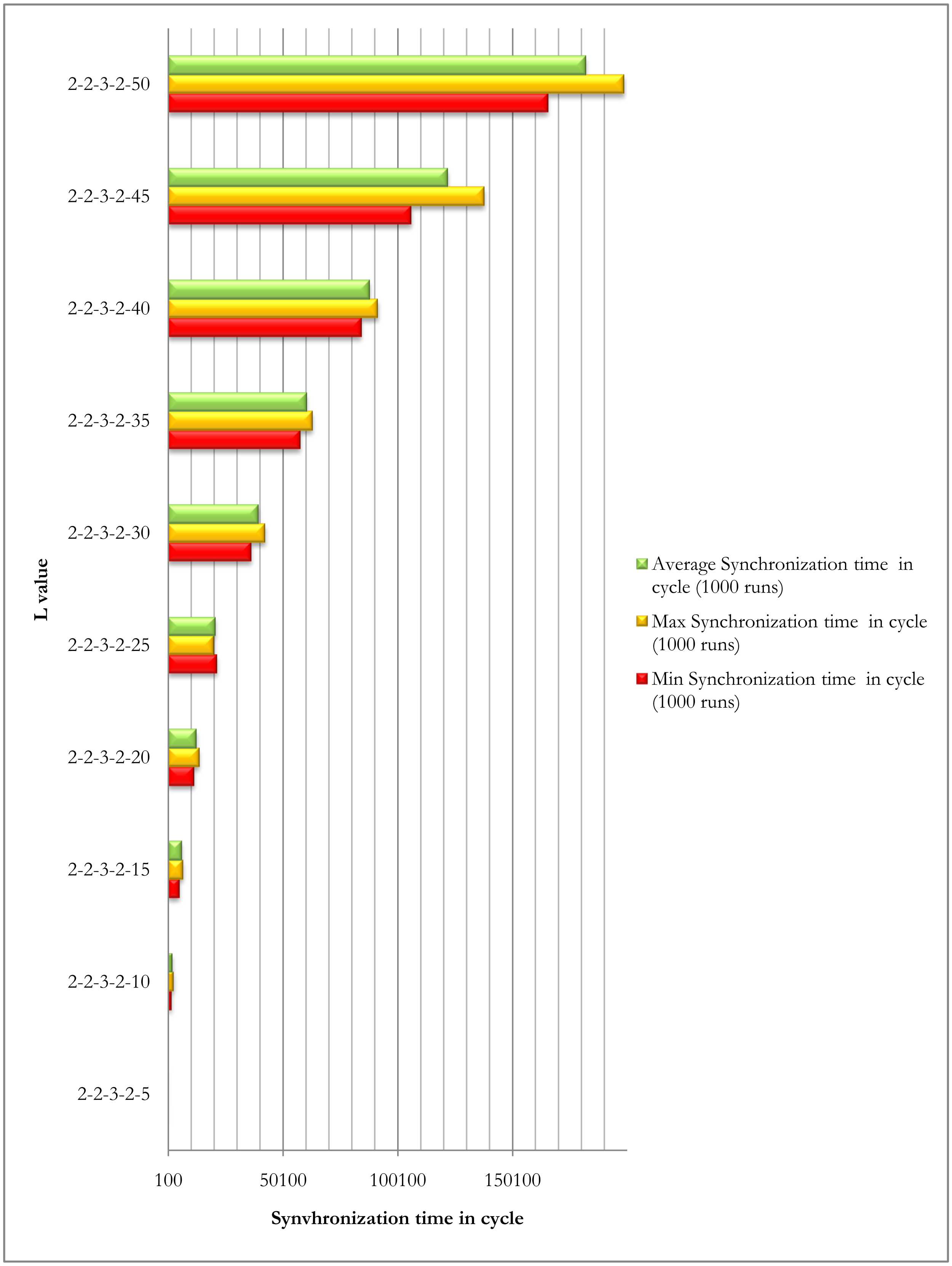


Figure 7.43: Generation of 128 bit session key using Hebbian learning rule with variable weight range ($L = 5$ to 50) and variable group size with fixed neurons $(2 - 2 - 3 - 2)$ in CGTHLPSCT

7.3.11 Comparison of memory heap used in both proposed and existing techniques for generation of 128 bit session key

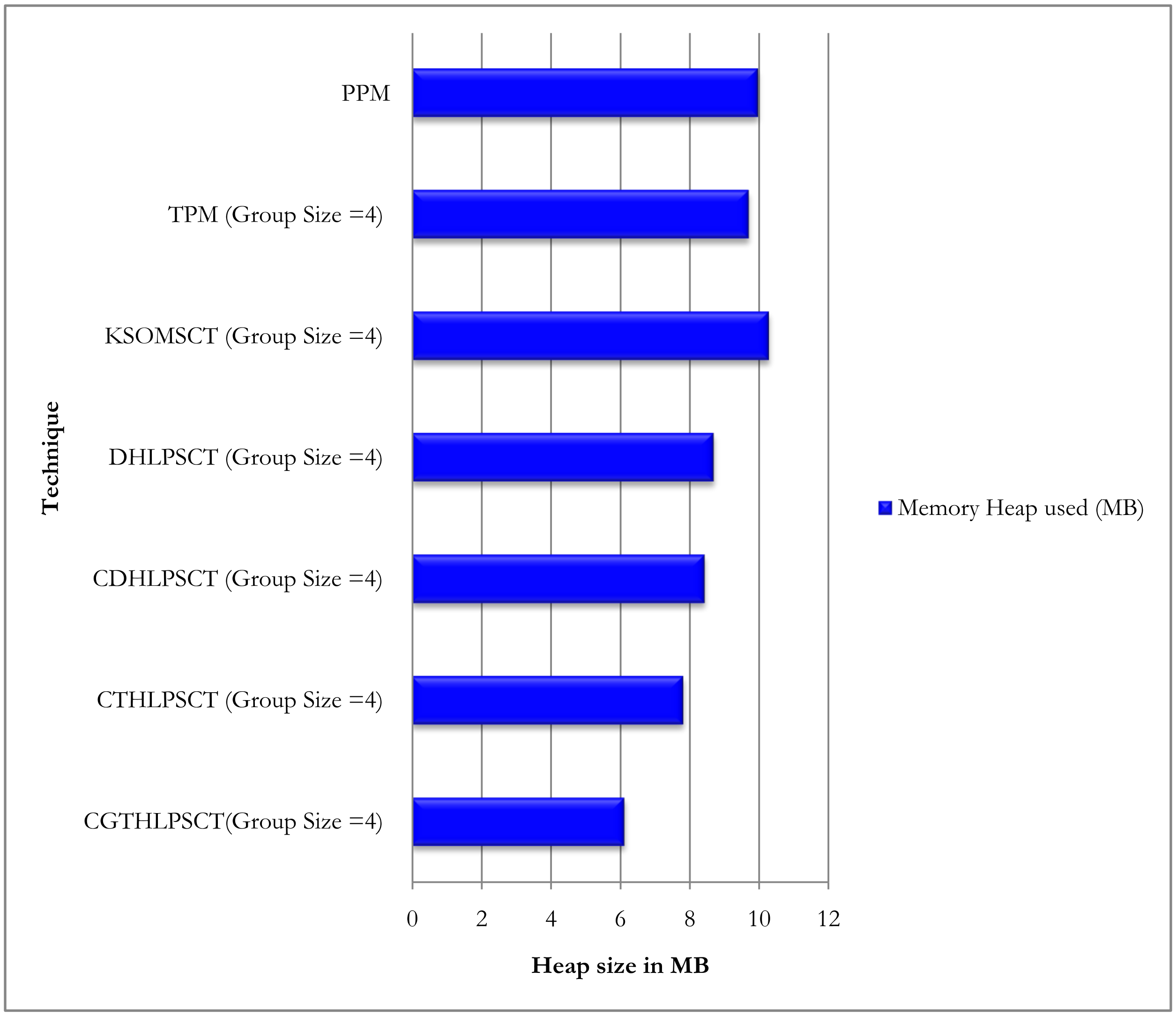


Figure 7.44: Comparisons of memory used to generate 128 bit session key

From the figure 7.44 it has been shown that in group synchronization phase CGTHLPSCT consumes less amount of memory compared to other techniques because it needs only $nlog(n-1)$ amount of synchronizations compared to $n(n-1)$ synchronizations steps in others.

7.3.12 Comparison of relative time spent in GC to generate 128 bit session key using both proposed and existing techniques

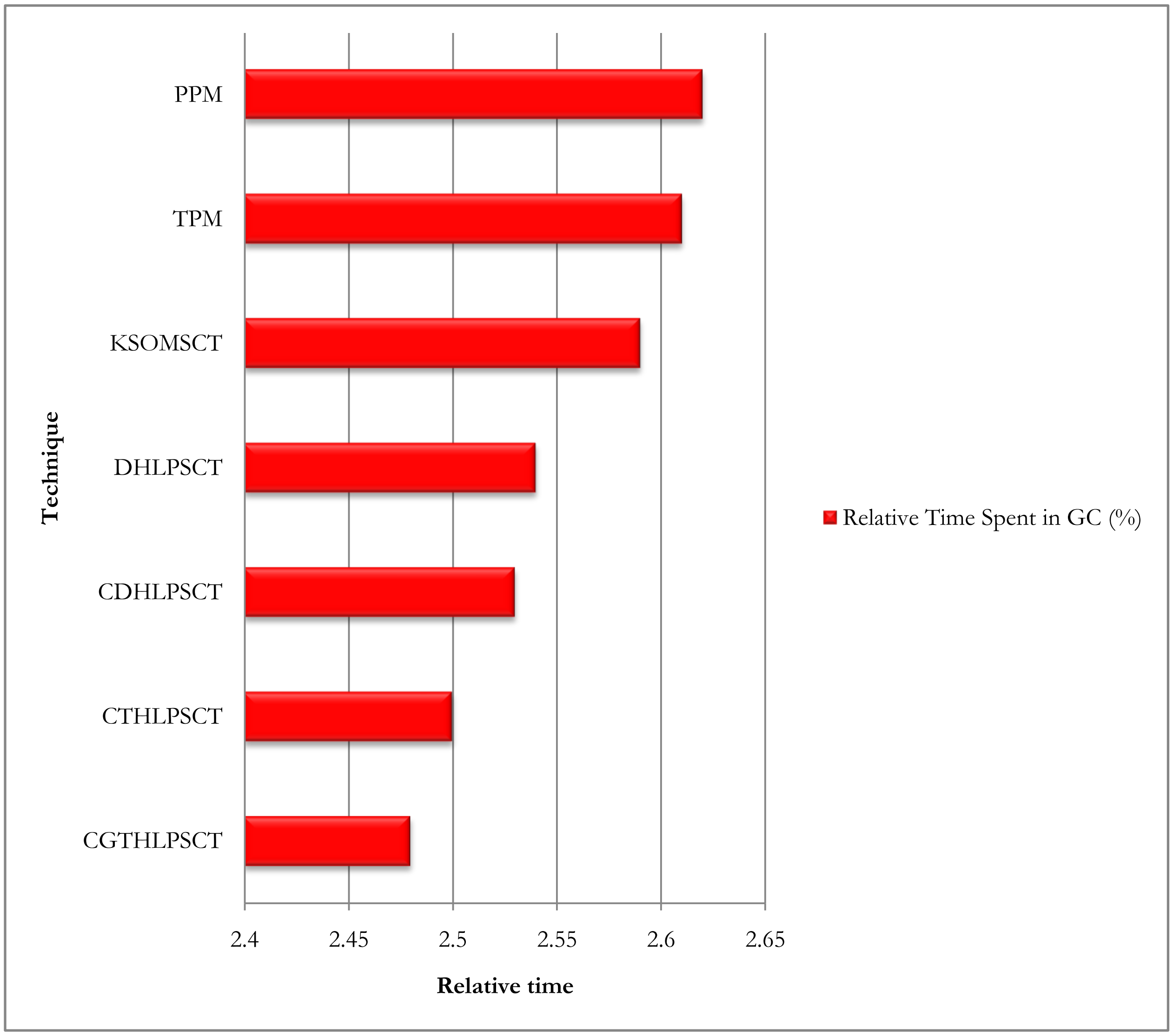


Figure 7.45: Comparisons of relative time spent in GC to generate 128 bit session key

From the figure 7.45 it has been shown that increasing order sequence of relative time spent in GC in group synchronization phase is CGTHLPSCT, CTHLPSCT, CDHLPSCT, DHLPSCT, KSOMSCT, TPM and PPM.

7.3.13 Comparisons of thread required to generate 128 bit session key using both proposed and existing techniques

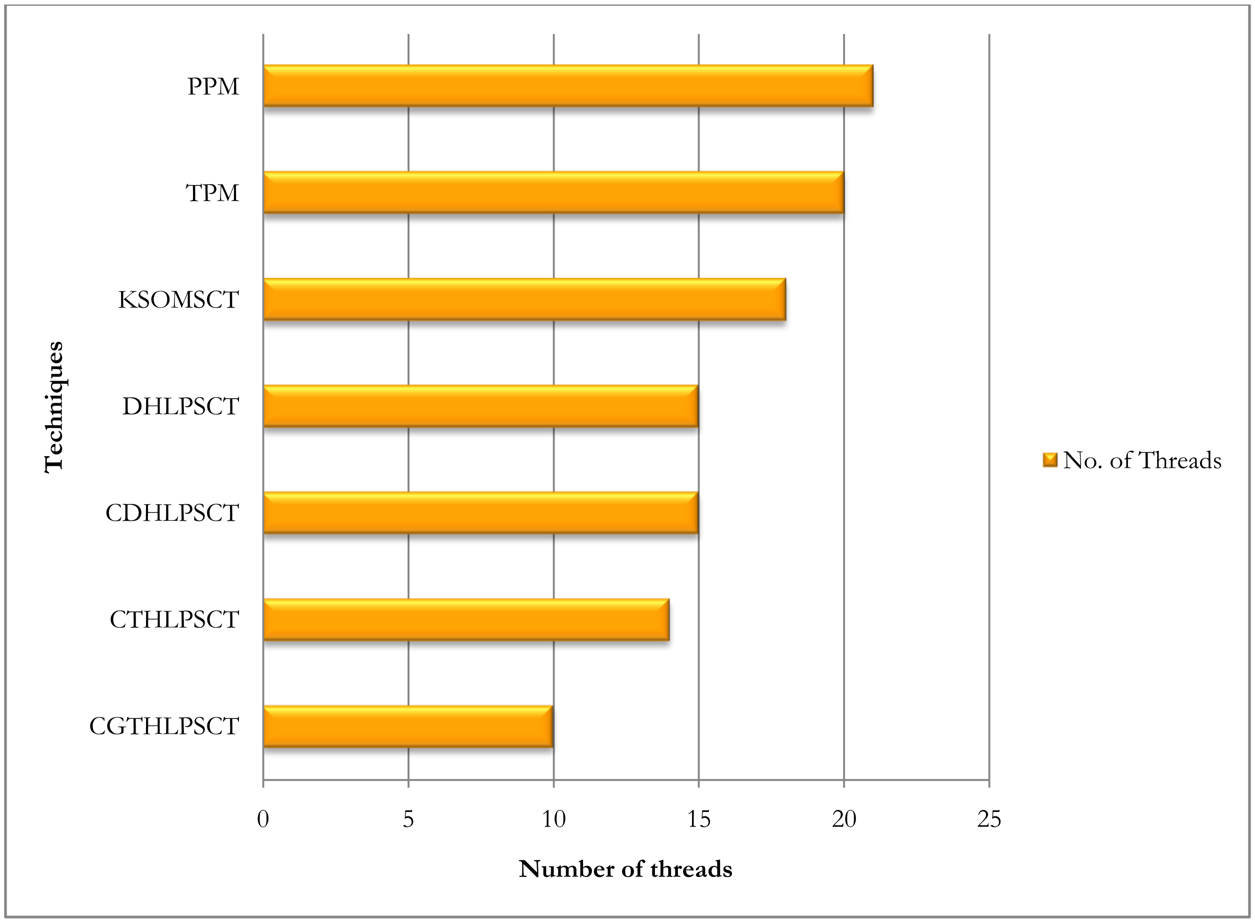


Figure 7.46: Comparisons of number of threads required generating 128 bit session key

From the figure 7.46 it has been shown that increasing order sequence of number of thread required in group synchronization phase is CGTHLPSCT, CTHLPSCT, CDHLPSCT, DHLPSCT, KSOMSCT, TPM and PPM.

7.3.14 Analysis of dimension of KSOMSCT vs. average number of iterations

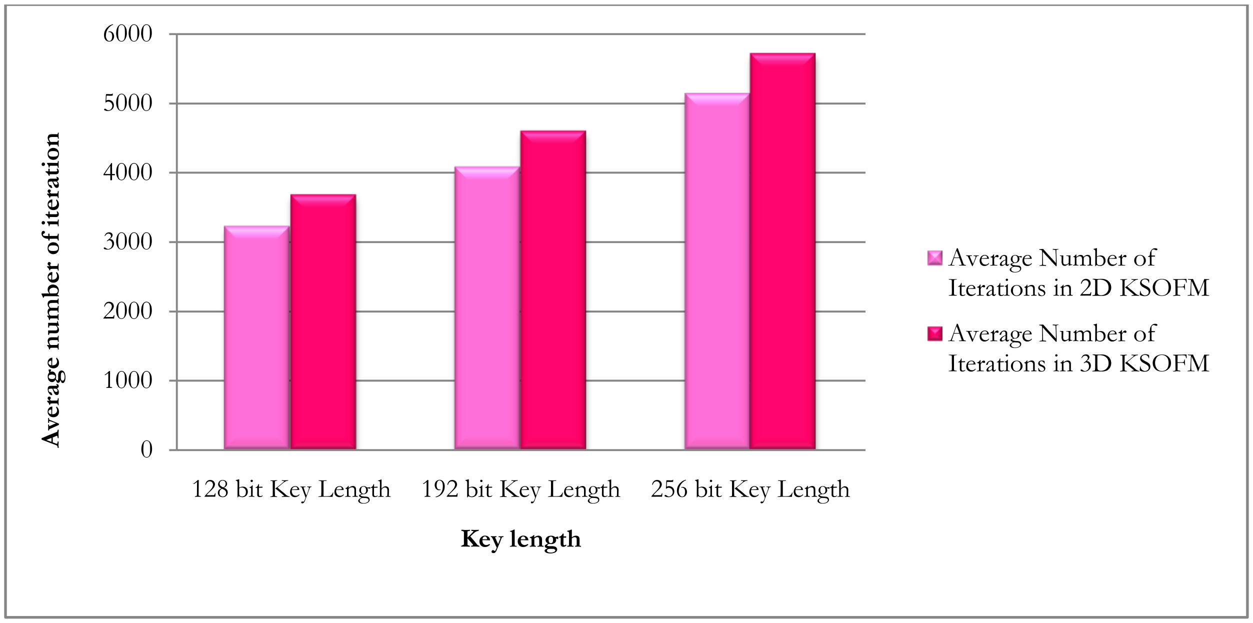


Figure 7.47: KSOMSCT dimension vs. average number of iterations

Figure 7.47 shows the average number of iterations to be needed for generating $128, 192$ and 256 bit session key in $2D$ and $3D$ KSOMSCT. The above figure depicts that $3D$ KSOMSCT takes more iterations to train the map in compared to $2D$ KSOMSCT. So, the energy consumption is more in $3D$ KSOMSCT than $2D$. For this reason $2D$ KSOMSCT is the best alternative in wireless communication where resource constrains (in terms of energy, memory) is a vital issues for generation of session key.

7.3.15 Analysis of number of generations vs. average fitness value in Simulated Annealing guided fittest keystream generation in DHLPSCT

Table 7.57 and figure 7.48 depicts the average fitness values of different number of generations. Table shows four set of entries where 40, 60 80, 100 numbers of generations are considered. It is observed from the table that increasing the number of generation also increased the fitness values in average.

Table: 7.57
Average of fitness values in SA

Number of Generations	Average of fitness values
40	35.1486
60	35.8713
80	36.2581
100	36.7316

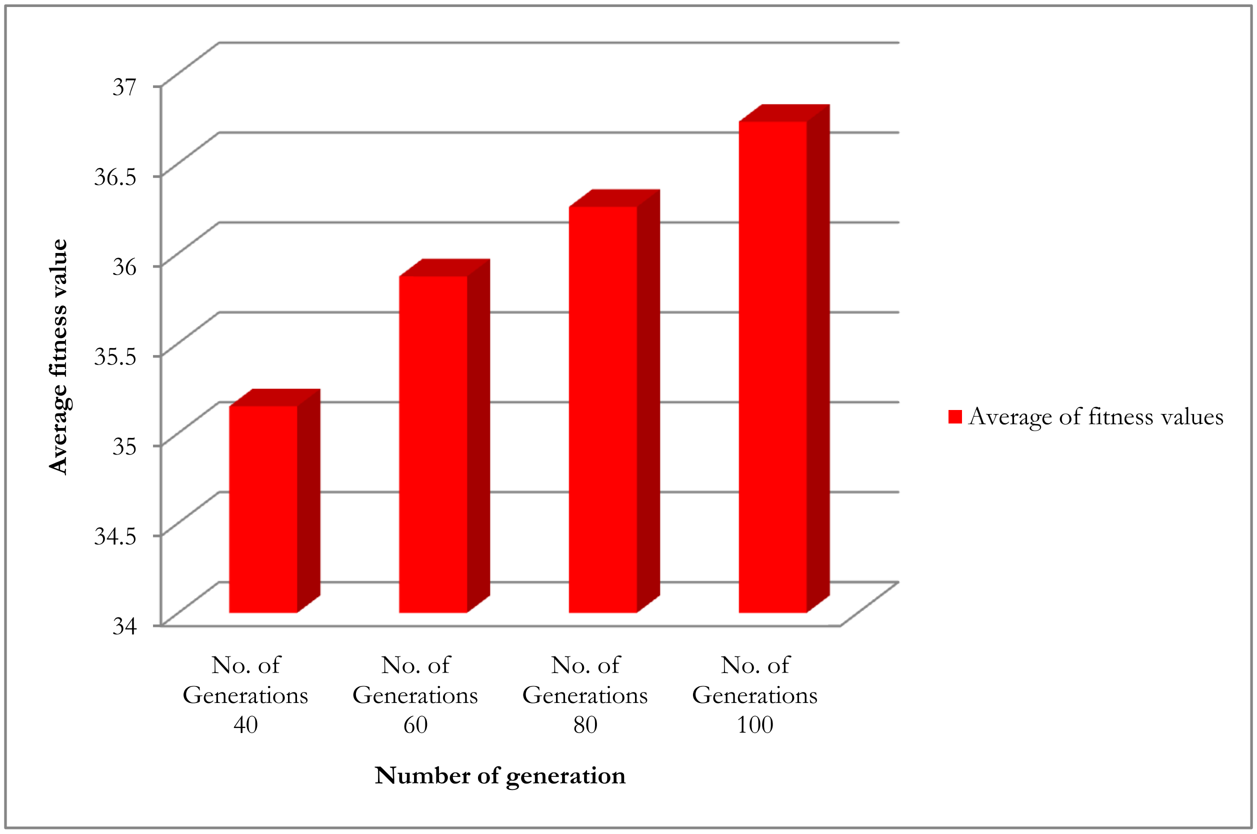


Figure 7.48: Number of generation vs. average of fitness values in SA guided fittest keystream generation technique

Table 7.58 tabulated the best fitness values of fifty different runs of SA. The average fitness value of fifty runs is 34.89712. The proposed SA based encryption/decryption technique has been run fifty different times on a identical source file and each time the fitness value calculated by the SA based proposed technique is tabulated to show that each time a completely random SA based keystream is generated with different fitness value. These generated fitness values confirms the generation of random SA based keystream in each different run of the technique.

Table: 7.58
List of best fitness values in 50 different runs of SA

Iteration	**Fitness Value**
1	31.4377
2	37.5723
3	34.4687
4	36.3263
5	31.8379
6	39.5962
7	41.6294
8	32.6138
9	28.5972
10	32.2379
. . .	. . .
40	37.6817
41	36.8629
42	38.6328
43	36.1684
44	38.8292
45	32.9716
46	35.4094
47	29.6962
48	42.7356
49	34.8038
50	37.5792

7.3.16 Analysis of number of generations vs. average fitness value in Genetic Algorithm guided fittest keystream generation in CDHLPSCT

Table 7.59 and figure 7.49 represents the average fitness values of different number of generations. Table shows four set of entries where 40, 60 80, 100 numbers of generations are considered. It is observed from the table that increasing the number of generation also increased the fitness values in average.

Table: 7.59
Average of fitness values in GA

Number of Generations	Average of fitness values
40	35.8350
60	36.2346
80	36.9535
100	38.5472

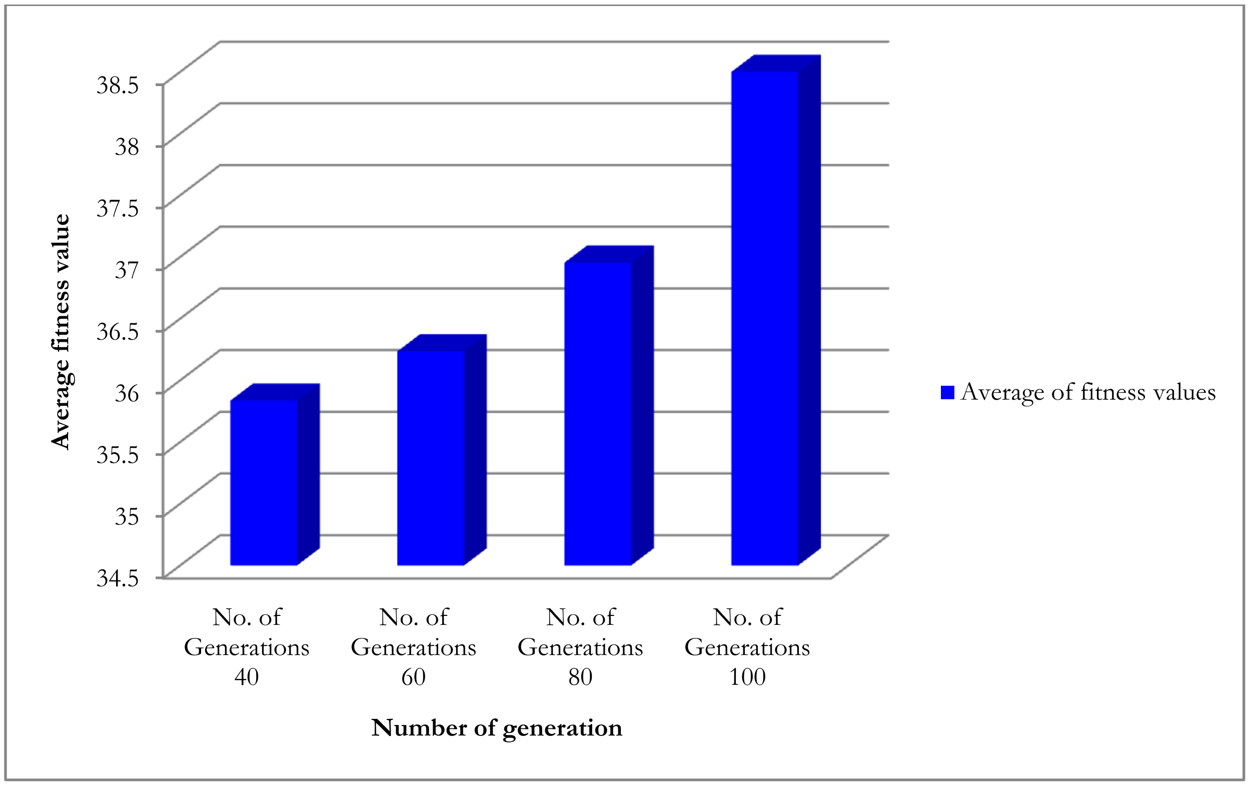


Figure 7.49: Number of generation vs. average of fitness values in GA guided fittest keystream generation technique

Table 7.60 tabulated the best fitness values of fifty different runs of GA. The average fitness value of fifty runs is 37.116146. The proposed GA based encryption/decryption technique has been run fifty different times on a identical source file and each time the fitness value calculated by the GA based proposed technique is tabulated to show that each time a completely random GA based keystream is generated with different fitness value. These generated fitness values confirms the generation of random GA based keystream in each different run of the technique.

Table: 7.60
List of best fitness values in 50 different runs of GA

Iteration	**Fitness Value**
1	34.1069
2	39.4297
3	32.9237
4	38.3263
5	29.5436
6	44.0764
7	43.3057
8	32.1490
9	31.4927
10	37.5192
. . .	. . .
40	30.2973
41	43.0401
42	27.5291
43	49.6033
44	25.0072
45	39.3781
46	32.6194
47	40.2051
48	42.6397
49	37.1893
50	50.2985

7.3.17 Comparisons of length of plain text vs. Keystream storage between proposed and existing techniques

Table 7.61 and figure 7.50 shows the comparisons of length of plan text vs. keystream storage between proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing AES, RC4, Vernam Cipher.

Table: 7.61
Comparisons of length of plan text vs. keystream storage between proposed and existing techniques

Length of Plaintext	Key Storage (KSOM-SCT)	Key Storage (DHLP-SCT)	Key Storage (CDHLP-SCT)	Key Storage (CTHLP-SCT)	Key Storage (CGTHLP-SCT)	Key Storage (AES)	Key Storage (RC4)	Key Storage (Vernam Cipher)
64	-	128	128	15	15	128	52	60
120	-	128	128	17	15	128	106	120
500	-	128	128	20	15	128	437	500
1000	-	128	128	22	20	128	913	1000

In KSOMSCT based technique fractal triangle guided encryption/decryption technique has been used. In this technique encryption/decryption key gets form from the KSOMSCT synchronized session key and if the length of the plain text get increased then four bits circular left shift operation get perform on the synchronized session key to generate the encryption/decryption key for the rest of the portion. So, this technique does not need to store encryption/decryption key.

In DHLPSCT and CDHLPSCT Simulated Annealing and Genetic Algorithm based encryption/decryption technique has been performed respectively. These two techniques generates fittest 128 bit key for encryption/decryption purpose and if the length of the plain text is greater than 128 then triangle edge and square edge based key expansion technique respectively is used to generate the key for the exceed portion. So, these two technique stores only 128 bit fittest keystream which is at part AES but less compare to RC4 and Vernam Cipher.

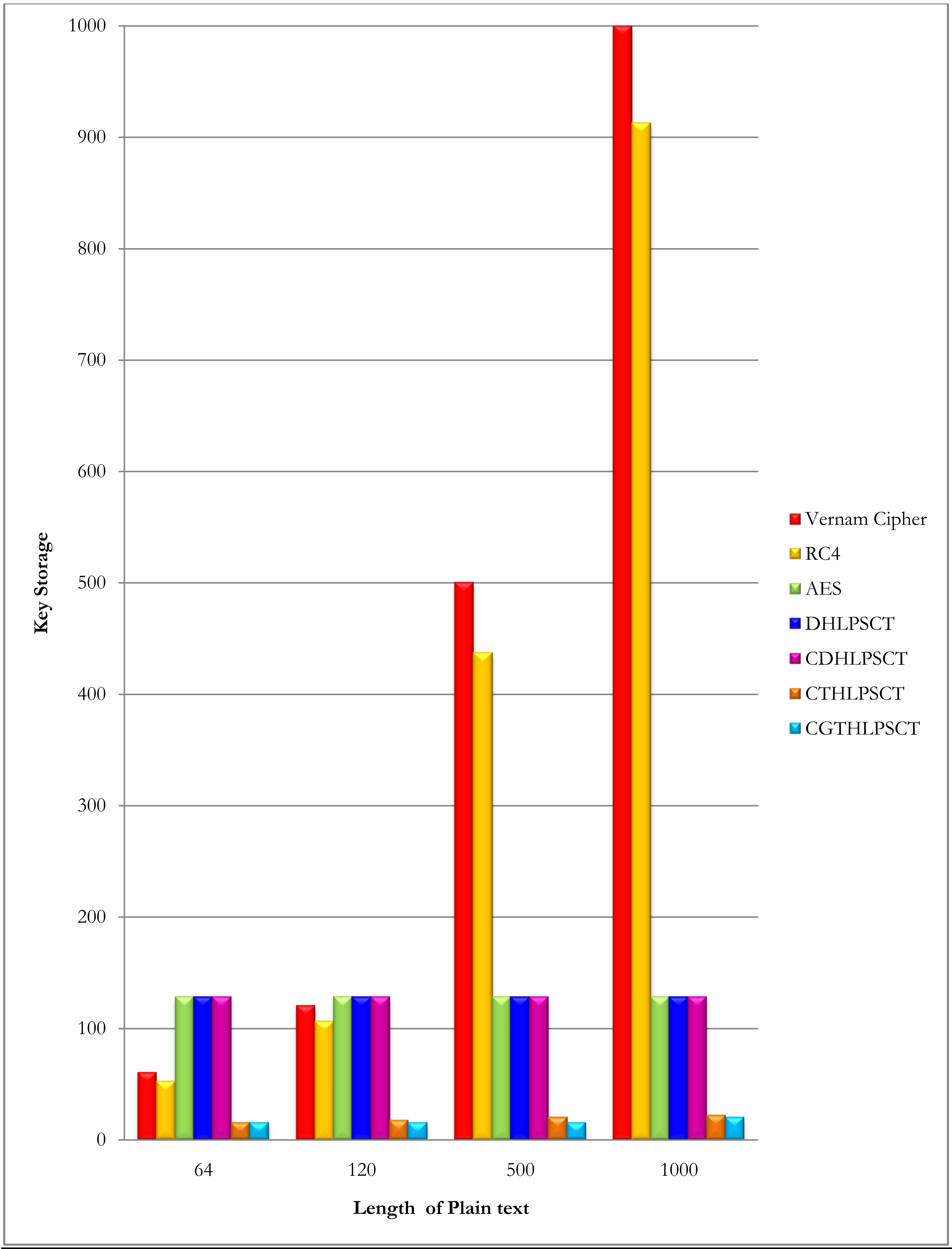


Figure 7.50: Comparisons of length of plain text vs. keystream storage between proposed and existing techniques

In CTHLPSCT, Ant Colony Intelligence (ACI) based technique is used for encryption/decryption and in this technique the number of keys to be stored is less when compared to AES, RC4, Vernam Cipher. In ACI keystream is generated based on the distribution of characters in the plain text. A Comparatively smaller number of keys has to be stored since the keys for the remaining portion of the text are generated using the keys in the keystream. In ACI only fifteen bits keystream need to be store for plain text size $64, 120, 500$ and for the plain text of length 1000 only 20 bits keystream need to be store. In ACI if number of bits in a plain text is grater than the keystream then the values of the keystream are added to a predetermined value to generate the keys for the characters in the plain text which is at a position grater than the length of the key stream. In Ant Colony Intelligence based technique, to generate the keystream for encryption/decryption based on the distribution of characters in the plain text has several shortcomings. The drawback of this method was that the pheromone deposition of the ant agent evaporates when it moves to the next trail and therefore the ant agent needs to update the pheromone deposition representing the keystream. The energy value denoting its attractiveness towards the solution is found by counting the number of characters in the keystream occurring in the plain text. Let suppose the minimum length of the keystream used by the ant agent is nine during each trail and the solution is obtained in three trails. Then the minimum number of total comparisons of the characters in the keystream with the plain text is 27, to obtain a keystream of length nine. Due to the evaporation of the pheromone deposition in each trail the length of the keystream may increase or decrease.

In the CGTHLPSCT, Particle Swarm Intelligence (PSI) based technique is used for encryption/decryption. In the PSI, though the keys used for encryption looks like a series of random numbers, the keys cannot be cracked because a random number generator is not used to generate the keys. Also the keystream generation depends on the character distribution in the plain text overcoming the drawback of Vernam Cipher. In addition to this the PSI method reduces the number of keys to be stored and distributed compared to that of AES, RC4, Vernam Cipher when the length of the plain text is large. The characters used for comparison is stored and each time a velocity is given to the particle only the new characters denoting the velocity are compared with the plain text. Consider the case where the length of the keystream is nine. Since the characters in the keystream do not change until the solution is obtained it is not necessary that the particle keystream length should be nine initially. Let suppose a particle keystream of length five is taken and a velocity whose keystream length is two given to the particle during the first move of the particle. During the second move of the particle a velocity keystream of length two is given and the solution is obtained. Then the minimum number of total comparisons of the characters in the keystream with the plain text is nine to obtain a keystream of length nine. Each time a velocity is given to the particle the characters in the keystream are unique. This would ensure that unlike ACI method the same characters are not compared with the plain text for their occurrence. So, PSI based encryption/decryption is better than ACI based technique.

In RC4 the number of keys to be stored is less when compared to Vernam Cipher. This stream cipher method is vulnerable to analytic attacks. 1 out of every 256 keys is a weak key. These keys can be identified by cryptanalysis which can find whether the generated bytes are strongly correlated with the bytes of the key.

In Vernam Cipher the keys are randomly generated using random stream generator. The drawback is that the number of keys to be stored and distributed should be equal to the length of the plain text. Also the keys used to encrypt the plain text can be found if the random number generator is cracked.

In the PSI and ACI, though the keys used for encryption looks like a series of random numbers, the keys cannot be cracked because a random number generator is not used to generate the keys. Also the keystream generation depends on the character distribution in the plain text overcoming the drawback of Vernam Cipher. In addition to this the PSI and ACI method reduces the number of keys to be stored and distributed compared to that of Vernam Cipher when the length of the plain text is large.

7.4 Encryption/Decryption Time

All test programs for the algorithms were equipped to display the total encryption and decryption time. Time taken is the difference between processor clock ticks in starting and end. Times are computed in milliseconds (ms). The lower the time taken, the better is for a typical end user. Since the CPU clock ticks taken as time, there might be a slight variation in actual time. This variation is insignificant and may be ignored.

Section 7.4.1 shows the result on *.dll* files, section 7.4.2 shows the result on *.exe* files, section 7.4.3 shows the result on *.txt* files, section 7.4.4 shows the result on *.doc* files.

7.4.1 *.dll* files

Twenty *.dll* files of different sizes varying from 3,216 bytes to 5,456,704 bytes have been taken to generate the data containing various attributes for evaluation of the proposed technique. Table 7.62 shows the encryption times (Enc.) and decryption times (Dec.) of *.dll* type files obtained using proposed and existing TDES, AES. Enc. varies from 16 m.sec. to 345 m.sec. for CGTHLPSCT, from 15 m.sec. to 479 m.sec. for CTHLPSCT, from 16 m.sec. to 429 m.sec. for CDHLPSCT, from 15 m.sec. to 430 m.sec. for DHLPSCT, from 15 m.sec. to 403 m.sec. for KSOMSCT, from 12 m.sec. to 154 m.sec. for AES, from 14 m.sec. to 1180 m.sec. for TDES. Dec. varies from 15 m.sec. to 350 m.sec. for CGTHLPSCT, from 15 m.sec. to 538 m.sec. for CTHLPSCT, from 15 m.sec. to 460 m.sec. for CDHLPSCT, from 15 m.sec. to 427 m.sec. for DHLPSCT, from 15 m.sec. to 670 m.sec. for KSOMSCT, from 11 m.sec. to 269 m.sec. for AES, from 15 m.sec. to 1168 m.sec. for TDES.

Figure 7.51 and 7.52 shows the graphical representation of the relationship between the encryption times against the *.dll* type source files and the decryption times against the *.dll* type source files respectively for proposed, AES and TDES techniques. Enc. and Dec. for proposed and AES are near equal but much lower than that of TDES. In both the figures, the gradients of the curves for TDES are higher for larger source files.

Table: 7.62
Comparisons of encryption and decryption times for *.dll* files

Sl. no.	**Source file name**	**Source file size (in bytes)**	**CGTHLPSCT (in m.sec.)**		**CTHLPSCT (in m.sec.)**		**CDHLPSCT (in m.sec.)**		**DHLPSCT (in m.sec.)**		**KSOMSCT (in m.sec.)**		**AES (in m.sec.)**		**TDES (in m.sec.)**	
			Enc.	**Dec.**	**Enc.**	**Dec.**	**Enc.**	**Dec.**	**Enc.**	**Dec.**	**Enc.**	**Dec.**	**Enc.**	**Dec.**	**Enc.**	**Dec.**
1	a01.dll	3,216	16	15	15	15	16	16	15	30	15	15	12	11	14	15
2	a02.dll	6,656	32	16	15	30	32	15	16	15	30	33	16	16	16	15
3	a03.dll	12,288	17	15	16	15	30	33	38	38	17	17	19	17	15	15
4	a04.dll	24,576	30	16	33	33	16	49	33	16	43	34	13	36	39	13
5	a05.dll	58,784	16	15	30	40	43	33	49	46	39	17	12	30	34	12
6	a06.dll	85,020	33	16	30	15	48	48	62	40	32	38	16	31	33	14
7	a07.dll	169,472	17	32	31	30	92	72	69	45	37	30	15	13	49	38
8	a08.dll	359,936	33	30	79	62	68	70	61	64	42	65	18	34	71	76
9	a09.dll	593,920	50	53	104	130	79	99	78	101	107	129	33	48	218	232
10	a10.dll	909,312	74	62	120	118	152	122	119	112	144	112	30	66	189	184
11	a11.dll	1,293,824	81	97	177	158	156	147	173	173	182	157	74	120	260	459
12	a12.dll	1,925,185	120	142	219	261	107	181	154	239	216	253	42	95	393	501
13	a13.dll	2,498,560	178	164	203	292	216	219	152	236	201	294	79	129	532	518
14	a14.dll	3,485,968	218	230	382	327	171	233	239	280	398	329	102	174	812	752
15	a15.dll	3,790,336	240	251	360	420	366	262	430	322	349	428	107	178	897	923
16	a16.dll	4,253,816	278	283	287	651	353	379	341	317	294	682	128	201	908	890
17	a17.dll	4,575,232	282	288	379	439	429	312	405	233	392	424	149	269	923	964
18	a18.dll	4,883,456	319	322	373	403	408	323	322	427	327	405	143	212	964	1051
19	a19.dll	5,054,464	345	337	432	386	355	257	264	401	403	287	154	213	1180	1162
20	a20.dll	5,456,704	337	350	479	538	358	460	280	343	376	670	152	256	1172	1168

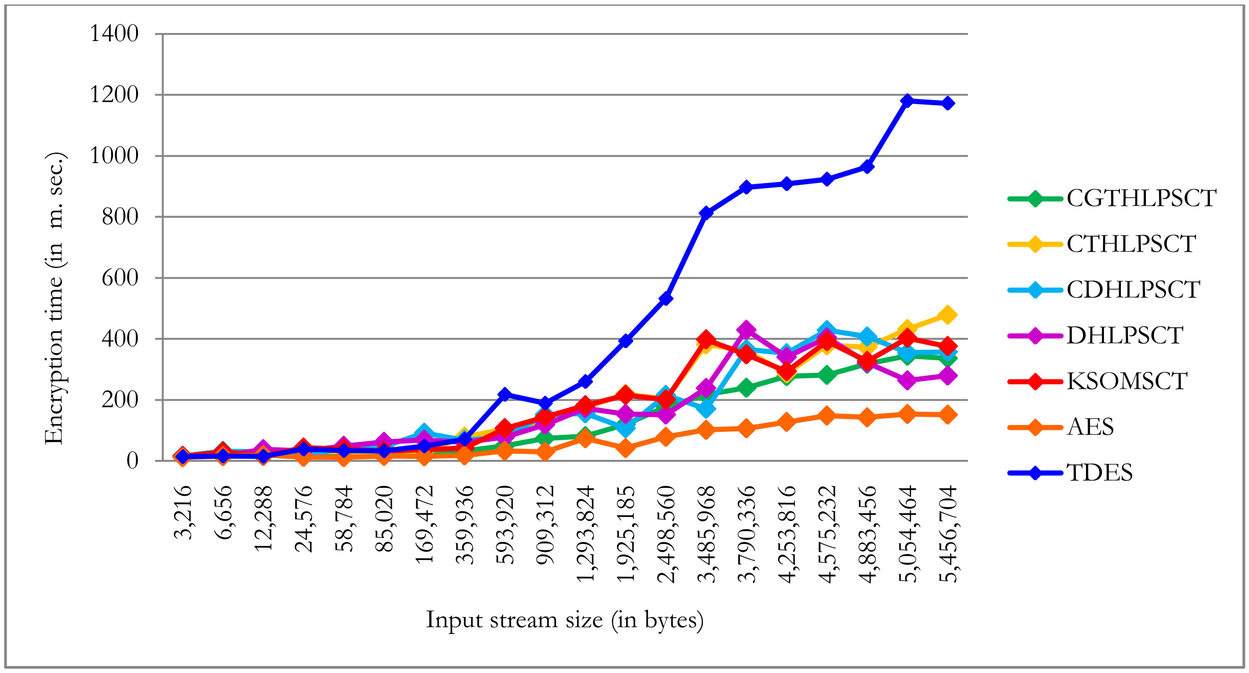


Figure 7.51: Graphical representation of encryption time against the varying size of input stream of *.dll* files

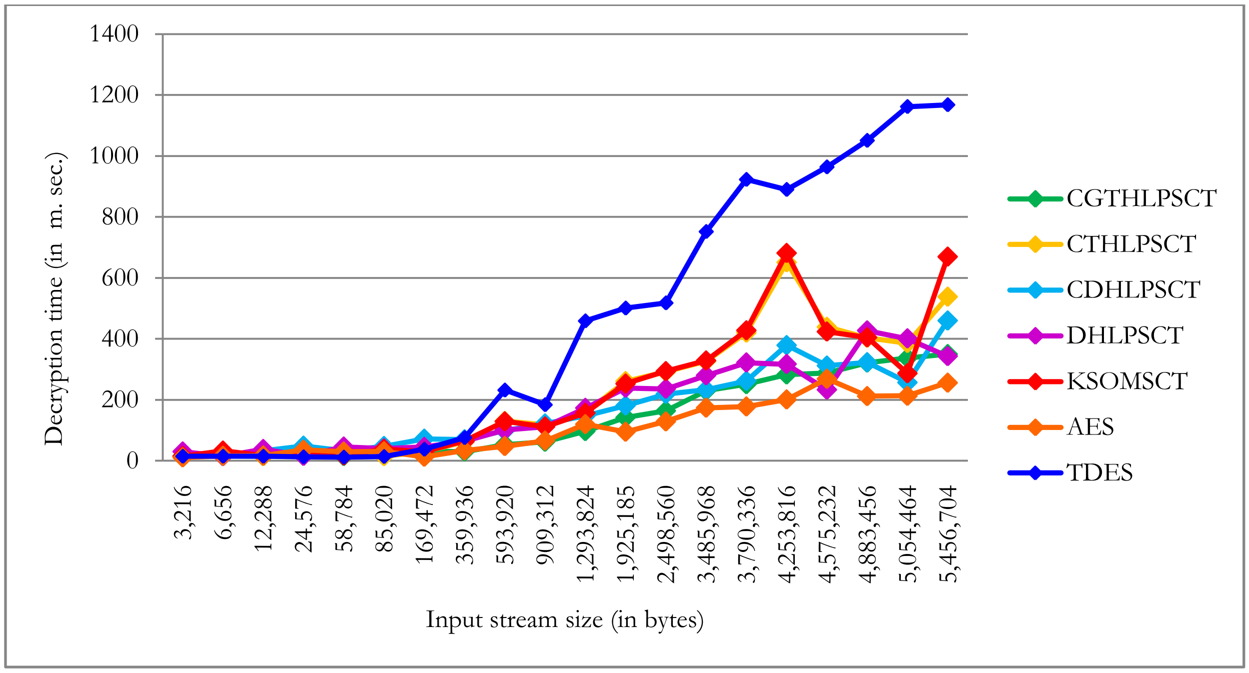


Figure 7.52: Graphical representation of decryption time against the varying size of input stream of *.dll* files

7.4.2 .*exe* files

Twenty .*exe* files of different sizes varying from 1,063 bytes to 6,735,934 bytes have been taken to generate the data containing various attributes for evaluation of the proposed technique. Table 7.63 shows the encryption times (Enc.) and decryption times (Dec.) of .*exe* type files obtained using proposed and existing TDES, AES. Enc. varies from 16 m.sec. to 458 m.sec. for CGTHLPSCT, from 16 m.sec. to 589 m.sec. for CTHLPSCT, from 15 m.sec. to 562 m.sec. for CDHLPSCT, from 15 m.sec. to 402 m.sec. for DHLPSCT, from 15 m.sec. to 674 m.sec. for KSOMSCT, from 12 m.sec. to 374 m.sec. for AES, from 12 m.sec. to 1379 m.sec. for TDES. Dec. varies from 15 m.sec. to 438 m.sec. for CGTHLPSCT, from 15 m.sec. to 395 m.sec. for CTHLPSCT, from 12 m.sec. to 646 m.sec. for CDHLPSCT, from 15 m.sec. to 433 m.sec. for DHLPSCT, from 15 m.sec. to 377 m.sec. for KSOMSCT, from 15 m.sec. to 399 m.sec. for AES, from 15 m.sec. to 1767 m.sec. for TDES.

Figure 7.53 and 7.54 shows the graphical representation of the relationship between the encryption times against the .*exe* type source files and the decryption times against the .*exe* type source files respectively for proposed, AES and TDES techniques. Enc. and Dec. for proposed and AES are near equal but much lower than that of TDES. In both the figures, the gradients of the curves for TDES are higher for larger source files.

Table: 7.63

Comparisons of encryption and decryption times for *.exe* files

Sl. no.	Source file name	Source file size (in bytes)	CGTHLPSCT (in m.sec.)		CTHLPSCT (in m.sec.)		CDHLPSCT (in m.sec.)		DHLPSCT (in m.sec.)		KSOMSCT (in m.sec.)		AES (in m.sec.)		TDES (in m.sec.)	
			Enc.	**Dec.**	**Enc.**	**Dec.**	**Enc.**	**Dec.**	**Enc.**	**Dec.**	**Enc.**	**Dec.**	**Enc.**	**Dec.**	**Enc.**	**Dec.**
1	a01. exe	1,063	16	15	16	15	15	17	16	15	17	15	12	16	15	16
2	a02. exe	2,518	16	15	16	15	15	12	39	35	19	15	16	16	15	15
3	a03. exe	8,250	30	15	15	30	16	17	15	16	16	34	46	15	15	15
4	a04. exe	15,937	15	16	30	15	15	16	16	17	15	17	48	15	16	15
5	a05. exe	22,874	17	15	44	33	17	12	16	18	30	31	49	15	17	15
6	a06. exe	35,106	16	39	39	49	39	37	38	37	37	48	49	16	17	15
7	a07. exe	52,032	36	16	33	31	43	41	42	43	34	36	53	15	39	15
8	a08. exe	145,387	33	33	67	48	42	30	31	69	32	43	62	44	42	32
9	a09. exe	248,273	39	36	72	76	97	99	75	91	71	75	63	31	98	94
10	a10. exe	478,321	42	31	79	60	111	92	75	73	66	61	16	46	94	100
11	a11. exe	738,275	50	68	103	122	120	124	123	115	108	124	79	102	171	189
12	a12. exe	1,594,276	113	95	104	144	175	149	177	122	143	100	95	74	296	327
13	a13. exe	2,273,670	177	172	158	207	213	208	283	217	107	206	147	100	504	561
14	a14. exe	2,985,306	239	180	251	261	257	177	238	327	259	295	238	259	642	738
15	a15. exe	3,412,639	232	239	269	232	358	342	369	233	266	232	147	151	678	714
16	a16. exe	3,872,984	251	288	254	263	363	280	267	179	233	269	159	178	766	1157
17	a17. exe	4,038,387	299	314	329	299	315	372	402	314	322	317	176	182	1064	1003
18	a18. exe	5,284,796	340	322	376	298	291	575	398	317	397	298	268	313	1177	1404
19	a19. exe	5,628,037	393	383	492	326	505	371	316	406	489	298	296	296	1147	1329
20	a20. exe	6,735,934	458	438	589	395	562	646	379	433	674	377	374	399	1379	1767

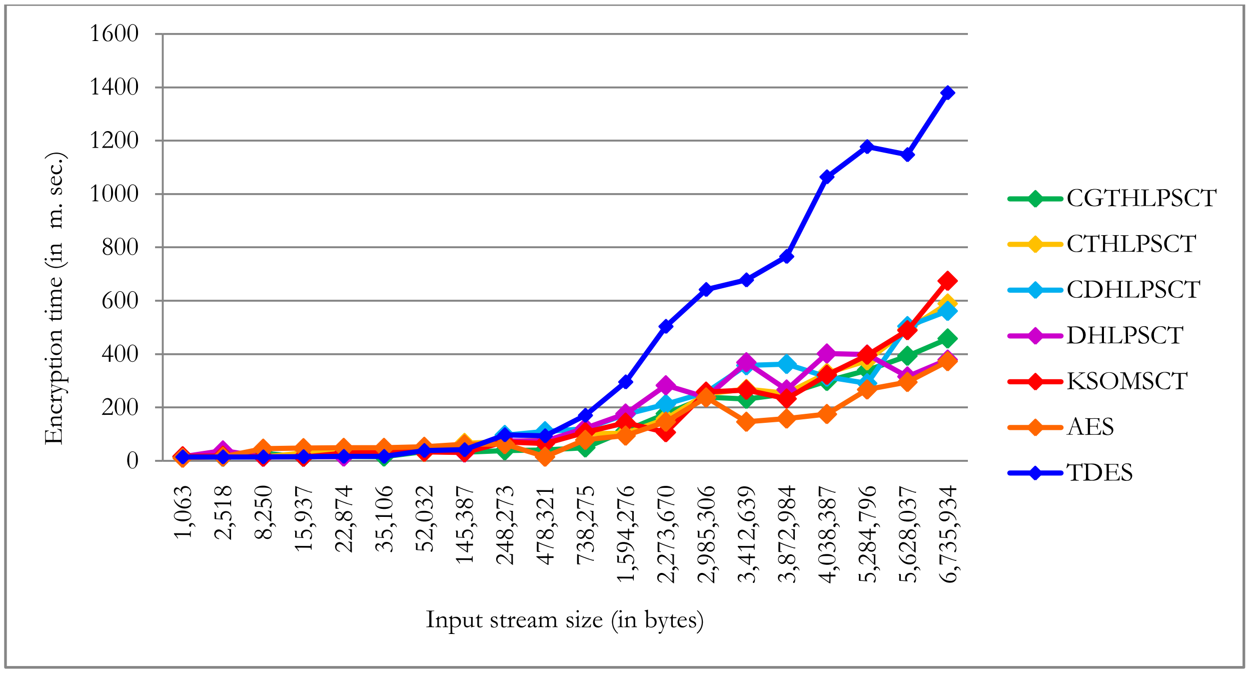


Figure 7.53: Graphical representation of encryption time against the varying size of input stream of *.exe* files

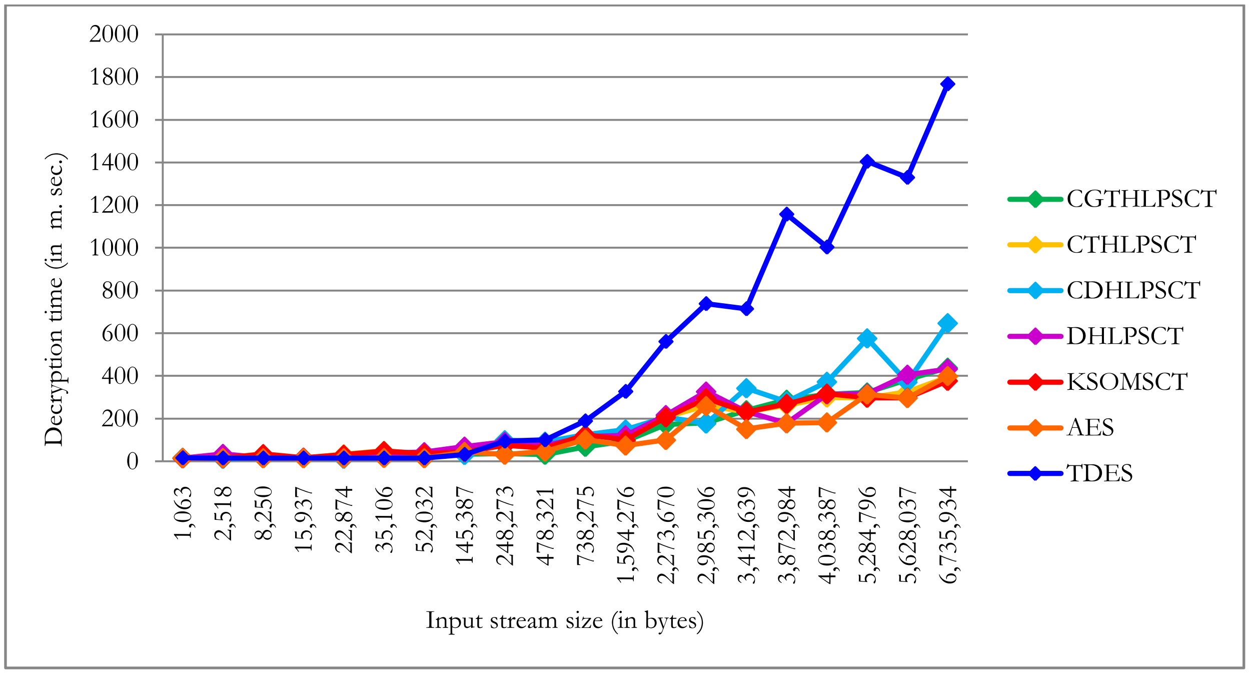


Figure 7.54: Graphical representation of decryption time against the varying size of input stream of *.exe* files

7.4.3 *.txt* files

Twenty *.txt* files of different sizes varying from 1,504 bytes to 6,702,831 bytes have been taken to generate the data containing various attributes for evaluation of the proposed technique. Table 7.64 shows the encryption times (Enc.) and decryption times (Dec.) of *.txt* type files obtained using proposed and existing TDES, AES. Enc. varies from 32 m.sec. to 421 m.sec. for CGTHLPSCT, from 16 m.sec. to 539 m.sec. for CTHLPSCT, from 31 m.sec. to 603 m.sec. for CDHLPSCT, from 15 m.sec. to 400 m.sec. for DHLPSCT, from 16 m.sec. to 542 m.sec. for KSOMSCT, from 16 m.sec. to 215 m.sec. for AES, from 16 m.sec. to 1428 m.sec. for TDES. Dec. varies from 15 m.sec. to 421 m.sec. for CGTHLPSCT, from 16 m.sec. to 629 m.sec. for CTHLPSCT, from 16 m.sec. to 530 m.sec. for CDHLPSCT, from 15 m.sec. to 508 m.sec. for DHLPSCT, from 16 m.sec. to 685 m.sec. for KSOMSCT, from 15 m.sec. to 356 m.sec. for AES, from 15 m.sec. to 1817 m.sec. for TDES.

Figure 7.55 and 7.56 shows the graphical representation of the relationship between the encryption times against the *.txt* type source files and the decryption times against the *.txt* type source files respectively for proposed, AES and TDES techniques. Enc. and Dec. for proposed and AES are near equal but much lower than that of TDES. In both the figures, the gradients of the curves for TDES are higher for larger source files.

Table: 7.64
Comparisons of encryption and decryption times for *.txt* files

Sl. no.	Source file name	Source file size (in bytes)	CGTHLPSCT (in m.sec.)		CTHLPSCT (in m.sec.)		CDHLPSCT (in m.sec.)		DHLPSCT (in m.sec.)		KSOMSCTCT (in m.sec.)		AES (in m.sec.)		TDES (in m.sec.)	
			Enc.	**Dec.**	**Enc.**	**Dec.**	**Enc.**	**Dec.**	**Enc.**	**Dec.**	**Enc.**	**Dec.**	**Enc.**	**Dec.**	**Enc.**	**Dec.**
1	a01.txt	1,504	32	16	16	16	32	16	15	15	16	33	16	15	16	16
2	a02.txt	7,921	33	17	38	16	33	32	17	32	32	48	67	15	16	16
3	a03.txt	17,036	33	30	17	30	32	30	16	33	16	16	32	15	17	15
4	a04.txt	44.624	48	15	35	31	30	49	32	32	32	33	44	16	18	15
5	a05.txt	68,823	30	30	37	32	31	42	32	46	33	17	34	16	16	116
6	a06.txt	161,935	37	32	31	33	33	39	67	65	33	33	39	39	34	38
7	a07.txt	328,017	44	33	44	46	79	68	68	68	46	44	40	38	127	145
8	a08.txt	587,290	67	77	60	99	93	73	102	99	78	108	63	47	257	178
9	a09.txt	1,049,763	79	77	141	147	176	106	178	176	146	147	75	64	183	217
10	a10.txt	1,418,025	96	71	179	158	253	211	149	204	154	156	77	70	285	326
11	a11.txt	1,681,329	113	129	187	179	178	179	205	207	183	175	94	98	347	363
12	a12.txt	2,059,318	141	157	202	201	269	267	146	212	203	204	105	129	438	450
13	a13.txt	2,618,492	217	205	203	262	210	215	268	209	205	263	184	235	566	647
14	a14.txt	3,154,937	213	232	284	399	313	422	219	378	287	390	143	158	713	766
15	a15.txt	4,073,829	262	259	350	352	235	259	204	287	428	359	297	311	953	1102
16	a16.txt	4,936,521	310	292	391	438	311	408	172	290	399	469	178	204	1002	959
17	a17.txt	5,125,847	362	343	266	463	427	347	400	544	260	438	264	215	1007	1128
18	a18.txt	5,593,219	408	424	456	376	408	432	313	372	433	377	202	255	1358	1156
19	a19.txt	5,898,302	363	377	475	453	603	435	396	404	372	456	214	327	1210	1262
20	a20.txt	6,702,831	421	439	539	629	365	530	395	508	542	685	215	356	1428	1817

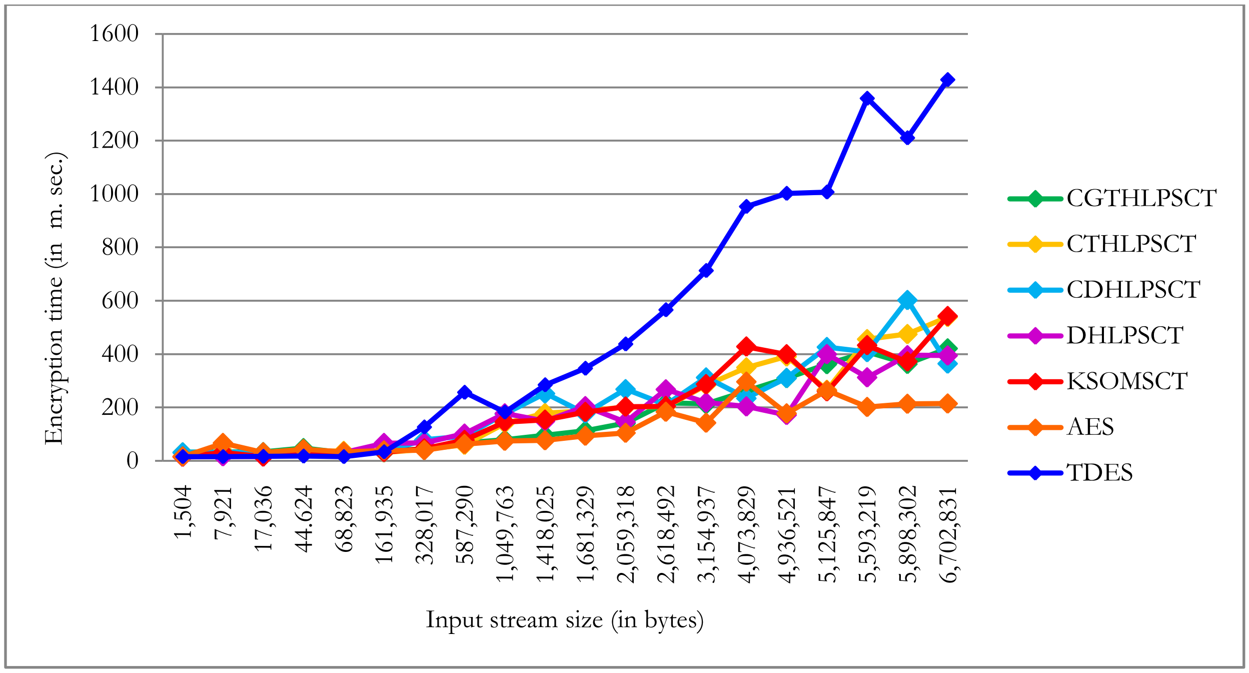


Figure 7.55: Graphical representation of encryption time against the varying size of input stream of *.txt* files

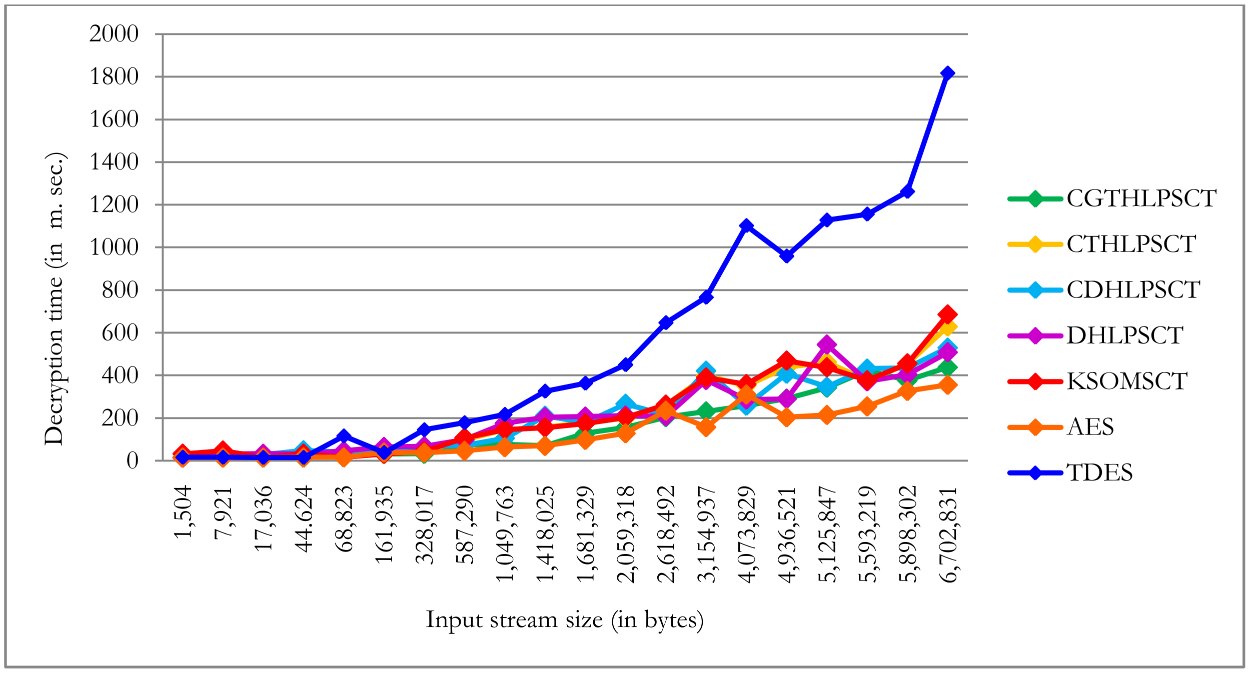


Figure 7.56: Graphical representation of decryption time against the varying size of input stream of *.txt* files

7.4.4 *.doc* files

Twenty *.doc* files of different sizes varying from 21,052 bytes to 5,472,298 bytes have been taken to generate the data containing various attributes for evaluation of the proposed technique. Table 7.65 shows the encryption times (Enc.) and decryption times (Dec.) of *.doc* type files obtained using proposed and existing TDES, AES. Enc. varies from 15 m.sec. to 331 m.sec. for CGTHLPSCT, from 17 m.sec. to 432 m.sec. for CTHLPSCT, from 32 m.sec. to 519 m.sec. for CDHLPSCT, from 17 m.sec. to 430 m.sec. for DHLPSCT, from 13 m.sec. to 549 m.sec. for KSOMSCT, from 12 m.sec. to 265 m.sec. for AES, from 15 m.sec. to 1043 m.sec. for TDES. Dec. varies from 15 m.sec. to 327 m.sec. for CGTHLPSCT, from 32 m.sec. to 466 m.sec. for CTHLPSCT, from 32 m.sec. to 536 m.sec. for CDHLPSCT, from 17 m.sec. to 483 m.sec. for DHLPSCT, from 11 m.sec. to 429 m.sec. for KSOMSCT, from 15 m.sec. to 218 m.sec. for AES, from 15 m.sec. to 1378 m.sec. for TDES.

Figure 7.57 and 7.58 shows the graphical representation of the relationship between the encryption times against the *.doc* type source files and the decryption times against the *.doc* type source files respectively for proposed, AES and TDES techniques. Enc. and Dec. for proposed and AES are near equal but much lower than that of TDES. In both the figures, the gradients of the curves for TDES are higher for larger source files.

Table: 7.65
Comparisons of encryption and decryption times for *.doc* files

Sl. no.	**Source file name**	**Source file size (in bytes)**	**CGTHLPSCT (in m.sec.)**		**CTHLPSCT (in m.sec.)**		**CDHLPSCT (in m.sec.)**		**DHLPSCT (in m.sec.)**		**KSOMSCT (in m.sec.)**		**AES (in m.sec.)**		**TDES (in m.sec.)**	
			Enc.	**Dec.**	**Enc.**	**Dec.**	**Enc.**	**Dec.**	**Enc.**	**Dec.**	**Enc.**	**Dec.**	**Enc.**	**Dec.**	**Enc.**	**Dec.**
1	a01. doc	21,052	17	16	17	32	32	32	17	17	33	35	12	33	15	15
2	a02. doc	33,897	15	16	18	33	48	37	17	32	16	11	15	15	16	15
3	a03. doc	45,738	15	15	34	49	69	68	48	43	37	33	15	16	16	16
4	a04. doc	75,093	16	15	39	33	43	41	44	35	13	32	16	15	16	17
5	a05. doc	106,872	16	16	43	36	67	69	33	38	47	45	16	15	37	36
6	a06. doc	327.054	16	33	78	73	68	62	66	74	98	76	15	35	108	72
7	a07. doc	582,831	34	16	70	60	73	79	65	60	64	44	16	44	125	113
8	a08. doc	729,916	37	34	104	129	96	98	93	129	99	99	37	43	141	159
9	a09. doc	1,170,251	78	32	126	158	108	117	128	158	113	126	34	68	235	328
10	a10. doc	1,749,272	96	113	197	204	204	15	179	202	216	231	63	77	566	327
11	a11. doc	2,045,805	139	130	268	231	293	237	261	234	217	212	69	95	407	423
12	a12. doc	2,372,014	145	149	224	284	217	252	210	281	209	266	127	187	678	485
13	a13. doc	2,869,275	187	228	258	216	239	183	213	200	213	344	75	129	675	562
14	a14. doc	3,161,353	199	202	262	299	296	178	267	268	327	328	73	124	812	650
15	a15. doc	3,570,295	213	213	327	298	343	373	313	296	404	323	174	148	761	864
16	a16. doc	3,834,427	244	256	284	466	372	428	258	483	429	310	102	174	756	925
17	a17. doc	4,011,986	266	248	342	353	318	287	359	359	268	365	129	179	1004	847
18	a18. doc	4,562,385	307	297	358	419	519	513	353	438	549	429	120	207	899	930
19	a19. doc	4,839,102	323	327	389	377	453	536	359	375	353	323	142	204	1037	1378
20	a20.doc	5,472,298	331	315	432	446	483	489	430	424	329	345	265	218	1043	1137

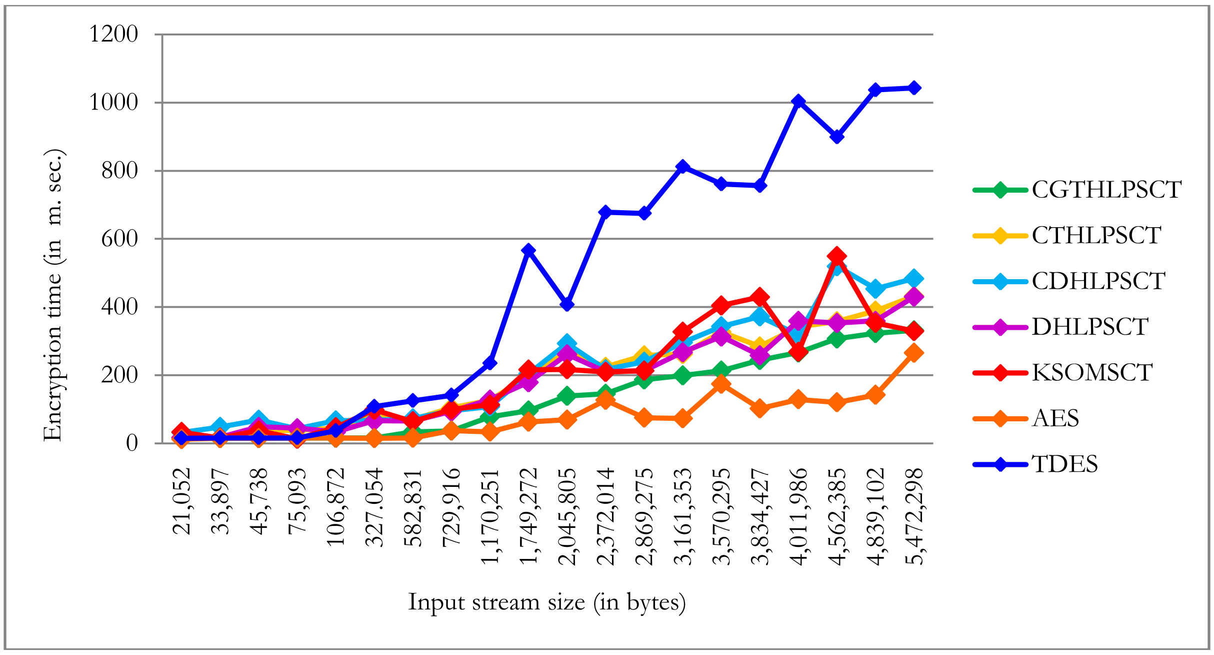


Figure 7.57: Graphical representation of encryption time against the varying size of input stream of *.doc* files

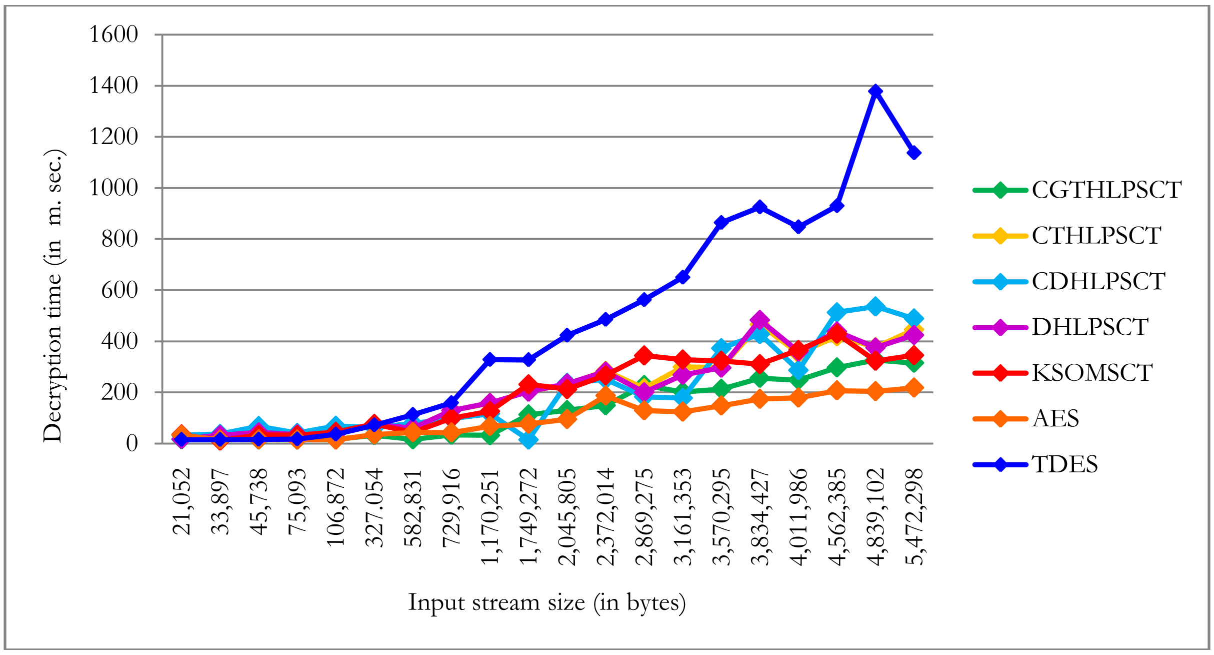


Figure 7.58: Graphical representation of decryption time against the varying size of input stream of *.doc* files

7.5 Avalanche, strict Avalanche and Bit Independence

Comparison between the source and encrypted byte has been made and changed of bits in encrypted bytes has been observed for a single bit change in the original message byte for the entire or a relative large number of bytes. The standard deviation from the expected values calculated. Subtract the ratio of the calculated standard deviation with expected value from 1.0 to get the avalanche and Strict Avalanche on a $0.0 - 1.0$ scale.

A function $f : \{0,1\}^n \rightarrow \{0,1\}^n$ satisfies the Bit Independence criteria if $\forall\, i, j, k \in \{1,2,\dots,n\}$, with $j \neq k$, inverting input bit i cause output bits j and k to change independently. To measure the Bit Independence concept, the correlation coefficient between the j^{th} and k^{th} components of the output difference string is needed, which is called the Avalanche vector A^{e_i}.

The higher and closer value to 1.0, the better Avalanche and Strict Avalanche is said to be satisfied. In case of files contacting only text messages in plain format, there are no bytes in the range of byte 128 to byte 255. That is the reason for which the values of Bit Independence test for text files are very low. Section 7.5.1 deals with *.dll* files and section 7.5.2 deals with *.exe* files and results of *.txt* and *.doc* files are shown in section 7.5.3 and 7.5.4 respectively.

7.5.1 *.dll* files

Twenty *.dll* files of different sizes varying from 3216 bytes to $5{,}456{,}704$ bytes have been taken for Avalanche, Strict Avalanche and Bit Independence test. Table 7.67, 7.68, 7.69 and 7.70 shows the Avalanche, Strict Avalanche and Bit Independence test of *.dll* type files obtained using proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing RSA, TDES, AES respectively. Avalanches, Strict Avalanche test value for each file are very close to the value 1 for all four techniques. Bit Independence values vary from 0.4041078 to 0.8273190 for KSOMSCT, from 0.4329089 to 0.8439064 for DHLPSCT, from 0.4890656 to 0.8328754 for CDHLPSCT, from 0.4739069 to 0.8729859 for CTHLPSCT, from 0.4389025 to 0.8673657 for CGTHLPSCT, from 0.4029032 to 0.8310369 for RSA, from 0.3859391 to 0.8187284 for TDES and from 0.4143399 to 0.8255937 for AES respectively.

Figure 7.59 and 7.60 show the graphical representation of the comparison of results of Avalanche, Strict Avalanche (average values) and Bit Independence test results (average values) respectively of the *.dll* type source files for using proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing RSA, TDES and AES techniques. Average Avalanche values of proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing RSA, TDES, AES are 0.9980800, 0.9926640, 0.9884830, 0.97320534, 0.97189467, 0.9999469, 0.9999142, and 0.9998914 respectively. Average Strict Avalanche values of proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing RSA, TDES, AES are 0.9967630, 0.9911350, 0.9865620, 0.9705220, 0.9687704, 0.9996540, 0.9996324, 0.9996890 respectively. Average Bit Independence values of proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing RSA, TDES, AES are 0.7262960, 0.7330390, 0.7419000, 0.7556560, 0.7569857, 0.7211989, 0.7147735, and 0.7190952 respectively. Proposed CGTHLPSCT has the highest average Bit Independence value which indicates that this technique provides better degree of security.

Table: 7.66
Comparisons of Avalanche of *.dll* files

Serial no.	Source file name	Source file size (in bytes)	CGTHLPSCT	CTHLPSCT	CDHLPSCT	DHLPSCT	KSOMSCT	RSA	TDES	AES
1	a01.dll	3216	0.9240385	0.9789072	0.9883765	0.993643	0.9942894	0.9999801	0.9991916	0.9997694
2	a02.dll	6,656	0.9792846	0.9747418	0.9841962	0.992867	0.9997120	0.9998242	0.9999723	0.9993602
3	a03.dll	12,288	0.9701239	0.9682101	0.9857208	0.9942318	0.9967306	0.9999427	0.9996628	0.9994563
4	a04.dll	24,576	0.9773894	0.9698297	0.9862714	0.9963427	0.9982543	0.9998167	0.9998776	0.9998459
5	a05.dll	58,784	0.9781406	0.9500342	0.9963092	0.9972906	0.9985238	0.9997956	0.9999581	0.9997965
6	a06.dll	85,020	0.9771053	0.9713896	0.988131	0.9903277	0.9993074	0.9998603	0.9999630	0.9999417
7	a07.dll	169,472	0.9682618	0.9710428	0.9962188	0.9951207	0.9974293	0.9999495	0.9999511	0.9999477
8	a08.dll	359,936	0.9663952	0.9892187	0.9956543	0.9984329	0.9998762	0.9999602	0.9999263	0.9999634
9	a09.dll	593,920	0.9681520	0.9741985	0.9872754	0.9917984	0.9969827	0.9999424	0.9999642	0.9999350
10	a10.dll	909,312	0.9702894	0.9632639	0.9966439	0.9951093	0.9971093	0.9999506	0.9999896	0.9999785
11	a11.dll	1,293,824	0.9698257	0.9733876	0.9903101	0.9953433	0.9972905	0.9999720	0.9999699	0.9999349
12	a12.dll	1,925,185	0.9682653	0.9781095	0.9894763	0.9942895	0.9969374	0.9999987	0.9999886	0.9999759
13	a13.dll	2,498,560	0.9681450	0.9823732	0.9858645	0.990342	0.9991086	0.9999988	0.9999816	0.9999875
14	a14.dll	3,485,968	0.9710536	0.9736942	0.9881193	0.9947721	0.9993217	0.9999933	0.9999749	0.9999809
15	a15.dll	3,790,336	0.9782879	0.980942	0.9812267	0.9975649	0.9989470	0.9999919	0.9999844	0.9999949
16	a16.dll	4,253,816	0.9681129	0.9758478	0.9870583	0.9954309	0.9974405	0.9999830	0.9999802	0.9999764
17	a17.dll	4,575,232	0.9885936	0.9687987	0.9811642	0.9875538	0.9984829	0.9999985	0.9999986	0.999990
18	a18.dll	4,883,456	0.9780166	0.9751095	0.988354	0.9657245	0.9967669	0.9999971	0.9999838	0.9999888
19	a19.dll	5,054,464	0.9890467	0.9755983	0.9881957	0.9951286	0.9995821	0.9999870	0.9999738	0.999998
20	a20.dll	5,456,704	0.9793654	0.9694095	0.9850876	0.9919673	0.9994990	0.9999959	0.9999918	0.9999949
Average			0.97189467	0.97320534	0.9884830	0.9926640	0.9980800	0.9999469	0.9999142	0.9998914

Table: 7.67
Comparisons of Strict Avalanche of *.dll* files

Serial no.	**Source file name**	**Source file size (in bytes)**	**CGTHLPSCT**	**CTHLPSCT**	**CDHLPSCT**	**DHLPSCT**	**KSOMSCT**	**RSA**	**TDES**	**AES**
1	a01.dll	3216	0.9226498	0.9950948	0.9867393	0.9836964	0.9938749	0.9969599	0.9971642	0.9978076
2	a02.dll	6,656	0.9774367	0.9710937	0.9817382	0.9913773	0.9984653	0.9986915	0.9990365	0.9987374
3	a03.dll	12,288	0.9693621	0.9639474	0.9847830	0.9937843	0.9953728	0.9994033	0.9992313	0.9993459
4	a04.dll	24,576	0.9724087	0.9673049	0.9847628	0.9947382	0.9972874	0.9995191	0.9994087	0.9994590
5	a05.dll	58,784	0.9713476	0.9479383	0.9937821	0.9966253	0.9963879	0.9996997	0.9995125	0.9996533
6	a06.dll	85,020	0.9707635	0.9683389	0.9873094	0.9886419	0.9983875	0.9996887	0.9996336	0.9997185
7	a07.dll	169,472	0.9593210	0.9683736	0.9958792	0.9937753	0.9962874	0.9998370	0.9997457	0.9997785
8	a08.dll	359,936	0.9653280	0.9860381	0.9937846	0.9963782	0.9983634	0.9998923	0.9997572	0.9999013
9	a09.dll	593,920	0.9639518	0.9719283	0.9846382	0.9904747	0.9953753	0.9998764	0.9998565	0.9998851
10	a10.dll	909,312	0.9693879	0.9619398	0.9950937	0.9946722	0.9957364	0.9999059	0.9999310	0.9999223
11	a11.dll	1,293,824	0,9680642	0.9702865	0.9873948	0.9947883	0.9962872	0.9999270	0.9997997	0.9999080
12	a12.dll	1,925,185	0.9634965	0.9759228	0.9884736	0.9937809	0.9959348	0.9999483	0.9999338	0.9999267
13	a13.dll	2,498,560	0.9653986	0.9790456	0.9834751	0.9893721	0.9983756	0.9999607	0.9999528	0.9999519
14	a14.dll	3,485,968	0.9692101	0.9718474	0.9859038	0.9938801	0.9984755	0.9999477	0.9999209	0.9999528
15	a15.dll	3,790,336	0.9774707	0.9609543	0.9783673	0.9968382	0.9967364	0.9999767	0.9999648	0.9999832
16	a16.dll	4,253,816	0.9597480	0.9730967	0.9850939	0.9947722	0.9967846	0.9999457	0.9999551	0.9999604
17	a17.dll	4,575,232	0.9842764	0.9650949	0.9783670	0.9867003	0.9972637	0.9999750	0.9999601	0.9999655
18	a18.dll	4,883,456	0.9784328	0.9729875	0.9863537	0.9642291	0.9951093	0.9999702	0.9999648	0.9999662
19	a19.dll	5,054,464	0.9899431	0.9732986	0.9869874	0.9947783	0.9973645	0.9999806	0.9999477	0.9999805
20	a20.dll	5,456,704	0.9774108	0.9659073	0.9823091	0.9893874	0.9973856	0.9999742	0.9999716	0.9999750
Average			0.9687704	0.9705220	0.9865620	0.9911350	0.9967630	0.9996540	0.9996324	0.9996890

Table: 7.68
Comparisons of Bit Independence of *.dll* files

Serial no.	**Source file name**	**Source file size (in bytes)**	**CGTHLPSCT**	**CTHLPSCT**	**CDHLPSCT**	**DHLPSCT**	**KSOMSCT**	**RSA**	**TDES**	**AES**
1	a01.dll	3216	0.6182748	0.5893847	0.5956399	0.5643908	0.5481262	0.5420852	0.5496153	0.5545931
2	a02.dll	6,656	0.7302857	0.7630963	0.7128967	0.7634921	0.7286318	0.7497170	0.7457876	0.7434461
3	a03.dll	12,288	0.8174396	0.8649063	0.8328754	0.7760548	0.8139045	0.8039742	0.8042259	0.8055426
4	a04.dll	24,576	0.4389025	0.4739069	0.7863429	0.4329089	0.4041078	0.4029032	0.3859391	0.4143399
5	a05.dll	58,784	0.8302439	0.8729859	0.4890656	0.8045673	0.7791335	0.7838634	0.7752800	0.7792759
6	a06.dll	85,020	0.8673657	0.8189434	0.6927452	0.8140930	0.8273190	0.8168374	0.8055285	0.8146824
7	a07.dll	169,472	0.7802569	0.7950953	0.7939874	0.7529177	0.7298631	0.7380596	0.7313304	0.7314167
8	a08.dll	359,936	0.8093753	0.8382987	0.7938756	0.7642098	0.7594504	0.7409458	0.7485585	0.7481308
9	a09.dll	593,920	0.7943137	0.7927642	0.7937654	0.7726347	0.7429820	0.7631679	0.7646517	0.7639671
10	a10.dll	909,312	0.7341124	0.795834	0.7639822	0.7399834	0.7174289	0.7165119	0.7124057	0.7161293
11	a11.dll	1,293,824	0.7273902	0.7849543	0.7739827	0.6967540	0.7117439	0.6764772	0.6214033	0.6420685
12	a12.dll	1,925,185	0.6461947	0.6904878	0.7827464	0.6129650	0.6529045	0.6325169	0.5884467	0.6058651
13	a13.dll	2,498,560	0.7951003	0.8175785	0.6948738	0.7840435	0.7639531	0.7710978	0.7676921	0.7695577
14	a14.dll	3,485,968	0.8035672	0.8309765	0.7963974	0.7519424	0.7319057	0.7306279	0.7456360	0.7484494
15	a15.dll	3,790,336	0.7883753	0.7049848	0.7837871	0.7740946	0.7732985	0.7652375	0.7628868	0.7623644
16	a16.dll	4,253,816	0.7825539	0.7863976	0.7253845	0.7640934	0.7726045	0.7557339	0.7539785	0.7546731
17	a17.dll	4,575,232	0.8142004	0.7186548	0.7736459	0.7995630	0.7904583	0.7722742	0.7713913	0.7701984
18	a18.dll	4,883,456	0.7953912	0.6864784	0.7728745	0.7732119	0.7529631	0.7503444	0.7550762	0.7498208
19	a19.dll	5,054,464	0.8571062	0.7342932	0.7527699	0.8439064	0.8164047	0.8310369	0.8187284	0.8256937
20	a20.dll	5,456,704	0.7092645	0.7530929	0.726355	0.6749438	0.7087279	0.6805664	0.6869081	0.6816883
Average			0.7569857	0.7556560	0.7419000	0.7330390	0.7262960	0.7211989	0.7147735	0.7190952

Table: 7.69
Comparisons of average values of Avalanche, Strict Avalanche and Bit Independence of *.dll* files

Techniques	Average values of		
	Avalanche	Strict Avalanche	Bit Independence
CGTHLPSCT	0.97189467	0.9687704	0.7569857
CTHLPSCT	0.97320534	0.9705220	0.7556560
CDHLPSCT	0.9884830	0.9865620	0.7419000
DHLPSCT	0.9926640	0.9911350	0.7330390
KSOMSCT	0.9980800	0.9967630	0.7262960
RSA	0.9999469	0.9996540	0.7211989
TDES	0.9999142	0.9996324	0.7147735
AES	0.9998914	0.9996890	0.7190952

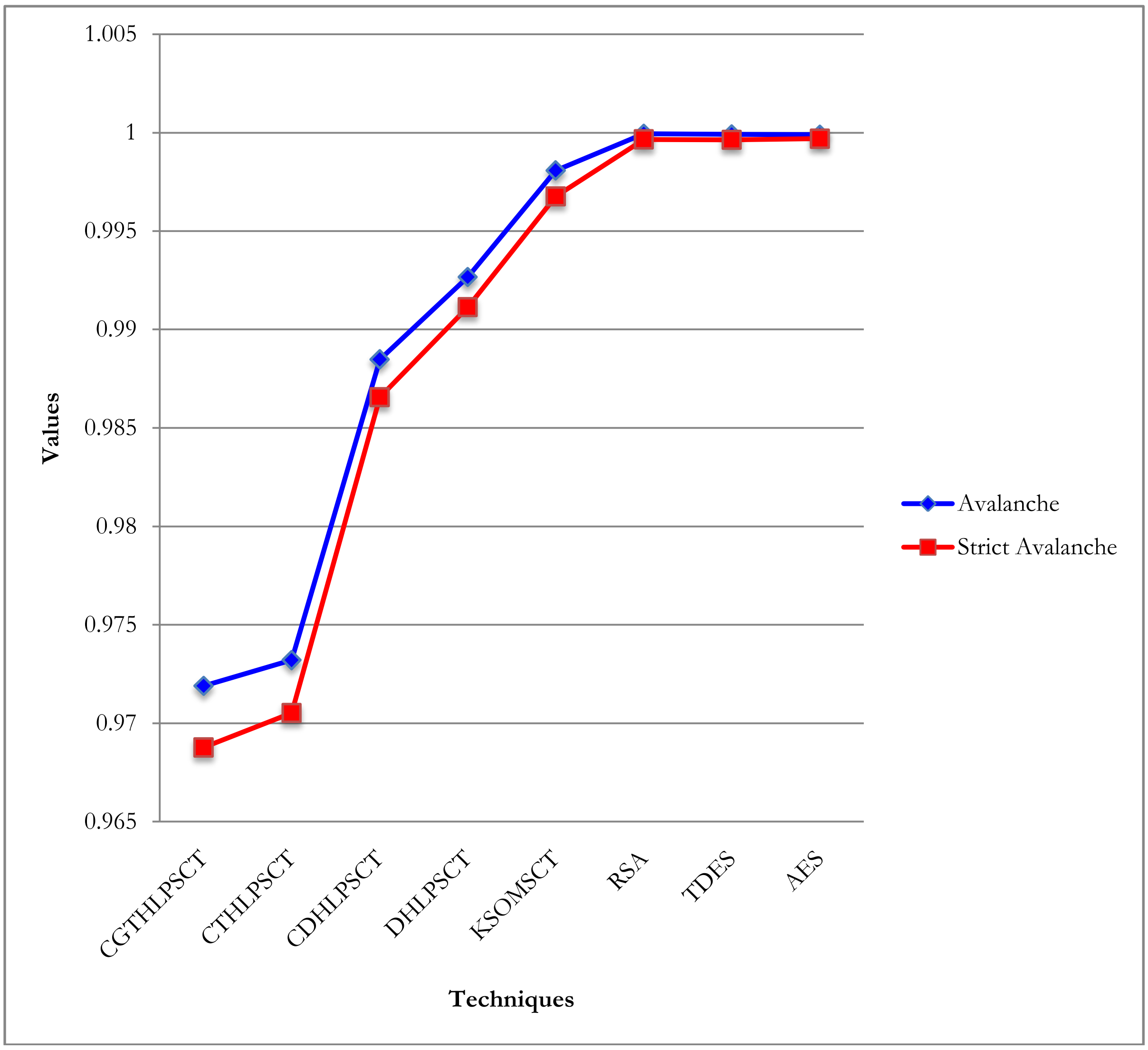


Figure 7.59: Pictorial representation of the average values of Avalanche and Strict Avalanche of *.dll* type bit stream

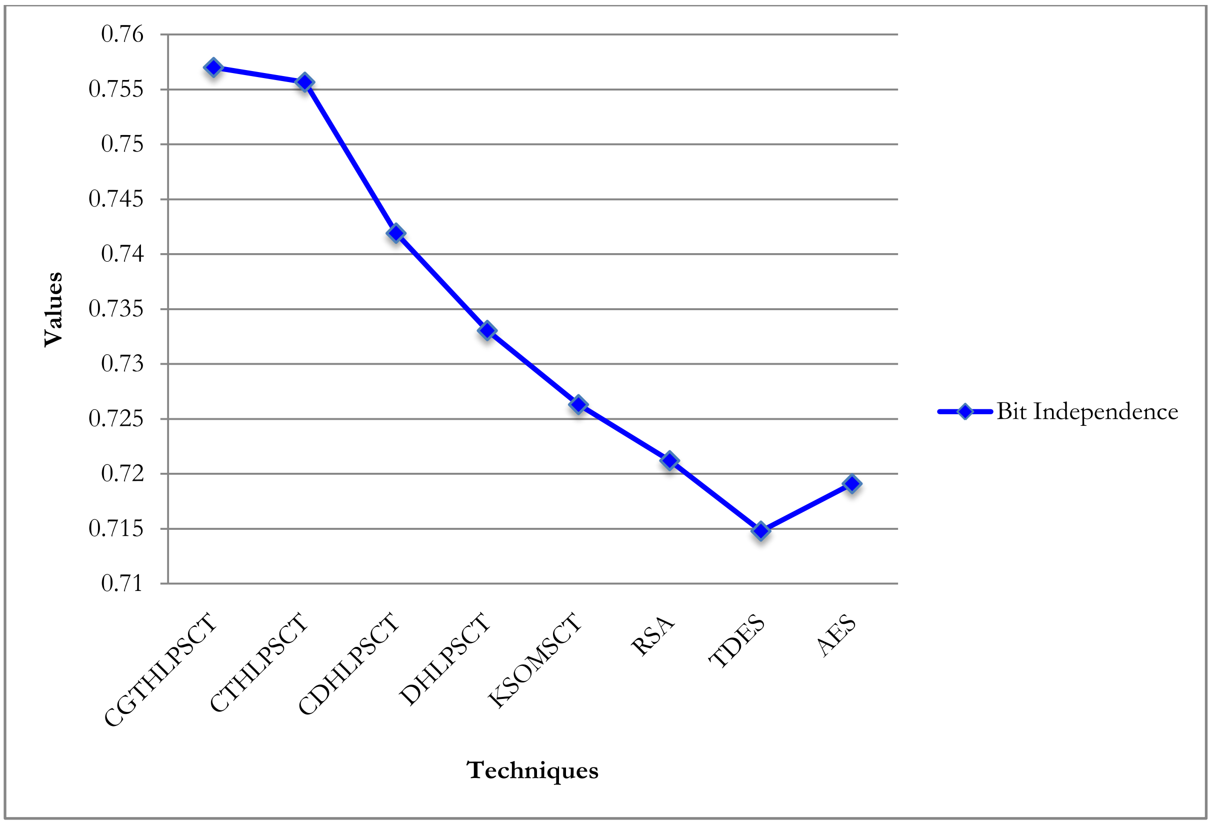


Figure 7.60: Pictorial representation of the average values of Bit Independence of *.dll* type bit stream

7.5.2 *.exe* files

Twenty *.exe* files of different sizes varying from 1,063 bytes to 6,735,934 bytes have been taken for Avalanche, Strict Avalanche and Bit Independence test. Table 7.71, 7.72, 7.73 and 7.74 shows the Avalanche, Strict Avalanche and Bit Independence test of *.exe* type files obtained using proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing RSA, TDES, AES respectively. Avalanches, Strict Avalanche test value for each file are very close to the value 1 for all four techniques. Bit Independence values vary from 0.0639870 to 0.9942170 for KSOMSCT, from 0.0464390 to 0.9956390 for DHLPSCT, from 0.0729941 to 0.9328874 for CDHLPSCT, from 0.5189645 to 0.9874045 for CTHLPSCT, from 0.3442897 to 0.9999287 for CGTHLPSCT, from 0.0214885 to 0.9846006 for RSA, from 0.2405326 to 0.9844929 for TDES and from 0.1741851 to 0.9845419 for AES respectively.

Figure 7.61 and 7.62 show the graphical representation of the comparison of results of Avalanche, Strict Avalanche (average values) and Bit Independence test results (average values) respectively of the *.exe* type source files for using proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing RSA, TDES and AES techniques. Average Avalanche values of proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing RSA, TDES, AES are 0.9988140, 0.9850070, 0.9773440, 0.9659739, 0.9651026, 0.9997574, 0.9992658, and 0.9996030 respectively. Average Strict Avalanche values of proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing RSA, TDES, AES are 0.9978890, 0.9839850, 0.9747220, 0.9616580, 0.9623935, 0.9992551, 0.9983186 and 0.9987340 respectively. Average Bit Independence values of proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing RSA, TDES, AES are 0.7057480, 0.7098500, 0.7248790, 0.7568470, 0.7709169, 0.7330390, 0.7042388, and 0.7002145 respectively. Proposed CGTHLPSCT has the highest average Bit Independence value which indicates that this technique provides better degree of security.

Table: 7.70
Comparisons of Avalanche of *.exe* files

Serial no.	**Source file name**	**Source file size (in bytes)**	**CGTHLPSCT**	**CTHLPSCT**	**CDHLPSCT**	**DHLPSCT**	**KSOMSCT**	**RSA**	**TDES**	**AES**
1	a01. exe	1,063	0.9130874	0.9572651	0.9672908	0.9752098	0.9993874	0.9975465	0.9903116	0.9955032
2	a02. exe	2,518	0.9129041	0.9673892	0.9783764	0.9842902	0.9992957	0.9987864	0.9967181	0.9988754
3	a03. exe	8,250	0.8762783	0.9521894	0.9643987	0.9714765	0.9964292	0.9995752	0.9990860	0.9986514
4	a04. exe	15,937	0.9518346	0.9598376	0.9716453	0.9804587	0.9994278	0.9996331	0.9998266	0.9995827
5	a05. exe	22,874	0.9851873	0.9562218	0.9683764	0.9763544	0.9992864	0.9999308	0.9998392	0.9998455
6	a06. exe	35,106	0.9561904	0.9653094	0.9826548	0.9907856	0.9993476	0.9999126	0.9998999	0.9999326
7	a07. exe	52,032	0.9815782	0.9716229	0.9863654	0.9923842	0.9984289	0.9999999	0.9999144	0.9999608
8	a08. exe	145,387	0.9994093	0.9542987	0.9689376	0.9717845	0.9991427	0.9999638	0.9999091	0.9998902
9	a09. exe	248,273	0.9825782	0.9751095	0.9892873	0.9943767	0.9984963	0.9998971	0.9999740	0.9999689
10	a10. exe	478,321	0.9570274	0.9672633	0.9783215	0.9897842	0.9984524	0.9999957	0.9999571	0.9999874
11	a11. exe	738,275	0.9662907	0.9652982	0.9724657	0.9813649	0.9982631	0.9999999	0.9999675	0.9999726
12	a12. exe	1,594,276	0.9394872	0.9872064	0.9952982	0.9973864	0.9983274	0.9999651	0.9999968	0.9999856
13	a13. exe	2,273,670	0.9742903	0.9598736	0.9683764	0.9752745	0.9972754	0.9999920	0.9999713	0.9999805
14	a14. exe	2,985,306	0.9773763	0.9531808	0.9642194	0.9712736	0.9993672	0.9999939	0.9999890	0.9999809
15	a15. exe	3,412,639	0.9869036	0.9721766	0.9854388	0.9913549	0.9993878	0.9999905	0.9999933	0.9999914
16	a16. exe	3,872,984	0.9883785	0.9783229	0.9873095	0.9932635	0.9991872	0.9999932	0.9999925	0.9999962
17	a17. exe	4,038,387	0.9941874	0.9763861	0.9836724	0.9914907	0.9993783	0.9999880	0.9999998	0.9999980
18	a18. exe	5,284,796	0.9885673	0.9673525	0.9783755	0.9852845	0.9991526	0.9999902	0.9999975	0.9999949
19	a19. exe	5,628,037	0.9851873	0.9589654	0.9693874	0.9931094	0.9985212	0.9999984	0.9999925	0.9999935
20	a20. exe	6,735,934	0.9853095	0.9742098	0.9866733	0.9934287	0.9997341	0.9999966	0.9999804	0.9999851
Average			0.9651026	0.9659739	0.9773440	0.9850070	0.9988140	0.9997574	0.9992658	0.9996038

Table: 7.71
Comparisons of Strict Avalanche of *.exe* files

Serial no.	**Source file name**	**Source file size (in bytes)**	**CGTHLPSCT**	**CTHLPSCT**	**CDHLPSCT**	**DHLPSCT**	**KSOMSCT**	**RSA**	**TDES**	**AES**
1	a01. exe	1,063	0.8980487	0.9538748	0.9630982	0.9739849	0.9986732	0.99331106	0.9792702	0.9838478
2	a02. exe	2,518	0.9064903	0.9626484	0.9753883	0.9837655	0.9983764	0.9976206	0.9946514	0.9962429
3	a03. exe	8,250	0.8672654	0.9478293	0.9616638	0.9703771	0.9947836	0.9973483	0.9966029	0.9976199
4	a04. exe	15,937	0.9502891	0.9526151	0.9696303	0.9784773	0.9983764	0.9989716	0.9986018	0.9990597
5	a05. exe	22,874	0.9852897	0.9514398	0.9663436	0.9752445	0.9978732	0.9996162	0.9991326	0.9995076
6	a06. exe	35,106	0.9541942	0.9612934	0.9790373	0.9894662	0.9983635	0.9996864	0.9996162	0.9995609
7	a07. exe	52,032	0.9802675	0.9672902	0.9846101	0.9916732	0.9973908	0.9996280	0.9997134	0.9997644
8	a08. exe	145,387	0.9972765	0.9498366	0.9650964	0.9704883	0.9987610	0.9998091	0.9997664	0.9998125
9	a09. exe	248,273	0.9825696	0.9716309	0.9875643	0.9938774	0.9973876	0.9998250	0.9997608	0.9998805
10	a10. exe	478,321	0.9562385	0.9625058	0.9759475	0.9881088	0.9971762	0.9998954	0.9998640	0.9999289
11	a11. exe	738,275	0.9642341	0.9593763	0.9697334	0.9803846	0.9978367	0.9999389	0.9998806	0.9998947
12	a12. exe	1,594,276	0.9389045	0.9857250	0.9919845	0.99678364	0.9976514	0.9999356	0.9999451	0.9999219
13	a13. exe	2,273,670	0.9725409	0.9561093	0.9653994	0.9737836	0.9968934	0.9999454	0.9999480	0.9999410
14	a14. exe	2,985,306	0.9786532	0.9498302	0.9635540	0.9703985	0.9986563	0.9999526	0.9999352	0.9999626
15	a15. exe	3,412,639	0.9867054	0.9673629	0.9817645	0.9903883	0.9983762	0.9999608	0.9999742	0.9999688
16	a16. exe	3,872,984	0.9870965	0.9735198	0.9846773	0.9927846	0.9987106	0.9999808	0.9999419	0.9999440
17	a17. exe	4,038,387	0.9876509	0.9716309	0.9817056	0.9903881	0.9983765	0.9999732	0.9999588	0.9999696
18	a18. exe	5,284,796	0.9863476	0.9659046	0.9759474	0.9837762	0.9983761	0.9999582	0.9999564	0.9999630
19	a19. exe	5,628,037	0.9835694	0.9528947	0.9674095	0.9927740	0.9973654	0.9999731	0.999975	0.9999631
20	a20. exe	6,735,934	0.9842397	0.9698369	0.9838747	0.9927833	0.9983766	0.9999714	0.9998772	0.9999264
Average			0.9623935	0.9616580	0.9747220	0.9839850	0.9978890	0.9992551	0.9983186	0.9987340

Table: 7.72
Comparisons of Bit Independence of *.exe* files

Serial no.	**Source file name**	**Source file size (in bytes)**	**CGTHLPSCT**	**CTHLPSCT**	**CDHLPSCT**	**DHLPSCT**	**KSOMSCT**	**RSA**	**TDES**	**AES**
1	a01. exe	1,063	0.6108763	0.5289719	0.4920094	0.4783436	0.5289540	0.3636724	0.5416969	0.4971214
2	a02. exe	2,518	0.3442897	0.6309387	0.0729941	0.0464390	0.0639870	0.0214885	0.2405326	0.1741851
3	a03. exe	8,250	0.6644390	0.5817843	0.4277464	0.3328974	0.3632980	0.3356391	0.2903861	0.3252181
4	a04. exe	15,937	0.8056424	0.5189645	0.8178835	0.7129076	0.7322900	0.7236569	0.7254496	0.7247688
5	a05. exe	22,874	0.8067548	0.573429	0.7628934	0.8348739	0.7815090	0.7703270	0.7664000	0.7695320
6	a06. exe	35,106	0.8340986	0.5980435	0.8177453	0.8539085	0.8218730	0.8166921	0.8001057	0.8148537
7	a07. exe	52,032	0.7905674	0.7187409	0.7261098	0.7834290	0.7529870	0.7465041	0.7475072	0.7492728
8	a08. exe	145,387	0.7442381	0.6040953	0.7560109	0.6853054	0.7043870	0.7055450	0.7057941	0.7047457
9	a09. exe	248,273	0.7856327	0.6386097	0.9328874	0.7756439	0.7321720	0.7468688	0.7390269	0.7494657
10	a10. exe	478,321	0.8793084	0.9927432	0.8177302	0.8950934	0.8601180	0.8585859	0.8567059	0.8596827
11	a11. exe	738,275	0.8398745	0.9174045	0.7218849	0.8128056	0.8319530	0.8317745	0.8228424	0.8297953
12	a12. exe	1,594,276	0.6635490	0.817834	0.6529984	0.6943209	0.6403280	0.6280267	0.6345553	0.6288788
13	a13. exe	2,273,670	0.6589378	0.9630738	0.6903915	0.6710963	0.6429070	0.6139199	0.6154782	0.6134784
14	a14. exe	2,985,306	0.8124650	0.5219457	0.8510988	0.7942908	0.7834200	0.7817622	0.7791619	0.7830216
15	a15. exe	3,412,639	0.9999846	0.9710954	0.8377294	0.9985632	0.9942170	0.9846006	0.9844929	0.9845419
16	a16. exe	3,872,984	0.8208465	0.9543093	0.8947566	0.7921764	0.7943920	0.7780248	0.7852720	0.7774266
17	a17. exe	4,038,387	0.9999287	0.8827609	0.8727714	0.9956390	0.9923960	0.9971851	0.9981982	0.9973744
18	a18. exe	5,284,796	0.8853005	0.9576412	0.7827361	0.7743298	0.7573789	0.7497550	0.7606758	0.7593839
19	a19. exe	5,628,037	0.8018475	0.7890646	0.7928844	0.6754900	0.7230965	0.6987982	0.7128113	0.7018017
20	a20. exe	6,735,934	0.6697569	0.9754893	0.7763263	0.5894536	0.5732987	0.5658394	0.5776820	0.5597410
Average			0.7709169	0.7568470	0.7248790	0.7098500	0.7037480	0.7330390	0.7042388	0.7002145

Table: 7.73

Comparisons of average values of Avalanche, Strict Avalanche and Bit Independence of *.exe* files

Techniques	**Average values of**		
	Avalanche	**Strict Avalanche**	**Bit Independence**
CGTHLPSCT	0.9651026	0.9623935	0.7709169
CTHLPSCT	0.9659739	0.9616580	0.7568470
CDHLPSCT	0.9773440	0.9747220	0.7248790
DHLPSCT	0.9850070	0.9839850	0.7098500
KSOMSCT	0.9988140	0.9978890	0.703748
RSA	0.9997574	0.9992551	0.7330390
TDES	0.9992658	0.9983186	0.7042388
AES	0.9996038	0.9987340	0.7002145

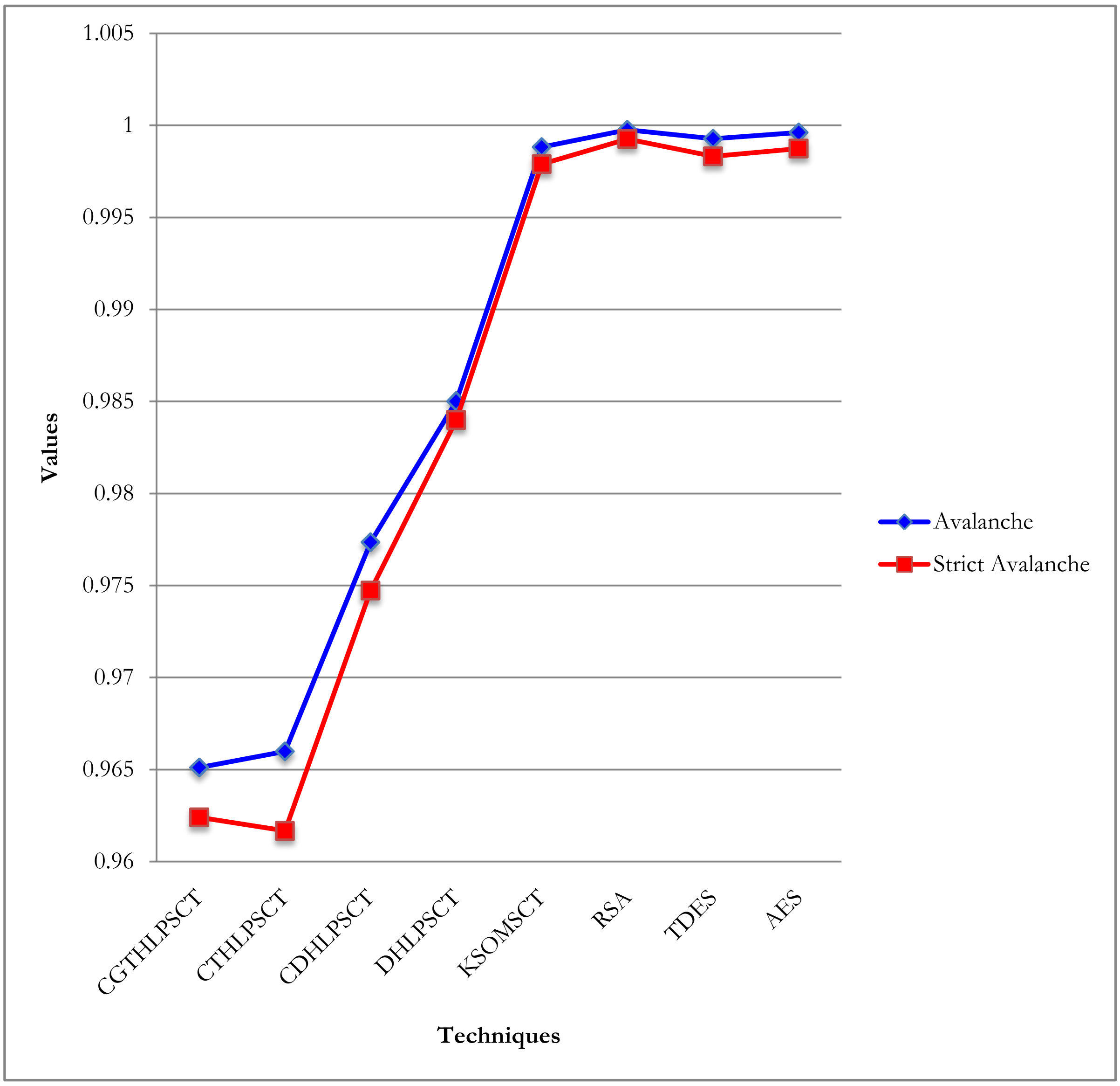


Figure 7.61: Pictorial representation of the average values of Avalanche and Strict Avalanche of *.exe* type bit stream

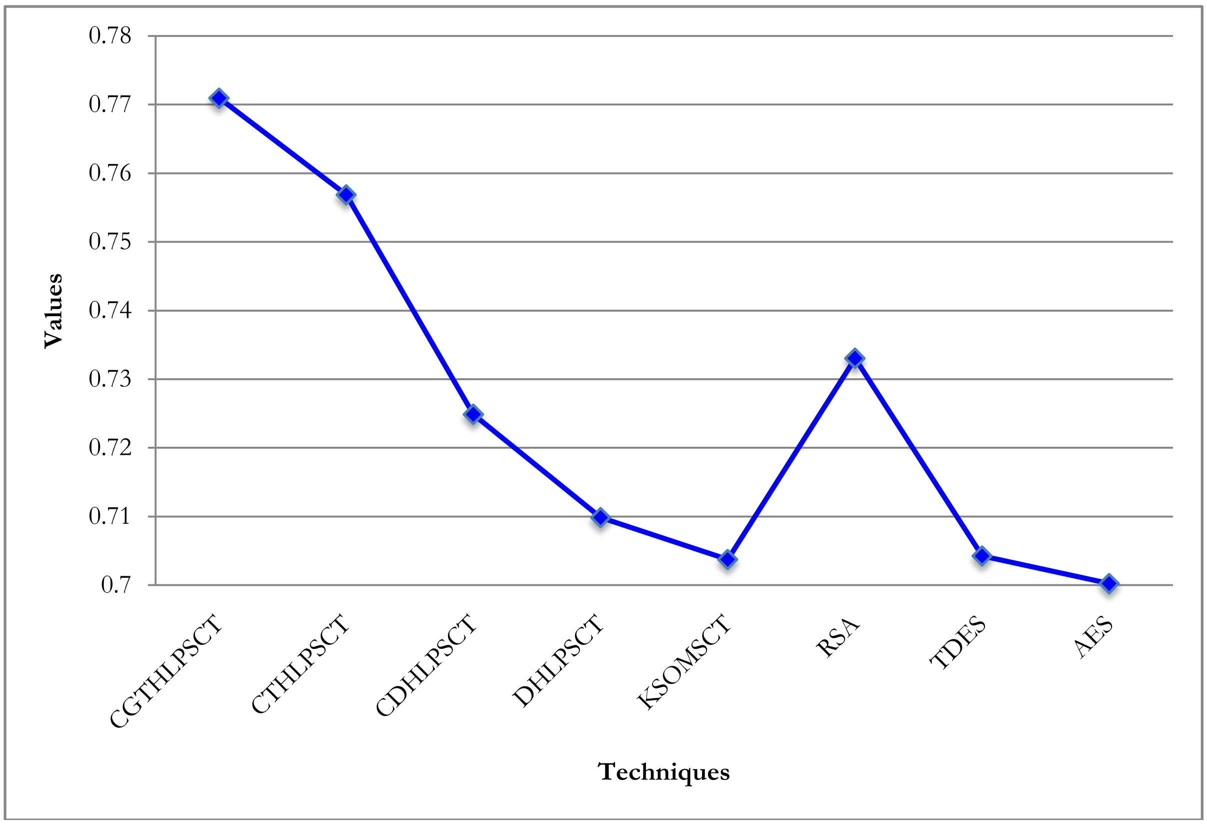


Figure 7.62: Pictorial representation of the average values of Bit Independence of *.exe* type bit stream

7.5.3 *.txt* files

Twenty *.txt* files of different sizes varying from 1,504 bytes to 6,702,831 bytes have been taken for Avalanche, Strict Avalanche and Bit Independence test. Table 7.75, 7.76, 7.77 and 7.78 shows the Avalanche, Strict Avalanche and Bit Independence test of *.txt* type files obtained using proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing RSA, TDES, AES respectively. Avalanches, Strict Avalanche test value for each file are very close to the value 1 for all four techniques. Bit Independence values vary from 0.0328965 to 0.5923987 for KSOMSCT, from 0.0425987 to 0.5983409 for DHLPSCT, from 0.0873611 to 0.5983774 for CDHLPSCT, from 0.1209735 to 0.5920943 for CTHLPSCT, from 0.0296755 to 0.5932186 for CGTHLPSCT, from 0.0214885 to 0.9971851 for RSA, from 0.2405326 to 0.9981982 for TDES and from 0.1741851 to 0.9973744 for AES respectively.

Figure 7.63 and 7.64 show the graphical representation of the comparison of results of Avalanche, Strict Avalanche (average values) and Bit Independence test results (average values) respectively of the *.txt* type source files for proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing RSA, TDES, AES techniques. Average Avalanche values of proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing RSA, TDES, AES are 0.9980140, 0.9948260, 0.9919460, 0.9823468, 0.9786178, 0.9998823, 0.9997381, and 0.9998726 respectively. Average Strict Avalanche value of proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing RSA, TDES, AES are 0.9966320, 0.9922320, 0.9866200, 0.9814860, 0.9754595, 0.9994315, 0.9992106 and 0.9996183 respectively. Average Bit Independence values of proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing RSA, TDES, AES are 0.3304380, 0.3531750, 0.4097800, 0.4214050, 0.4426890, 0.3234268, 0.3016146, and 0.3112921 respectively. Proposed technique has the highest average Bit Independence value which indicates that proposed technique provides better degree of security and comparable to other techniques.

Table: 7.74
Comparisons of Avalanche of *.txt* files

Serial no.	**Source file name**	**Source file size (in bytes)**	**CGTHLPSCT**	**CTHLPSCT**	**CDHLPSCT**	**DHLPSCT**	**KSOMSCT**	**RSA**	**TDES**	**AES**
1	a01.txt	1,504	0.8956478	0.9773299	0.9900538	0.9918745	0.9928543	0.9999623	0.9968761	0.9991553
2	a02.txt	7,921	0.9186569	0.9783421	0.9930256	0.9943876	0.9965489	0.9979894	0.9993790	0.9990676
3	a03.txt	17,036	0.8885646	0.9922907	0.9952093	0.9972645	0.9921756	0.9999004	0.9993587	0.9999557
4	a04.txt	44.624	0.9928951	0.9932198	0.9963087	0.9978374	0.9995439	0.9999831	0.9999334	0.9998811
5	a05.txt	68,823	0.9864350	0.9923197	0.9952901	0.9962937	0.9974298	0.9999456	0.9998621	0.9997820
6	a06.txt	161,935	0.9926796	0.9794531	0.9931097	0.9983764	0.9992376	0.9999865	0.9998455	0.9999418
7	a07.txt	328,017	0.9927785	0.9621986	0.989429	0.9973645	0.9997543	0.9999802	0.9999852	0.9999104
8	a08.txt	587,290	0.9934705	0.964298	0.9893864	0.9927464	0.9985640	0.9999671	0.9998538	0.9999568
9	a09.txt	1,049,763	0.9933098	0.9762863	0.9853095	0.9893746	0.9996548	0.9999908	0.9999060	0.9999283
10	a10.txt	1,418,025	0.9912906	0.9873197	0.9932904	0.9963874	0.9997865	0.9999953	0.9999992	0.9999984
11	a11.txt	1,681,329	0.9946879	0.9752095	0.9829076	0.9872645	0.9987698	0.9999980	0.9999442	0.9999935
12	a12.txt	2,059,318	0.9927847	0.9831987	0.9918639	0.9963865	0.9992365	0.9999918	0.9999826	0.9999639
13	a13.txt	2,618,492	0.9948041	0.9872092	0.9963093	0.9973874	0.9997658	0.9999996	0.9999669	0.9999855
14	a14.txt	3,154,937	0.9928768	0.9912865	0.9963094	0.9972864	0.9980767	0.9999902	0.9999494	0.9999964
15	a15.txt	4,073,829	0.9918739	0.9863986	0.9942095	0.995298	0.9995647	0.9999901	0.9999857	0.9999991
16	a16.txt	4,936,521	0.9919378	0.9872987	0.9942985	0.9973875	0.9993214	0.9999905	0.9999972	0.9999834
17	a17.txt	5,125,847	0.9927809	0.983429	0.9912753	0.9962874	0.9986754	0.9999885	0.9999846	0.9999983
18	a18.txt	5,593,219	0.9937892	0.9893421	0.9942907	0.9951094	0.9975679	0.9999998	0.9999740	0.9999858
19	a19.txt	5,898,302	0.9893092	0.9852976	0.9913865	0.9931093	0.9971648	0.9999968	0.9999840	0.9999761
20	a20.txt	6,702,831	0.9917832	0.9752098	0.9856490	0.9890947	0.9965890	0.9999991	0.9999910	0.9999934
Average			0.9786178	0.9823468	0.9919460	0.9948260	0.9980140	0.9998823	0.9997381	0.9998726

Table: 7.75
Comparisons of Strict Avalanche of *.txt* files

Serial no.	**Source file name**	**Source file size (in bytes)**	**CGTHLPSCT**	**CTHLPSCT**	**CDHLPSCT**	**DHLPSCT**	**KSOMSCT**	**RSA**	**TDES**	**AES**
1	a01.txt	1,504	0.8817383	0.9769374	0.9975644	0.9893774	0.9908736	0.9928458	0.9914139	0.9970272
2	a02.txt	7,921	0.9029049	0.9771093	0.9918736	0.9928947	0.9957835	0.9972093	0.9972475	0.9983127
3	a03.txt	17,036	0.8742984	0.9911091	0.9928745	0.9962883	0.9918742	0.9993831	0.9988550	0.9989337
4	a04.txt	44.624	0.9927840	0.9926843	0.9928764	0.9957836	0.9982836	0.9998628	0.9998166	0.9997219
5	a05.txt	68,823	0.9859303	0.9913732	0.9937741	0.9941021	0.9957837	0.9998719	0.9994130	0.9996298
6	a06.txt	161,935	0.9903874	0.9780186	0.9910837	0.9978203	0.9982621	0.9998412	0.9994992	0.9997208
7	a07.txt	328,017	0.9928467	0.9617355	0.9867368	0.9967925	0.9983119	0.9999293	0.9996606	0.9997932
8	a08.txt	587,290	0.9930849	0.9639037	0.9872982	0.9903822	0.9973653	0.9999376	0.9996230	0.9998149
9	a09.txt	1,049,763	0.9928768	0.9748208	0.9183986	0.9625292	0.9946732	0.9999488	0.9996680	0.9998524
10	a10.txt	1,418,025	0.9904843	0.9863932	0.9908117	0.9950923	0.9982663	0.9999877	0.9999034	0.9999336
11	a11.txt	1,681,329	0.9938949	0.9746489	0.9793763	0.9856930	0.9963672	0.9999784	0.9999008	0.9999667
12	a12.txt	2,059,318	0.9904955	0.9817393	0.9873635	0.9937299	0.9981095	0.9999770	0.9998949	0.9999475
13	a13.txt	2,618,492	0.9938054	0.9866289	0.9938927	0.9956383	0.9984763	0.9999879	0.9999146	0.9999552
14	a14.txt	3,154,937	0.9893049	0.9908373	0.9938709	0.9960937	0.9970920	0.9999777	0.9999097	0.9999748
15	a15.txt	4,073,829	0.9917478	0.9851094	0.9910957	0.9947836	0.9982764	0.9999818	0.9999328	0.9999792
16	a16.txt	4,936,521	0.9918494	0.9862018	0.9928746	0.9969037	0.9982776	0.9999772	0.9999167	0.9999692
17	a17.txt	5,125,847	0.9904783	0.9826382	0.9875091	0.9957722	0.9973653	0.9999791	0.9999298	0.9999700
18	a18.txt	5,593,219	0.9927484	0.9881273	0.9928943	0.9947822	0.9967831	0.9999899	0.9999119	0.9999651
19	a19.txt	5,898,302	0.9881632	0.9847219	0.9873776	0.9927831	0.9967389	0.9999846	0.9999090	0.9999444
20	a20.txt	6,702,831	0.9893673	0.9749856	0.9828433	0.9873992	0.9956738	0.9999793	0.9998914	0.9999537
Average			0.9754595	0.9814860	0.9866200	0.9922320	0.9966320	0.9994315	0.9992106	0.9996183

Table: 7.76
Comparisons of Bit Independence of *.txt* files

Serial no.	**Source file name**	**Source file size (in bytes)**	**CGTHLPSCT**	**CTHLPSCT**	**CDHLPSCT**	**DHLPSCT**	**KSOMSCT**	**RSA**	**TDES**	**AES**
1	a01.txt	1,504	0.2390473	0.1209735	0.0928744	0.2319809	0.0845397	0.0028819	0.0187584	0.0465815
2	a02.txt	7,921	0.1347038	0.3489373	0.2309845	0.2542983	0.1534298	0.1529343	0.0246086	0.0763064
3	a03.txt	17,036	0.0296755	0.1963931	0.0873611	0.0425987	0.0328965	0.0395585	0.0060197	0.0326123
4	a04.txt	44.624	0.4536897	0.2190328	0.4903984	0.4982769	0.4895649	0.4469041	0.4277136	0.4260388
5	a05.txt	68,823	0.5332578	0.3782308	0.5119838	0.5329077	0.5534208	0.5227443	0.4571534	0.5084995
6	a06.txt	161,935	0.5573468	0.4830933	0.5983774	0.5129076	0.5723474	0.5323910	0.4991341	0.5109318
7	a07.txt	328,017	0.5708478	0.5920943	0.5827634	0.5630986	0.5923987	0.5452650	0.5410970	0.5215336
8	a08.txt	587,290	0.5634219	0.5688034	0.6728463	0.5983409	0.5423120	0.5381117	0.5147633	0.5197619
9	a09.txt	1,049,763	0.562349	0.4782932	0.5189336	0.4739808	0.2178594	0.5169141	0.4310866	0.4486184
10	a10.txt	1,418,025	0.5432589	0.4103896	0.5518832	0.2709865	0.2914653	0.2404651	0.2557573	0.2527478
11	a11.txt	1,681,329	0.532498	0.4891082	0.3897345	0.3129876	0.2763409	0.2372808	0.2477444	0.2475468
12	a12.txt	2,059,318	0.3236797	0.3827203	0.2989043	0.2754398	0.3523098	0.3710863	0.3377319	0.3386753
13	a13.txt	2,618,492	0.5485358	0.5341082	0.3980774	0.2896754	0.2623197	0.2424210	0.2605189	0.2561983
14	a14.txt	3,154,937	0.5561906	0.4901827	0.4389284	0.2187906	0.2309658	0.2402317	0.2539287	0.2525674
15	a15.txt	4,073,829	0.5479423	0.4261972	0.3827761	0.2897452	0.2512986	0.2424554	0.2613233	0.2572494
16	a16.txt	4,936,521	0.3861308	0.5389209	0.3620984	0.3908675	0.3129087	0.3271911	0.3110799	0.3180304
17	a17.txt	5,125,847	0.2690358	0.4671971	0.3908487	0.312894	0.3165498	0.2729138	0.2604710	0.2786139
18	a18.txt	5,593,219	0.5207415	0.4092788	0.3190475	0.3328731	0.3218954	0.3008293	0.2957896	0.2986022
19	a19.txt	5,898,302	0.5932186	0.4673092	0.4872948	0.2896408	0.3617583	0.3366394	0.3037460	0.3074793
20	a20.txt	6,702,831	0.3882044	0.4268272	0.3894759	0.3712054	0.3921711	0.3593174	0.3238654	0.3272460
Average			0.442689	0.4214050	0.4097800	0.3531750	0.3304380	0.3234268	0.3016146	0.3112921

Table: 7.77
Comparisons of average values of Avalanche, Strict Avalanche and Bit Independence of *.txt* files

Techniques	**Average values of**		
	Avalanche	**Strict Avalanche**	**Bit Independence**
CGTHLPSCT	0.9786178	0.9754595	0.442689
CTHLPSCT	0.9823468	0.9814860	0.4214050
CDHLPSCT	0.9919460	0.9866200	0.4097800
DHLPSCT	0.9948260	0.9922320	0.3531750
KSOMSCT	0.9980140	0.9966320	0.3304380
RSA	0.9998823	0.9994315	0.3234268
TDES	0.9997381	0.9992106	0.3016146
AES	0.9998726	0.9996183	0.3112921

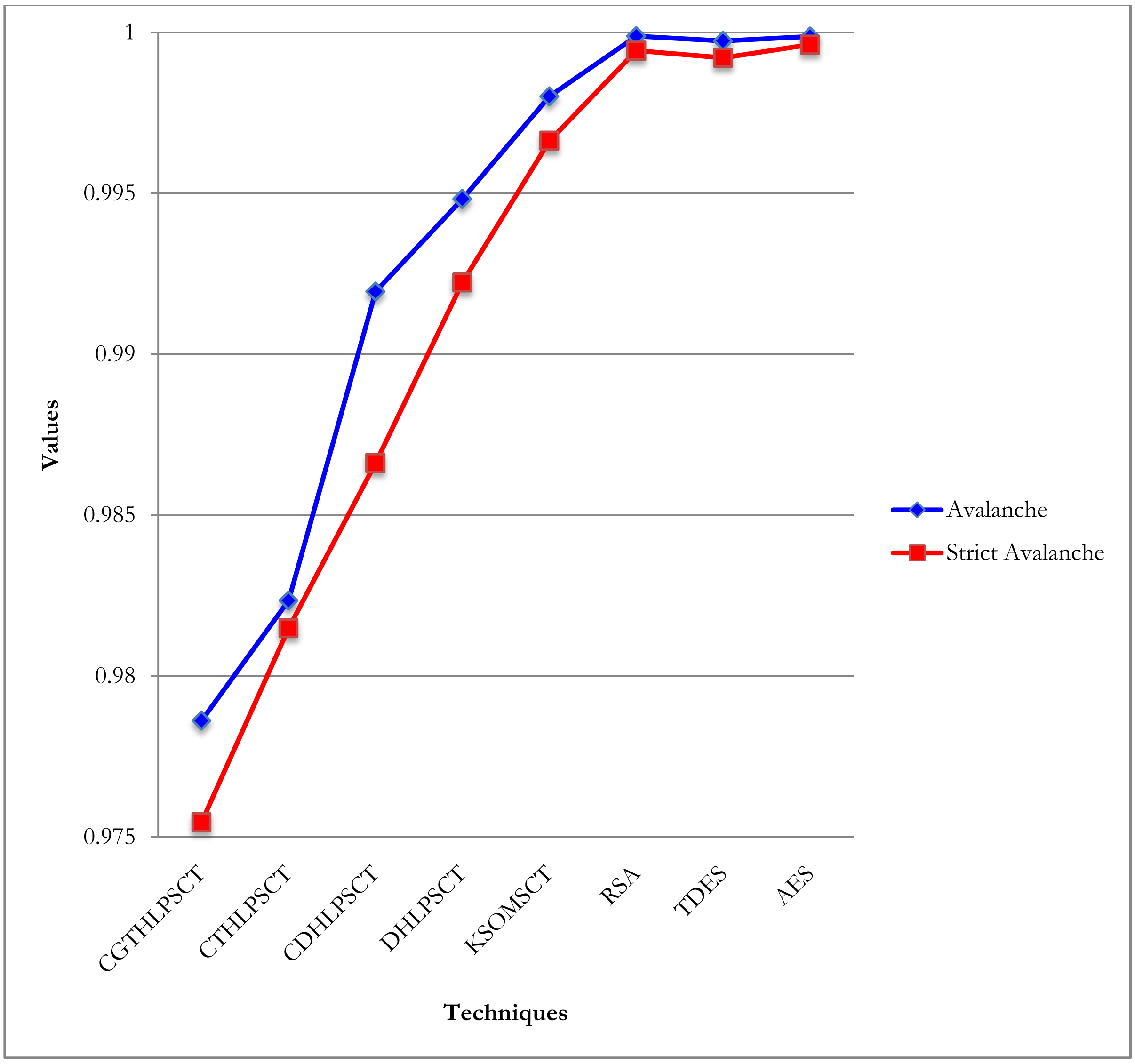


Figure 7.63: Pictorial representation of the average values of Avalanche and Strict Avalanche of *.txt* type bit stream

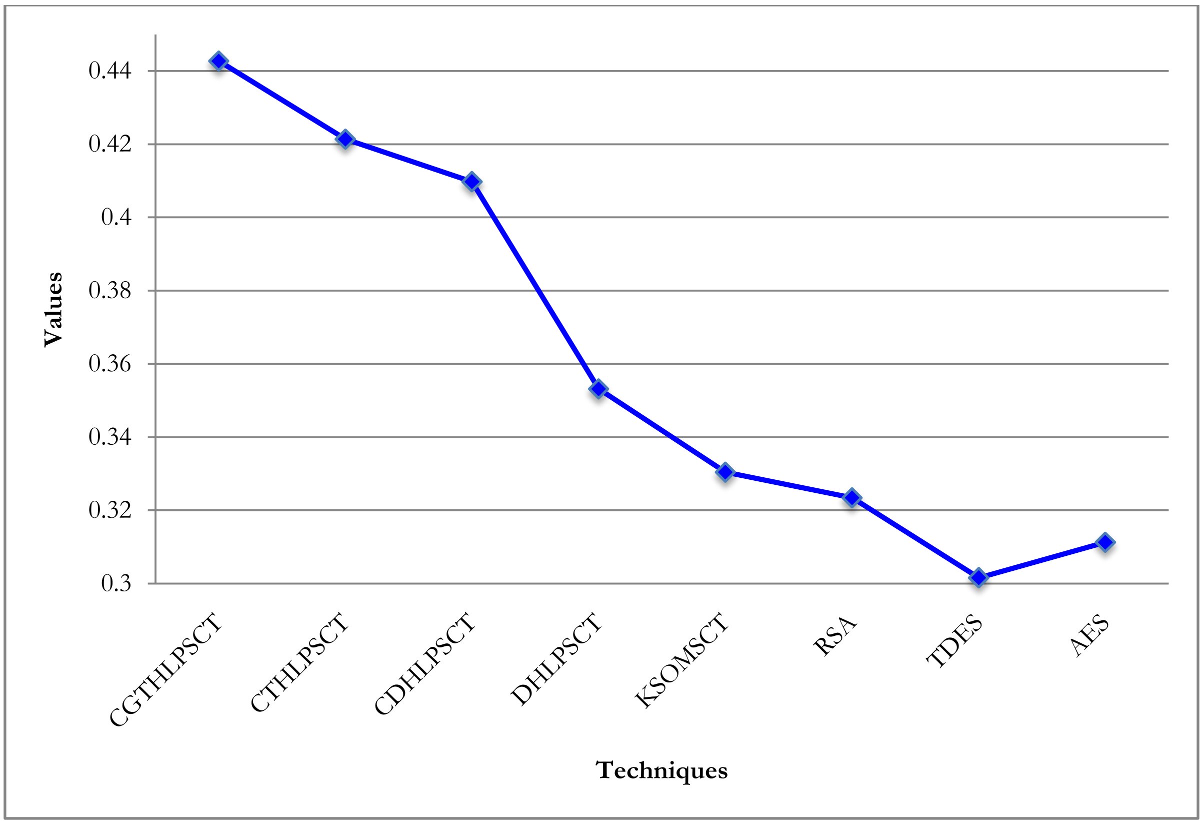


Figure 7.64: Pictorial representation of the average values of Bit Independence of *.txt* type bit stream

7.5.4 *.doc* files

Twenty *.doc* files of different sizes varying from 21,052 bytes to 5,472,298 bytes have been taken for Avalanche, Strict Avalanche and Bit Independence test. Table 7.79, 7.80, 7.81 and 7.82 shows the Avalanche, Strict Avalanche and Bit Independence test of *.doc* type files obtained using proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing RSA, TDES, AES respectively. Avalanches, Strict Avalanche test value for each file are very close to the value 1 for all four techniques. Bit Independence values vary from 0.2409878 to 0.9357209 for KSOMSCT, from 0.2730959 to 0.9529072 for DHLPSCT, from 0.2890763 to 0.9345298 for CDHLPSCT, from 0.3178352 to 0.9424093 for CTHLPSCT, from 0.3358732 to 0.9917649 for CGTHLPSCT, from 0.0140414 to 0.8851778 for RSA, from 0.2257191 to 0.9180687 for TDES and from 0.518730 to 0.9019331 for AES respectively.

Figure 7.65 and 7.66 show the graphical representation of the comparison of results of Avalanche, Strict Avalanche (average values) and Bit Independence test results (average values) respectively of the *.doc* type source files for proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing RSA, TDES, AES techniques. Average Avalanche values of proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing RSA, TDES, AES are 0.9986350, 0.9897060, 0.9843530, 0.9739355, 0.9698779, 0.9999707, 0.9999233, and 0.9999362 respectively. Average Strict Avalanche values of proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing RSA, TDES, AES are 0.9974800, 0.9878430, 0.9816040, 0.9723390, 0.9680110, 0.9998032, 0.9997301, and 0.9997919 respectively. Average Bit Independence values of proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT and existing RSA, TDES, AES are 0.7664320, 0.7851330, 0.7883700, 0.8014841, 0.8237869, 0.7353065, 0.7611090, and 0.7484538 respectively. Proposed technique has the highest average Bit Independence value which indicates that proposed technique provides better degree of security and comparable to other techniques.

Table: 7.78
Comparisons of Avalanche of *.doc* files

Serial no.	**Source file name**	**Source file size (in bytes)**	**CGTHLPSCT**	**CTHLPSCT**	**CDHLPSCT**	**DHLPSCT**	**KSOMSCT**	**RSA**	**TDES**	**AES**
1	a01. doc	21,052	0.8217485	0.9683764	0.9786549	0.9874534	0.9987645	0.9999730	0.9991441	0.9996181
2	a02. doc	33,897	0.9428461	0.9763987	0.9874302	0.9916743	0.997645	0.9999252	0.9998423	0.9997996
3	a03. doc	45,738	0.9245618	0.9693875	0.9821896	0.9874309	0.9994287	0.9998809	0.9998321	0.9997896
4	a04. doc	75,093	0.9724750	0.9869873	0.9956348	0.9984532	0.999875	0.9999732	0.9999611	0.9999513
5	a05. doc	106,872	0.9601832	0.9763617	0.9896743	0.9912765	0.9953429	0.9999922	0.9999624	0.9998099
6	a06. doc	327.054	0.9851273	0.9694654	0.9786534	0.9865875	0.9995643	0.9999793	0.9999608	0.9999998
7	a07. doc	582,831	0.9883795	0.9673645	0.9755389	0.9823186	0.9996754	0.9999912	0.9999880	0.9999918
8	a08. doc	729,916	0.9892774	0.9706539	0.9772987	0.9873421	0.9980756	0.9999963	0.9999943	0.9999647
9	a09. doc	1,170,251	0.9772001	0.9714203	0.9863907	0.9912897	0.9985644	0.9999859	0.9999935	0.9999945
10	a10. doc	1,749,272	0.9851294	0.9783765	0.9912876	0.9934298	0.9943209	0.9998950	0.9999859	0.9999971
11	a11. doc	2,045,805	0.9845851	0.981549	0.9934291	0.996439	0.9995423	0.9999846	0.9999721	0.9999560
12	a12. doc	2,372,014	0.9841272	0.9732764	0.9815632	0.9875342	0.9996734	0.9999984	0.9999920	0.9999839
13	a13. doc	2,869,275	0.9847452	0.9701846	0.9832897	0.9896755	0.9990675	0.9999712	0.9999940	0.9999827
14	a14. doc	3,161,353	0.9803853	0.9673844	0.9712896	0.9765397	0.9997637	0.9999677	0.9999706	0.9999901
15	a15. doc	3,570,295	0.9842737	0.9723642	0.9848897	0.9912653	0.998342	0.9999903	0.9999811	0.9999728
16	a16. doc	3,834,427	0.9742775	0.9732109	0.9852398	0.9934299	0.9991732	0.9999765	0.9999958	0.9999682
17	a17. doc	4,011,986	0.9861843	0.9826542	0.9941259	0.9967043	0.9984537	0.9999873	0.9999839	0.9999967
18	a18. doc	4,562,385	0.9886342	0.9883091	0.9917589	0.9932907	0.9993211	0.9999667	0.9999632	0.9999791
19	a19. doc	4,839,102	0.9951087	0.9633541	0.9754332	0.9832874	0.9993427	0.9999897	0.9999825	0.9999854
20	a20.doc	5,472,298	0.9883086	0.9716309	0.9832897	0.9887065	0.9987655	0.9999902	0.9999663	0.9999926
Average			0.9698779	0.9739355	0.9843530	0.9897060	0.9986350	0.9999707	0.9999233	0.9999362

Table: 7.79
Comparisons of Strict Avalanche of *.doc* files

Serial no.	**Source file name**	**Source file size (in bytes)**	**CGTHLPSCT**	**CTHLPSCT**	**CDHLPSCT**	**DHLPSCT**	**KSOMSCT**	**RSA**	**TDES**	**AES**
1	a01. doc	21,052	0.8107439	0.9678212	0.9749383	0.9850934	0.9978107	0.9988219	0.9984031	0.9991557
2	a02. doc	33,897	0.9402374	0.9740937	0.9884091	0.9883746	0.9956734	0.9996332	0.9990557	0.9994002
3	a03. doc	45,738	0.9183651	0.9672353	0.9799047	0.9873094	0.9984762	0.9995262	0.9991422	0.9994292
4	a04. doc	75,093	0.9723994	0.9853428	0.9927585	0.9973641	0.9983661	0.9997938	0.9997298	0.9995753
5	a05. doc	106,872	0.9592052	0.9751095	0.9886498	0.9893746	0.9948776	0.9997412	0.9997946	0.9996929
6	a06. doc	327.054	0.9843788	0.9682687	0.9749037	0.9834857	0.9980398	0.9998934	0.9997839	0.9997969
7	a07. doc	582,831	0.9867093	0.9662839	0.9718395	0.9817363	0.9983842	0.9997936	0.9998611	0.9998608
8	a08. doc	729,916	0.9871874	0.9682082	0.9753983	0.9860935	0.9973534	0.9997095	0.9998100	0.9998740
9	a09. doc	1,170,251	0.9741093	0.9701864	0.9838927	0.9897564	0.9973653	0.9998811	0.9998950	0.9999014
10	a10. doc	1,749,272	0.9841047	0.9773086	0.9872928	0.9917305	0.9936382	0.9998167	0.9998824	0.9998774
11	a11. doc	2,045,805	0.9836295	0.9803764	0.9893736	0.9950937	0.9983641	0.9999205	0.9999249	0.9998642
12	a12. doc	2,372,014	0.9831774	0.9715743	0.9774092	0.9863784	0.9983764	0.9999281	0.9998540	0.9998760
13	a13. doc	2,869,275	0.9813673	0.9683832	0.9793737	0.9883776	0.9972982	0.9999547	0.9999184	0.9999072
14	a14. doc	3,161,353	0.9813094	0.9662282	0.9686362	0.9746339	0.9982161	0.9999540	0.9999256	0.9999271
15	a15. doc	3,570,295	0.9847628	0.9702737	0.9818935	0.9883094	0.9973522	0.9999580	0.9999182	0.9999419
16	a16. doc	3,834,427	0.9741093	0.9716383	0.9829848	0.9903875	0.9983640	0.9999384	0.9999493	0.9999254
17	a17. doc	4,011,986	0.9851038	0.98028372	0.9918376	0.9948223	0.9974654	0.9999478	0.9999443	0.9999573
18	a18. doc	4,562,385	0.9871673	0.98726372	0.9887303	0.9910486	0.9983675	0.9999482	0.9999367	0.9999654
19	a19. doc	4,839,102	0.9938756	0.96162386	0.9727648	0.9803883	0.9984610	0.9999400	0.9999334	0.9999275
20	a20.doc	5,472,298	0.9882784	0.96926737	0.9810937	0.9871091	0.9973556	0.9999632	0.9999385	0.9999826
Average			0.9680110	0.9723390	0.9816040	0.9878430	0.9974800	0.9998032	0.9997301	0.9997919

Table: 7.80
Comparisons of Bit Independence of *.doc* files

Serial no.	**Source file name**	**Source file size (in bytes)**	**CGTHLPSCT**	**CTHLPSCT**	**CDHLPSCT**	**DHLPSCT**	**KSOMSCT**	**RSA**	**TDES**	**AES**
1	a01. doc	21,052	0.3358732	0.3178352	0.2890763	0.2730959	0.2409878	0.0140414	0.2257191	0.0518730
2	a02. doc	33,897	0.5510947	0.6095723	0.6198462	0.5965847	0.5129085	0.5350126	0.4961753	0.5268791
3	a03. doc	45,738	0.5931972	0.5801674	0.5739820	0.6230975	0.6530981	0.6097290	0.5711997	0.5948656
4	a04. doc	75,093	0.5092879	0.5092846	0.5278450	0.5529873	0.5537764	0.5325364	0.4980972	0.5128289
5	a05. doc	106,872	0.4881084	0.6489016	0.6329075	0.5520950	0.5298700	0.5217317	0.4751121	0.4993628
6	a06. doc	327.054	0.9286513	0.8351093	0.7904765	0.7831983	0.8640932	0.8361893	0.8310899	0.8313195
7	a07. doc	582,831	0.7971563	0.9412096	0.9345298	0.7341866	0.6520948	0.6905605	0.6957219	0.6982151
8	a08. doc	729,916	0.9917649	0.8183482	0.8476205	0.9529072	0.8737160	0.8608413	0.9180687	0.9019331
9	a09. doc	1,170,251	0.9441894	0.8720946	0.8698532	0.8639836	0.7904593	0.8655500	0.8766222	0.8716987
10	a10. doc	1,749,272	0.9603897	0.9424093	0.9275098	0.7921085	0.9165049	0.6997063	0.8466802	0.7936389
11	a11. doc	2,045,805	0.9621765	0.9200314	0.8837201	0.9329076	0.8310955	0.8291049	0.9059827	0.8842572
12	a12. doc	2,372,014	0.9153897	0.8920847	0.8725346	0.8834728	0.8854035	0.8134491	0.8476740	0.8370357
13	a13. doc	2,869,275	0.9267046	0.8917611	0.8518733	0.9045297	0.8840933	0.8353566	0.8611929	0.8518419
14	a14. doc	3,161,353	0.9367893	0.9409184	0.9228745	0.9238976	0.9256109	0.8625301	0.8947058	0.8848101
15	a15. doc	3,570,295	0.9351897	0.8830928	0.8903882	0.8649061	0.9178543	0.8704048	0.8784213	0.8715963
16	a16. doc	3,834,427	0.9592641	0.8109483	0.8920939	0.9342908	0.8109475	0.8851778	0.9020524	0.8937274
17	a17. doc	4,011,986	0.9480284	0.8309285	0.8121908	0.9438964	0.8849077	0.8857170	0.8920854	0.8863351
18	a18. doc	4,562,385	0.9361429	0.9390598	0.9045322	0.8532908	0.7367497	0.8650639	0.8809757	0.8720050
19	a19. doc	4,839,102	0.9359837	0.9256294	0.8294526	0.8630221	0.9357209	0.8585113	0.8746386	0.8651509
20	a20.doc	5,472,298	0.9203571	0.9202947	0.8940921	0.8742099	0.9287563	0.8349159	0.8499651	0.8397011
Average			0.8237869	0.8014841	0.7883700	0.7851330	0.7664320	0.7353065	0.7611090	0.7484538

Table: 7.81
Comparisons of average values of Avalanche, Strict Avalanche and Bit Independence of *.doc* files

Techniques	Average values of		
	Avalanche	Strict Avalanche	Bit Independence
CGTHLPSCT	0.9698779	0.9680110	0.8237869
CTHLPSCT	0.9739355	0.9723390	0.8014841
CDHLPSCT	0.9843530	0.9816040	0.7883700
DHLPSCT	0.9897060	0.9878430	0.7851330
KSOMSCT	0.9986350	0.9974800	0.7664320
RSA	0.9999707	0.9998032	0.7353065
TDES	0.9999233	0.9997301	0.7611090
AES	0.9999362	0.9997919	0.7484538

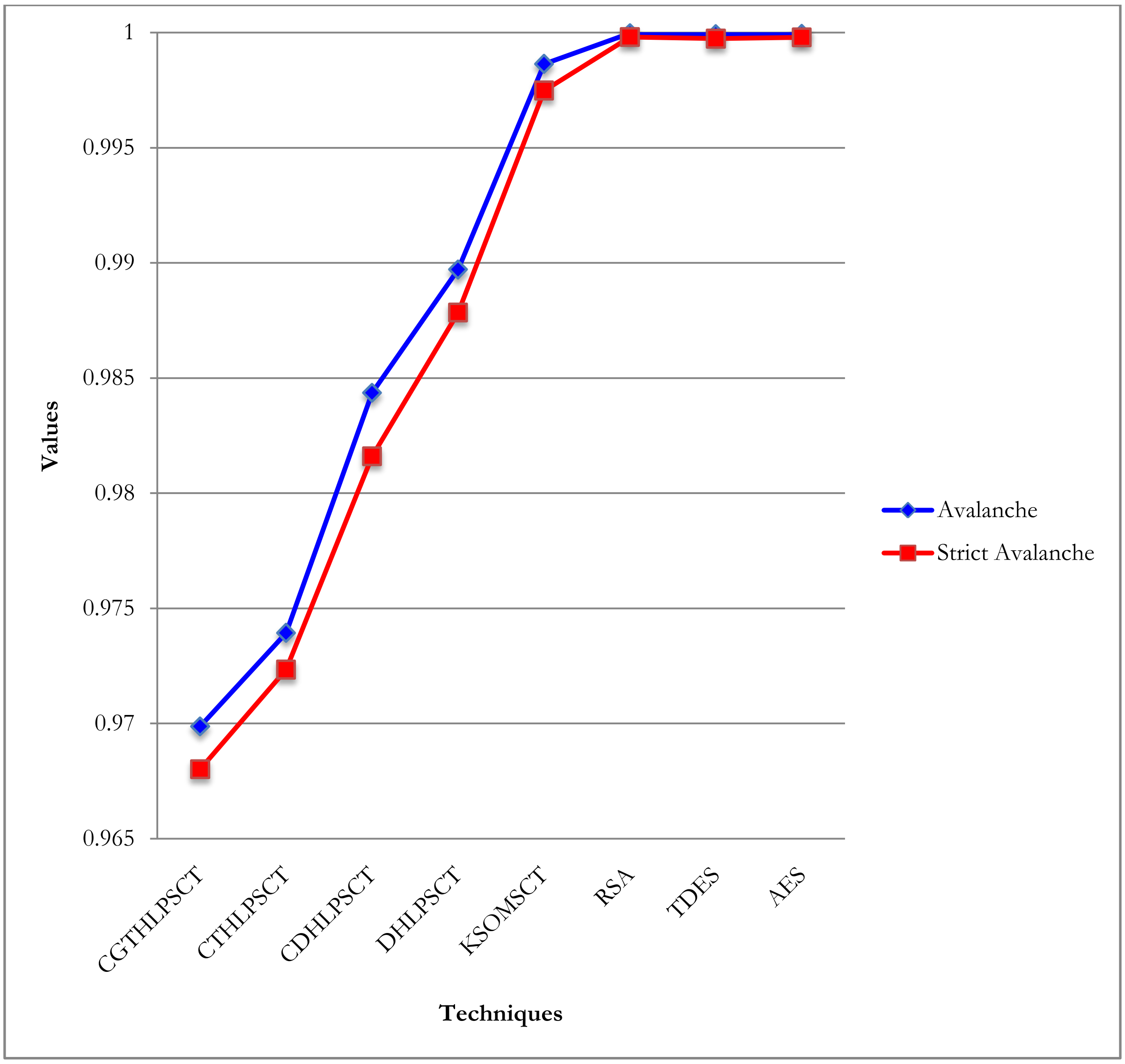


Figure 7.65: Pictorial representation of the average values of Avalanche and Strict Avalanche of *.doc* type bit stream

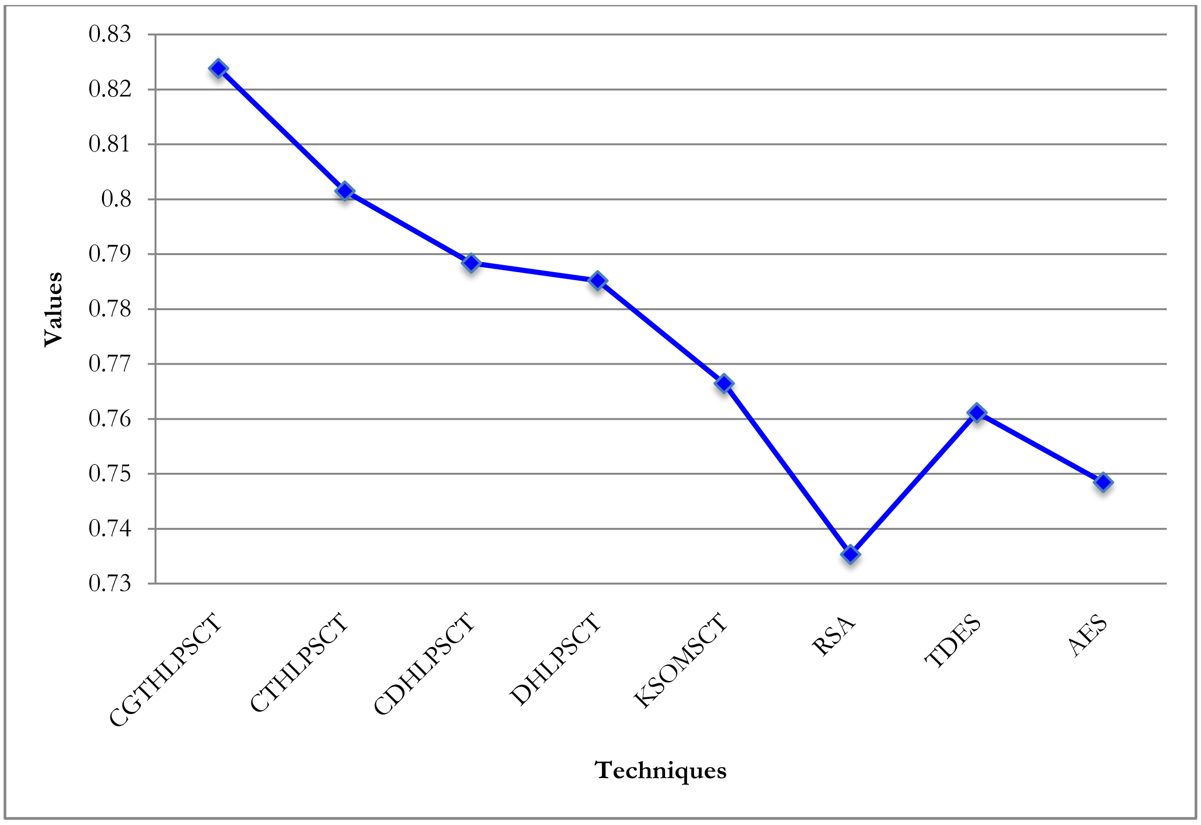


Figure 7.66: Pictorial representation of the average values of Bit Independence of *.doc* type bit stream

7.6 Test for Non-Homogeneity

Chi-Square value is calculated from the character frequencies using the formula devised by Karl Pearson which is called "Pearsonian Chi-Square". The higher the Chi-Square values the more deviation from the original message. Section 7.6.1 contains the results of *.dll* files and section 7.6.2 deals with *.exe* files. Section 7.6.3 and 7.6.4 deal with the results of *.txt* and *.doc* files respectively.

7.6.1 *.dll* files

Twenty *.dll* files of different sizes varying from 3216 bytes to 5,456,704 bytes have been taken to measure the Chi-Square values for different techniques. Table 7.83 shows the Chi-Square values obtained using CGTHLPSCT, CTHLPSCT, CDHLPSCT, DHLPSCT, KSOMSCT, TDES, and AES of *.dll* type files. The average Chi-Square values obtained using proposed CGTHLPSCT, CTHLPSCT, CDHLPSCT, DHLPSCT, KSOMSCT, TDES, and AES are 66934661, 6975581, 7046365, 7091691, 7118000, 7133646, and 6804334 respectively. Chi-Square values increase with the increase of source file sizes.

Figure 7.67 shows the comparison of the average Chi-Square values of *.dll* type of source files for proposed CGTHLPSCT, CTHLPSCT, CDHLPSCT, DHLPSCT, KSOMSCT, TDES, and AES. For all proposed techniques, the Chi-Square values of the encrypted files are very high. So, it may obtain better degree of security in proposed which is comparable with that of others.

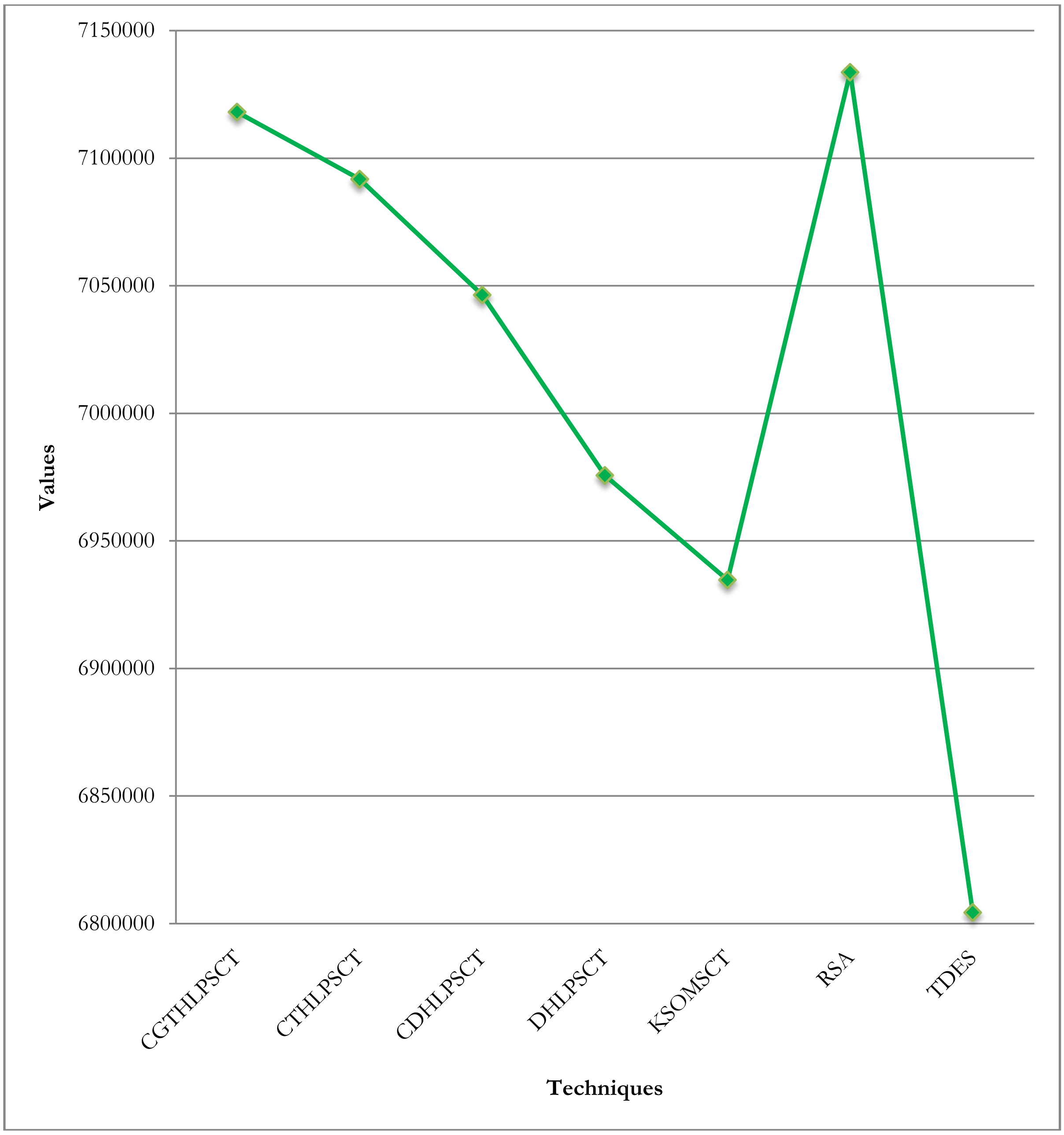


Figure 7.67: Pictorial representation of the average values of Chi-Square of *.dll* type bit stream

Table: 7.82
Comparisons of Chi-Square value of *.dll* files

Serial no.	Source File name	Source file size (In bytes)	Chi-Square values						
			CGTHLPSCT	**CTHLPSCT**	**CDHLPSCT**	**DHLPSCT**	**KSOMSCT**	**TDES**	**AES**
1	a01.dll	3216	36231	34083	33739	34094	34933	36054	26036
2	a02.dll	6,656	194393	185297	128393	126340	108674	193318	118331
3	a03.dll	12,288	169427	194032	147306	145543	137834	165053	81475
4	a04.dll	24,576	8260935	8260321	7249089	7129562	6147653	8310677	4794027
5	a05.dll	58,784	825397	809425	395936	381295	321983	806803	466466
6	a06.dll	85,020	664189	659304	481904	468943	449272	654756	433872
7	a07.dll	169,472	1482375	1462812	996952	971906	863406	1473410	1601070
8	a08.dll	359,936	1122096	899053	722904	735237	731276	423984	398685
9	a09.dll	593,920	1362671	1316539	1277829	1259042	1198749	1367968	1277751
10	a10.dll	909,312	2370478	2298417	2040637	1918420	1680956	2377544	2275676
11	a11.dll	1,293,824	2422941	2063494	1858428	1832907	1633962	1065999	948834
12	a12.dll	1,925,185	45910637	43290342	38922982	41264893	45172384	46245126	47627346
13	a13.dll	2,498,560	5182562	5098285	4780274	4712984	4887347	4616320	4625829
14	a14.dll	3,485,968	14220937	13884306	12031847	11939433	11590534	14567497	13560121
15	a15.dll	3,790,336	7001874	8830287	8807946	8783748	8942907	7110339	7051889
16	a16.dll	4,253,816	7971093	8472804	10952471	10321117	9215648	8451794	8194777
17	a17.dll	4,575,232	7850935	8651904	9464528	9393217	8914895	8632408	8649446
18	a18.dll	4,883,456	8736217	8910462	10963053	10872319	9912906	8866085	8450004
19	a19.dll	5,054,464	14629092	14740371	15920532	14556342	14109345	14875409	13265423
20	a20.dll	5,456,704	11981763	11806373	13784296	12498389	12673493	12432371	12239623
Average			7118000	7091691	7046365	6975581	6934661	7133646	6804334

7.5.2 *.exe* files

Twenty *.exe* files of different sizes varying from 1063 bytes to 6,735,934 bytes have been taken to measure the Chi-Square values for different techniques. Table 7.84 shows the Chi-Square values obtained using CGTHLPSCT, CTHLPSCT, CDHLPSCT, DHLPSCT, KSOMSCT, TDES, and AES of *.exe* type files. The average Chi-Square values obtained using proposed CGTHLPSCT, CTHLPSCT, CDHLPSCT, DHLPSCT, KSOMSCT, TDES, and AES are 4104189, 4349141, 4574169, 4757447, 4834975, 6169940, and 4096723 respectively. Chi-Square values increase with the increase of source file sizes.

Figure 7.68 shows the comparison of the average Chi-Square values of *.exe* type of source files for proposed CGTHLPSCT, CTHLPSCT, CDHLPSCT, DHLPSCT, KSOMSCT, TDES, and AES. For all proposed techniques, the Chi-Square values of the encrypted files are very high. So, it may obtain better degree of security in proposed which is comparable with that of others.

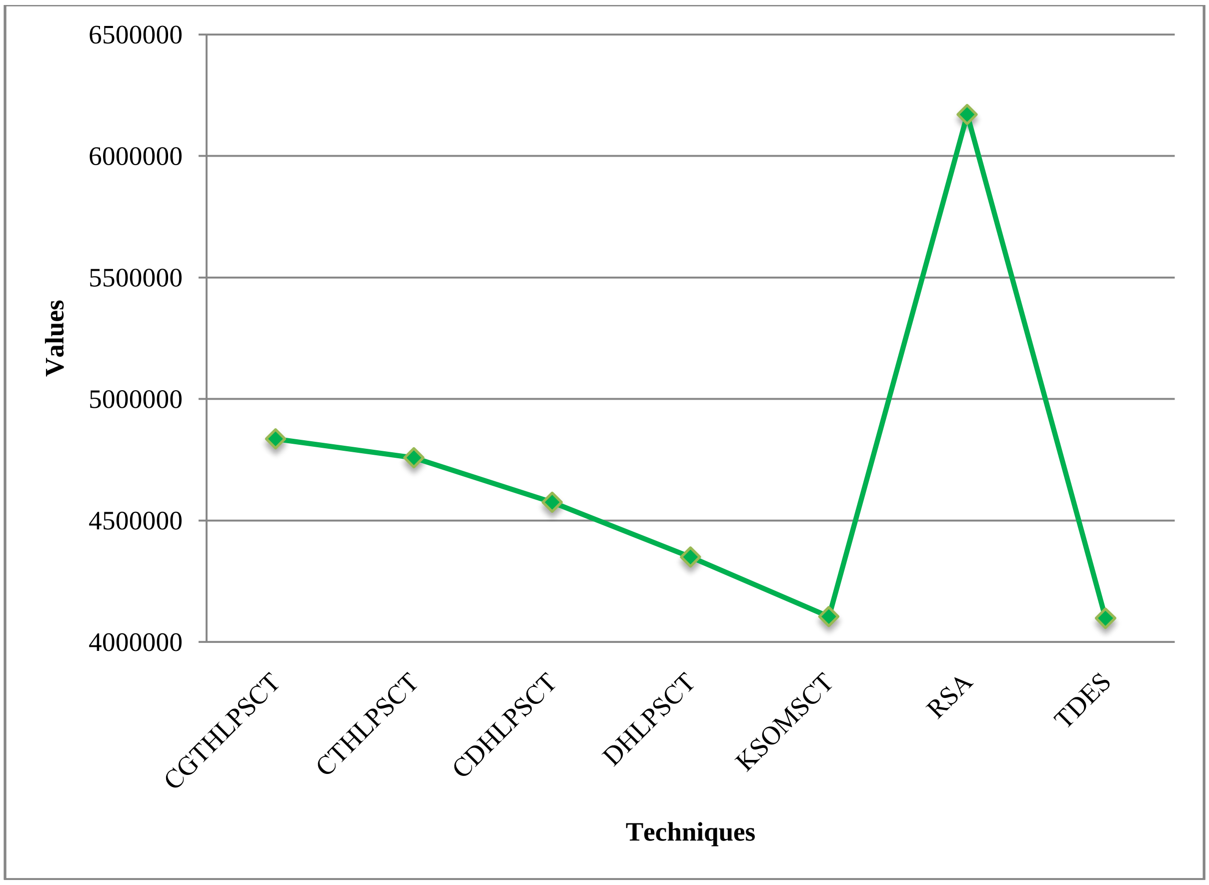


Figure 7.68: Pictorial representation of the average values of Chi-Square of *.exe* type bit stream

Table: 7.83

Comparisons of Chi-Square value of *.exe* files

Serial no.	Source File name	Source file size (In bytes)	Chi-Square values						
			CGTHLPSCT	**CTHLPSCT**	**CDHLPSCT**	**DHLPSCT**	**KSOMSCT**	**TDES**	**AES**
1	a01. exe	1,063	12873	11985	13806	12980	14172	31047	15349
2	a02. exe	2,518	69834	67093	79938	76321	89946	167604	58911
3	a03. exe	8,250	85297	84521	84902	84120	86091	3171258	1193952
4	a04. exe	15,937	118427	115239	108356	103895	99387	137421	90439
5	a05. exe	22,874	28783	27905	26894	26498	25985	40605	42948
6	a06. exe	35,106	278934	273973	272095	269874	257394	751034	996561
7	a07. exe	52,032	115342	114783	111875	110845	108746	252246	227972
8	a08. exe	145,387	645094	641097	637894	630955	622467	1619619	879622
9	a09. exe	248,273	643902	641117	636949	629864	618647	1188392	1206461
10	a10. exe	478,321	837231	825287	821674	801097	796454	1646895	1611814
11	a11. exe	738,275	1829055	1715636	1654098	1589549	1557482	1953381	1955305
12	a12. exe	1,594,276	2345287	2330865	2319485	228948	221837	3388013	3349821
13	a13. exe	2,273,670	4039075	4029864	4009856	3975635	3876748	5386323	5358508
14	a14. exe	2,985,306	3872393	3751834	3567849	3517843	3378487	4435189	4391280
15	a15. exe	3,412,639	197532	1933896	1897485	1820946	1765849	312451	304503
16	a16. exe	3,872,984	2540637	2519086	2487650	2285773	2018478	2859239	2529935
17	a17. exe	4,038,387	6015	5986	5521	5129	4786	8783	9015
18	a18. exe	5,284,796	5389043	536027	5343398	5276749	4987584	6874552	6590217
19	a19. exe	5,628,037	19854209	18837648	16654901	15749327	14894756	22431762	16734368
20	a20. exe	6,735,934	53790547	51852098	50748754	49786481	46658494	66742981	34387484
Average			4834975	4757447	4574169	4349141	4104189	6169940	4096723

7.5.3 .*txt* files

Twenty .*txt* files of different sizes varying from 1504 bytes to 6,702,831 bytes have been taken to measure the Chi-Square values for different techniques. Table 7.85 shows the Chi-Square values obtained using CGTHLPSCT, CTHLPSCT, CDHLPSCT, DHLPSCT, KSOMSCT, TDES, and AES of .*txt* type files. The average Chi-Square values obtained using proposed CGTHLPSCT, CTHLPSCT, CDHLPSCT, DHLPSCT, KSOMSCT, TDES, and AES are 476002855099, 457088752936, 41143854777, 36900009771, 30722317122, 28557702243, and 25826336277 respectively. Chi-Square values increase with the increase of source file sizes.

Figure 7.69 shows the comparison of the average Chi-Square values of .*txt* type of source files for proposed CGTHLPSCT, CTHLPSCT, CDHLPSCT, DHLPSCT, KSOMSCT, TDES, and AES. For all proposed techniques, the Chi-Square values of the encrypted files are very high. So, it may obtain better degree of security in proposed which is comparable with that of others.

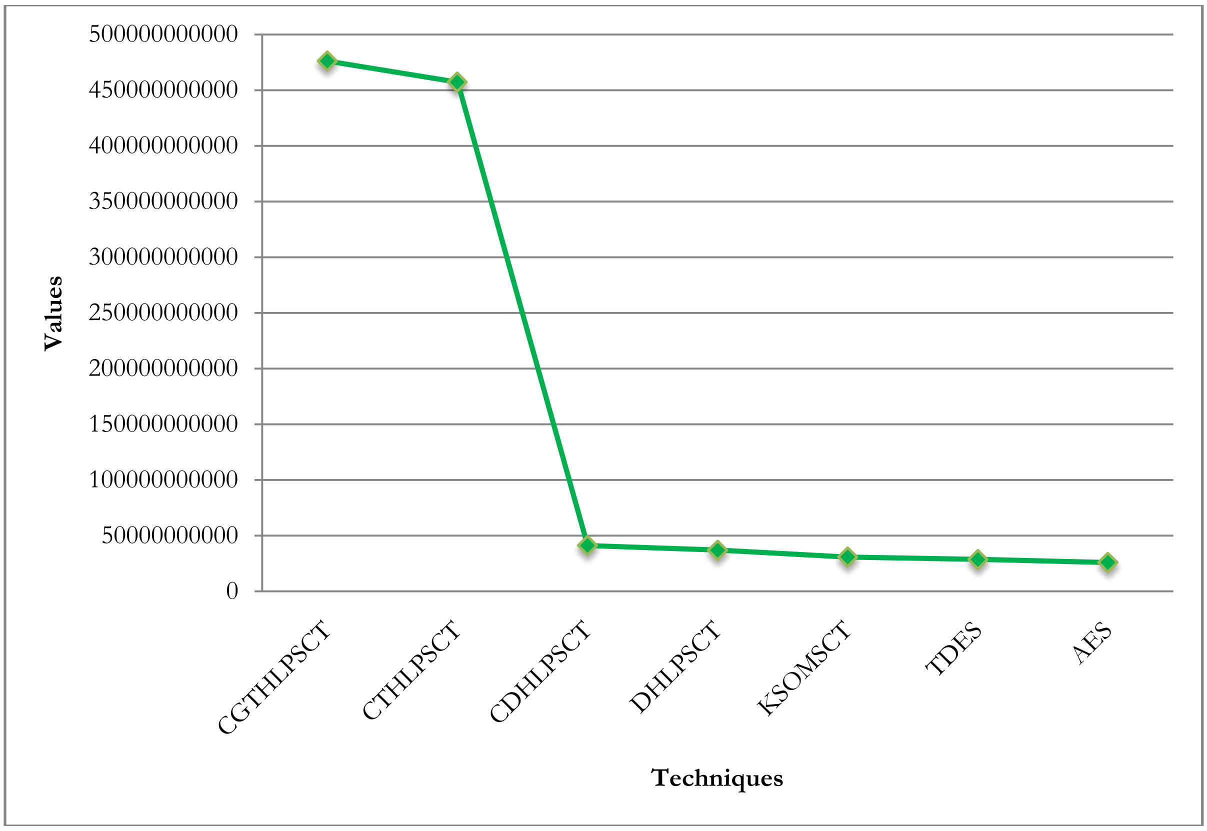


Figure 7.69: Pictorial representation of the average values of Chi-Square of .*txt* type bit stream

Table: 7.84
Comparisons of Chi-Square value of *.txt* files

Serial no.	Source File name	Source file size (In bytes)	Chi-Square values						
			CGTHLPSCT	**CTHLPSCT**	**CDHLPSCT**	**DHLPSCT**	**KSOMSCT**	**TDES**	**AES**
1	a01.txt	1,504	3928464	386473	37485	37198	31938	58385	15267
2	a02.txt	7,921	13289452	12258309	785634	653812	587129	1500874	347663
3	a03.txt	17,036	91542980	86129865	4823409	4425909	3809645	7721661	1731310
4	a04.txt	44.624	48715409	46919023	7239064	5956321	4923806	4709724	4753971
5	a05.txt	68,823	398287135	389278954	27187564	22129876	18569064	29704639	15663977
6	a06.txt	161,935	481597136	457129845	92790569	76438907	68865906	76621083	64043270
7	a07.txt	328,017	3741632472	3561209673	423376129	353890745	328096745	388539921	325837900
8	a08.txt	587,290	16129846761	15990965340	1738797676	1549867335	1364538932	1258362670	1082315460
9	a09.txt	1,049,763	58798471525	55312986743	4849846875	3485690453	2965423187	3264221211	2585100024
10	a10.txt	1,418,025	53821651743	51287369561	8195645098	7549087564	8237908765	5896971610	5524089746
11	a11.txt	1,681,329	173290541286	165532816732	14145390986	12198087654	7941894390	9087072783	8355902146
12	a12.txt	2,059,318	184975984675	181632907453	18967452398	18767453098	12967095437	11627270156	11387797334
13	a13.txt	2,618,492	321085924331	305095623874	31912985534	29543098734	25839096738	24260650965	21978420834
14	a14.txt	3,154,937	518190376344	487589453287	40756340987	34529187230	28634908759	32021906499	29332709650
15	a15.txt	4,073,829	642907845126	619432716093	75327909654	49756230987	53867340987	47346524666	44660520923
16	a16.txt	4,936,521	948292764102	926431276356	81470983456	56867215490	52412907645	57698683717	49783638147
17	a17.txt	5,125,847	1311762068483	1255439061805	141907654387	116340982395	56164389079	72922461490	64846889153
18	a18.txt	5,593,219	1572893597324	1496745328775	125984609879	96490863457	88954120953	81707468147	77543081318
19	a19.txt	5,898,302	1753290563853	1695790756468	128653909645	175563409174	126390854880	101228345379	89456481325
20	a20.txt	6,702,831	1959838473374	1880940484091	148409329116	134895489087	148280978453	122325249286	109577386113
Average			476002855099	457088752936	41143854777	36900009771	30722317122	28557702243	25826336277

7.6.4 *.doc* files

Twenty *.doc* files of different sizes varying from 21,052 bytes to 5,472,298 bytes have been taken to measure the Chi-Square values for different techniques. Table 7.86 shows the Chi-Square values obtained using CGTHLPSCT, CTHLPSCT, CDHLPSCT, DHLPSCT, KSOMSCT, TDES, and AES of *.doc* type files. The average Chi-Square values obtained using proposed CGTHLPSCT, CTHLPSCT, CDHLPSCT, DHLPSCT, KSOMSCT, TDES, and AES are 6713314, 6858556, 6941065, 6976655, 7125858, 11021752, and 6763362 respectively. Chi-Square values increase with the increase of source file sizes.

Figure 7.70 shows the comparison of the average Chi-Square values of *.doc* type of source files for proposed CGTHLPSCT, CTHLPSCT, CDHLPSCT, DHLPSCT, KSOMSCT, TDES, and AES. For all proposed techniques, the Chi-Square values of the encrypted files are very high. So, it may obtain better degree of security in proposed which is comparable with that of others.

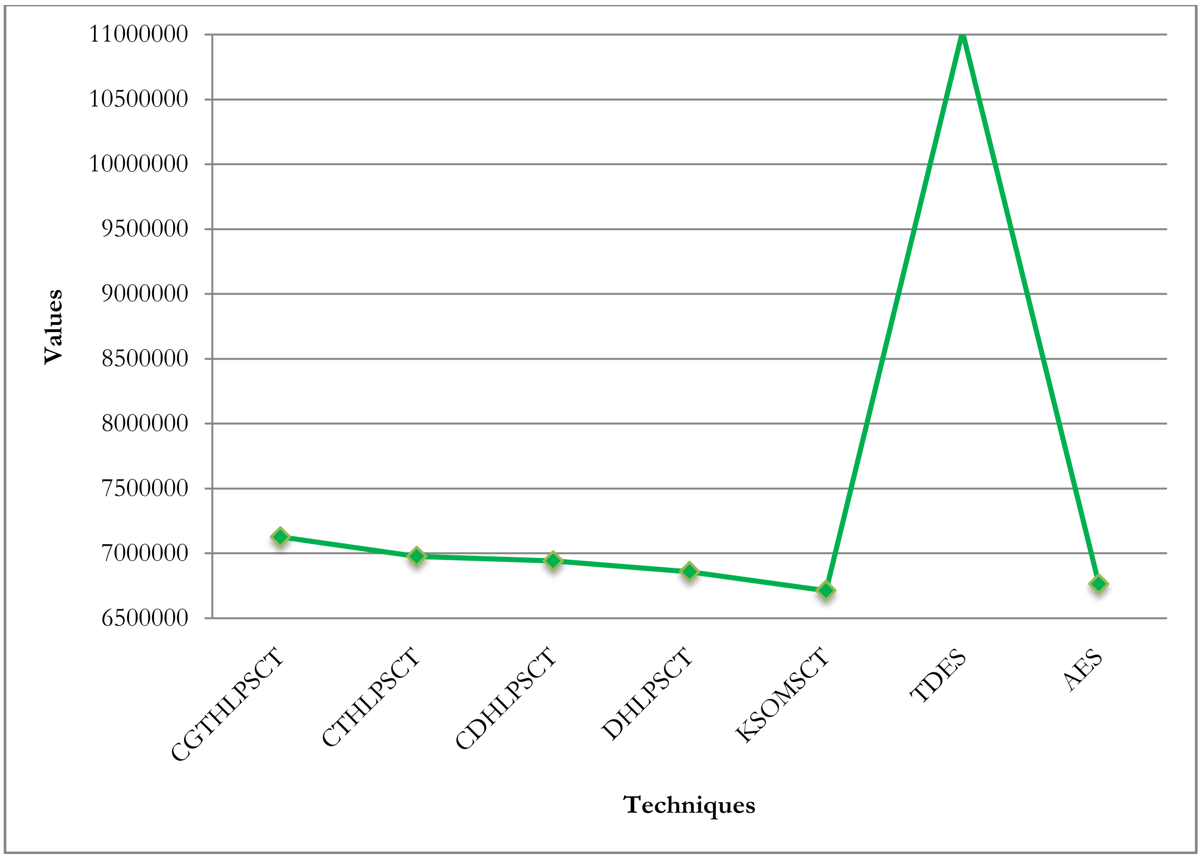


Figure 7.70: Pictorial representation of the average values of Chi-Square of *.doc* type bit stream

Table: 7.85
Comparisons of Chi-Square value of *.doc* files

Serial no.	Source File name	Source file size (In bytes)	Chi-Square values						
			CGTHLPSCT	**CTHLPSCT**	**CDHLPSCT**	**DHLPSCT**	**KSOMSCT**	**TDES**	**AES**
1	a01. doc	21,052	15197	14893	6210846	6109459	6584038	18918008	14182910
2	a02. doc	33,897	45015	43903	1298305	2540299	2890458	9503676	4431277
3	a03. doc	45,738	117328	115845	1467406	1904827	1832039	9361015	2145383
4	a04. doc	75,093	241578	237094	1760847	1799038	1719053	2848468	1347091
5	a05. doc	106,872	520967	507195	1909562	1870653	1794092	3933039	1898438
6	a06. doc	327.054	309610	297127	523096	590956	580984	537285	373599
7	a07. doc	582,831	961230	946538	920982	886429	863092	1349490	947148
8	a08. doc	729,916	6095382	5919048	5829064	5639042	5509832	5474962	4532789
9	a09. doc	1,170,251	2689163	2560942	2426703	2190563	2090482	4598604	3097778
10	a10. doc	1,749,272	26619328	26192730	25795683	25137093	24709385	41385774	27850217
11	a11. doc	2,045,805	22118906	21295672	19504987	19134097	18630942	23692555	11574426
12	a12. doc	2,372,014	15219859	14909583	14290539	14178095	13790434	18656807	11848004
13	a13. doc	2,869,275	8719432	8569396	6109565	6023896	5729084	17460853	8762683
14	a14. doc	3,161,353	10439826	10263935	10267429	10198431	9987353	9904389	6784251
15	a15. doc	3,570,295	9513042	9245042	8534074	8129054	7940973	11123554	6844351
16	a16. doc	3,834,427	8710934	8598424	7556031	7230986	6990386	7725687	5230567
17	a17. doc	4,011,986	8512943	8437261	7031864	6939093	6759037	9846653	6437662
18	a18. doc	4,562,385	7904577	7838924	6539063	6287235	6073904	7376693	5591776
19	a19. doc	4,839,102	6310939	6210498	4804169	4750934	4580856	8592223	5374094
20	a20.doc	5,472,298	7451905	7329048	6041093	5630939	5209857	8145414	6012872
Average			7125858	6976655	6941065	6858556	6713314	11021752	6763362

7.7 Analysis of Character Frequencies, Entropy, Floating Frequencies, Autocorrelation

Program access both the original and encrypted files and stores the occurrence of each character in an array. The final output is an excel file to facilitate generation of graph. The smoother or less curves in the spectrum of frequency distribution indicate that it is harder for a cryptanalyst to detect the original text bytes which implies better degree of security. Entropy near to 1 indicates the good encryption technique. Well distributed floating frequencies are indicate the robustness of the encryption and finally Autocorrelation indicates goodness of the technique. Section 7.7.1 deals with analysis of KSOMSCT encrypted *.dll* files. Section 7.7.2 presented the analysis of DHLPSCT encrypted *.com* files. Analysis of CDHLPSCT encrypted *.exe* files has been presented in section 7.7.3. Section 7.7.4 deals with analysis of CTHLPSCT encrypted *.cpp* files. Finally, analysis of CGTHLPSCT encrypted *.txt* files has been presented in section 7.7.5.

7.7.1 *.dll* file

Analysis of character frequencies of twenty source files of *.dll* type has been performed using KSOMSCT. Figure 7.71 shows the spectrum of frequency distribution of characters for the input source stream. Figure 7.72 shows the spectrum of frequency distribution of encrypted characters using KSOMSCT for the same input source stream. It has been observed that frequencies of characters are widely distributed in KSOMSCT encrypted *.dll* file.

Analysis of entropy of twenty source files of *.dll* type has been performed using KSOMSCT. The entropy of a source thus indicates its characteristic distribution. It measures the average amount of information which one can obtain through observation of the source or, conversely, the indeterminacy which prevails over the generated messages when one cannot observe the source. The entropy for the input source stream is 4.88. Whereas the entropy of encrypted characters using KSOMSCT for the same input source stream is 7.99. From the figures it is observed that entropy of KSOMSCT encrypted characters is near to eight which indicate the high degree of security.

Analysis of floating frequencies of twenty source files of *.dll* type has been performed using KSOMSCT. The floating frequency of a document is a characteristic of its local information content at individual points in the document. The floating frequency specifies how many different characters are to be found in any given 64-character long segment of the document. Figure 7.73 shows the spectrum of floating frequencies of characters for the input source stream. Figure 7.74 shows the spectrum of floating frequencies of encrypted characters using KSOMSCT for the same input source stream. From the figures it is observed that floating frequencies of KSOMSCT encrypted characters indicates the high degree of security.

Analysis of autocorrelation of twenty source files of *.dll* type has been performed using KSOMSCT. The autocorrelation of a document is an index of the similarity of different sections of the document. Figure 7.75 shows the spectrum of autocorrelation of characters for the input source stream. Figure 7.76 shows the spectrum of autocorrelation of encrypted characters using KSOMSCT for the same input source stream. From the figure it is observed that autocorrelation of KSOMSCT encrypted characters indicate the high degree of security.

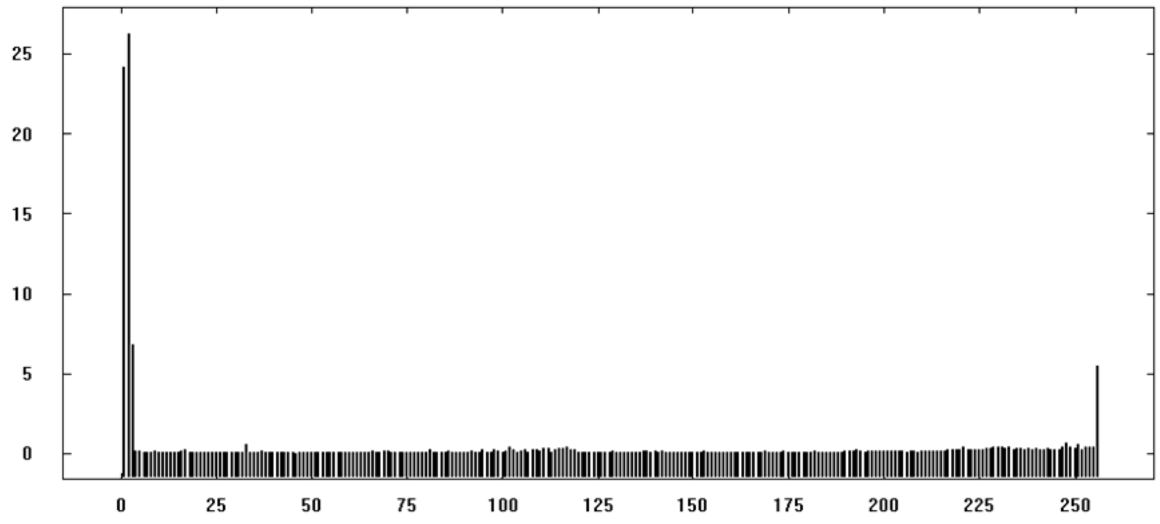


Figure 7.71: Graphical representation of frequency distribution spectrum of characters for the *.dll* type input source stream

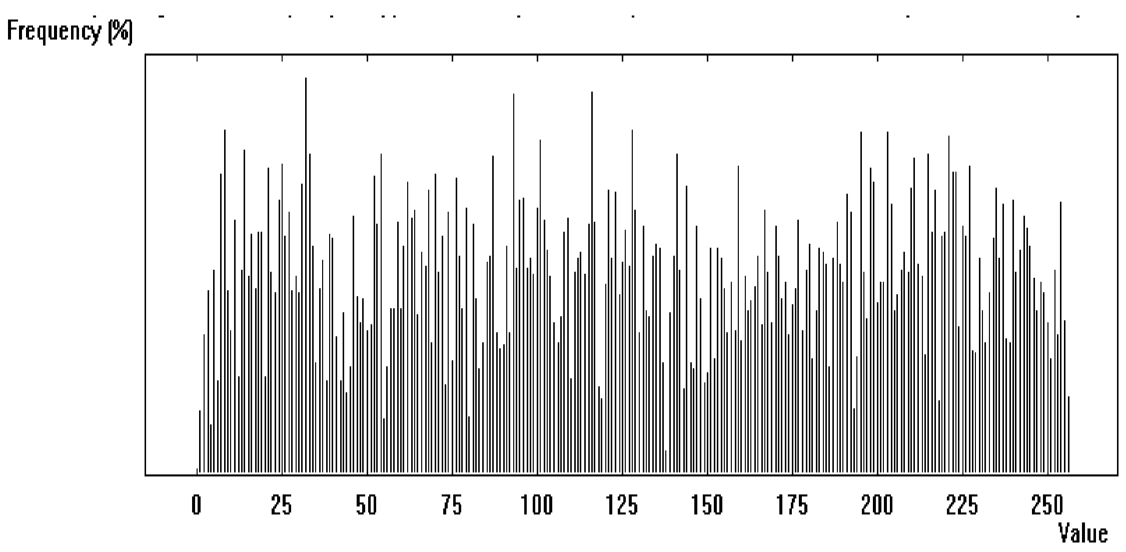


Figure 7.72: Graphical representation of frequency distribution spectrum of characters for the encrypted stream using KSOMSCT for *.dll* file

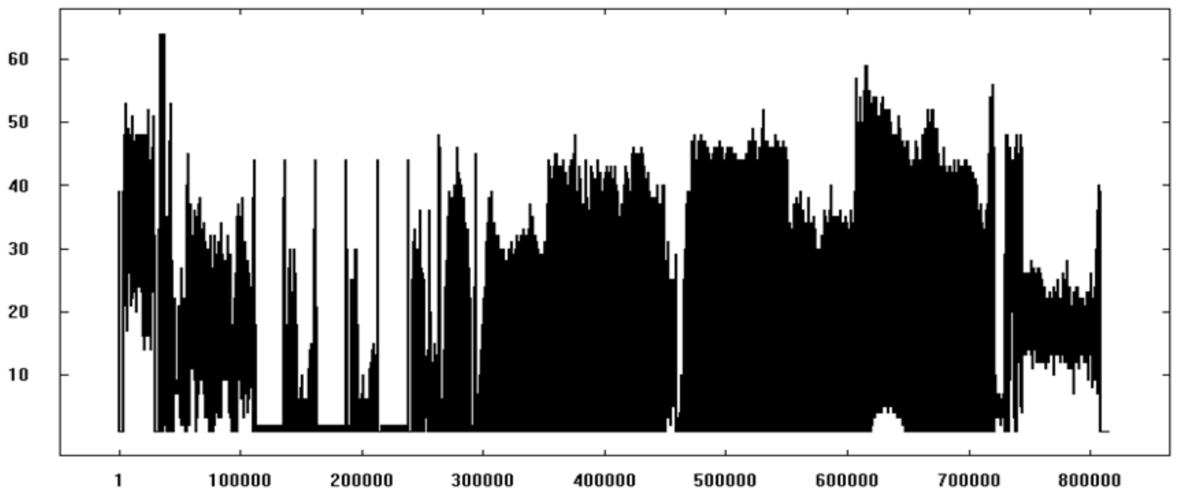


Figure 7.73: Floating frequency of the input *.dll* source stream

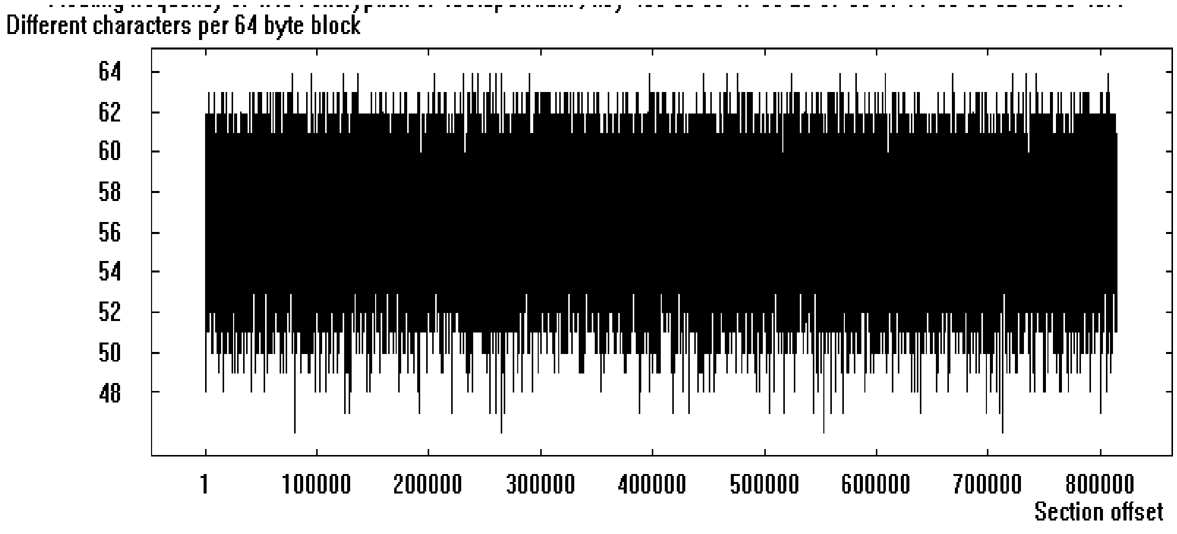


Figure 7.74: Floating frequency of the encrypted stream using KSOMSCT for *.dll* file

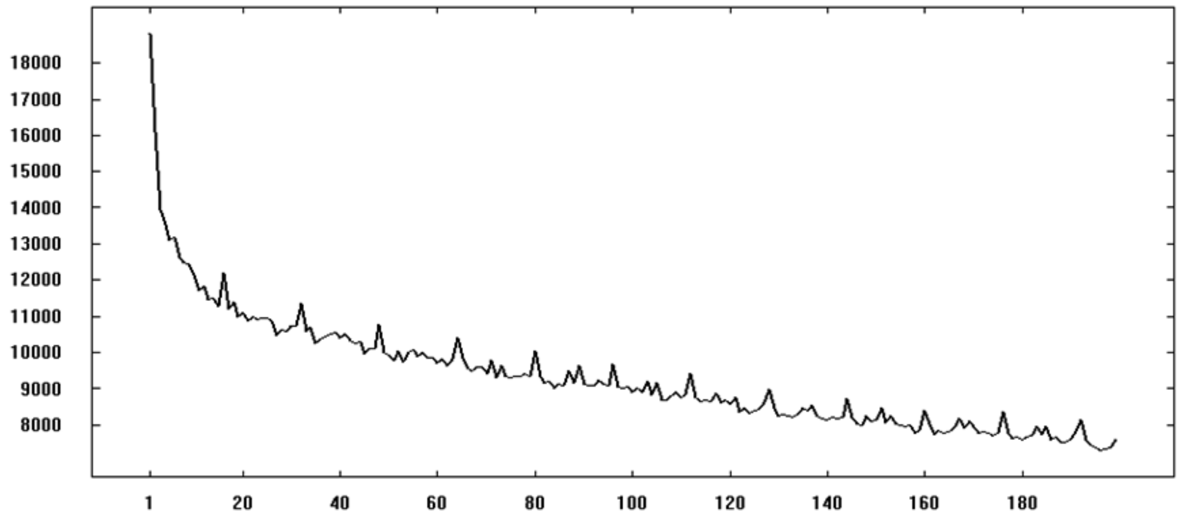


Figure 7.75: Autocorrelation of the input *.dll* source stream

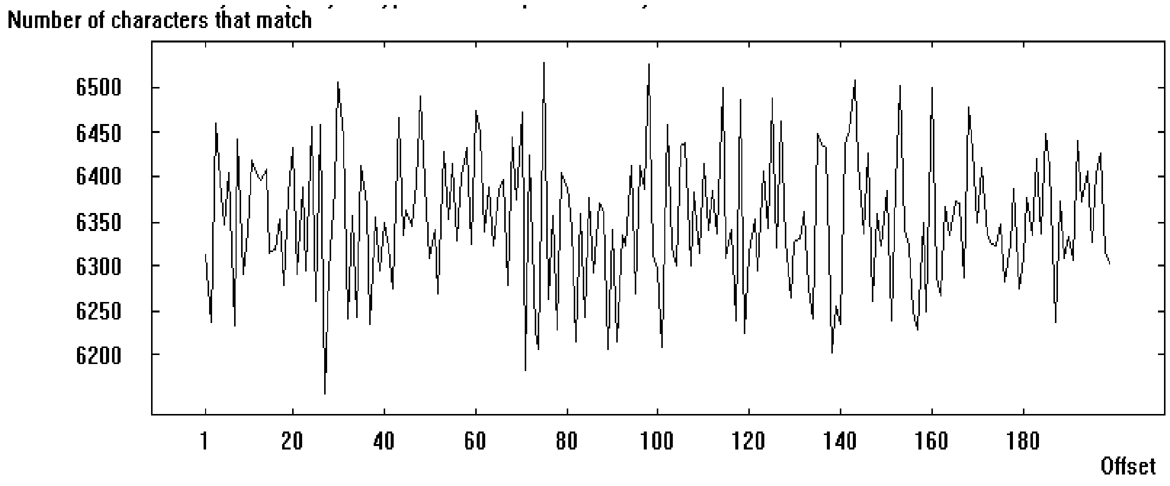


Figure 7.76: Autocorrelation of the encrypted stream using KSOMSCT for *.dll* file

7.7.2 *.com* file

Analysis of character frequencies of twenty source files of *.com* type has been performed using DHLPSCT. Figure 7.77 shows the spectrum of frequency distribution of characters for the input source stream. Figure 7.78 shows the spectrum of frequency distribution of encrypted characters using DHLPSCT for the same input source stream. It has been observed that frequencies of characters are widely distributed in DHLPSCT encrypted *.com* file.

Analysis of entropy of twenty source files of *.com* type has been performed using DHLPSCT. The entropy of a source thus indicates its characteristic distribution. It measures the average amount of information which one can obtain through observation of the source or, conversely, the indeterminacy which prevails over the generated messages when one cannot observe the source. The entropy for the input source stream is 4.04. Whereas the entropy of encrypted characters using DHLPSCT for the same input source stream is 7.99. From the figures it is observed that entropy of DHLPSCT encrypted characters is near to eight which indicate the high degree of security.

Analysis of floating frequencies of twenty source files of *.com* type has been performed using DHLPSCT. The floating frequency of a document is a characteristic of its local information content at individual points in the document. The floating frequency specifies how many different characters are to be found in any given 64-character long segment of the document. Figure 7.79 shows the spectrum of floating frequencies of characters for the input source stream. Figure 7.80 shows the spectrum of floating frequencies of encrypted characters using DHLPSCT for the same input source stream. From the figures it is observed that floating frequencies of DHLPSCT encrypted characters indicates the high degree of security.

Analysis of autocorrelation of twenty source files of *.com* type has been performed using DHLPSCT. The autocorrelation of a document is an index of the similarity of different sections of the document. Figure 7.81 shows the spectrum of autocorrelation of characters for the input source stream. Figure 7.82 shows the spectrum of autocorrelation of encrypted characters using DHLPSCT for the same input source stream. From the figure it is observed that autocorrelation of DHLPSCT encrypted characters indicate the high degree of security.

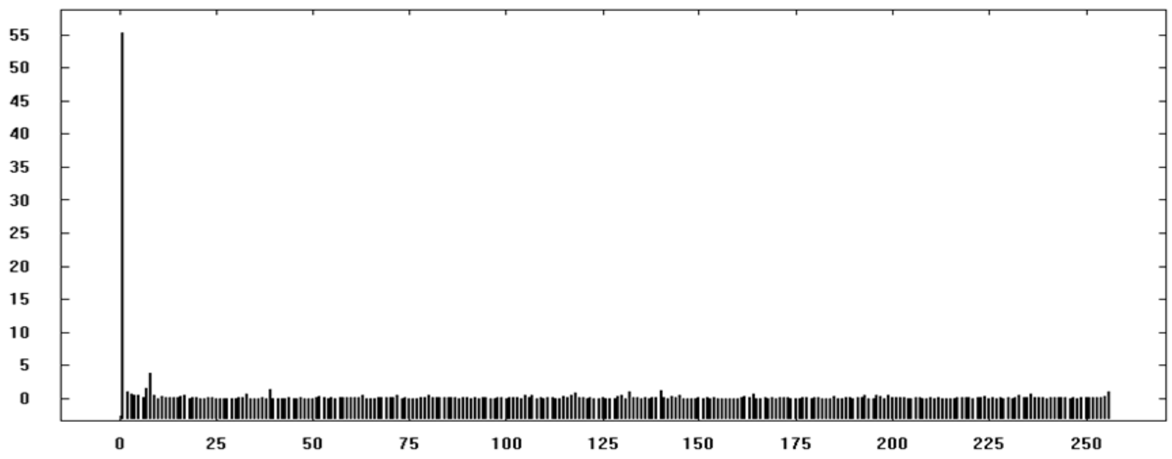


Figure 7.77: Graphical representation of frequency distribution spectrum of characters for the input *.com* source stream

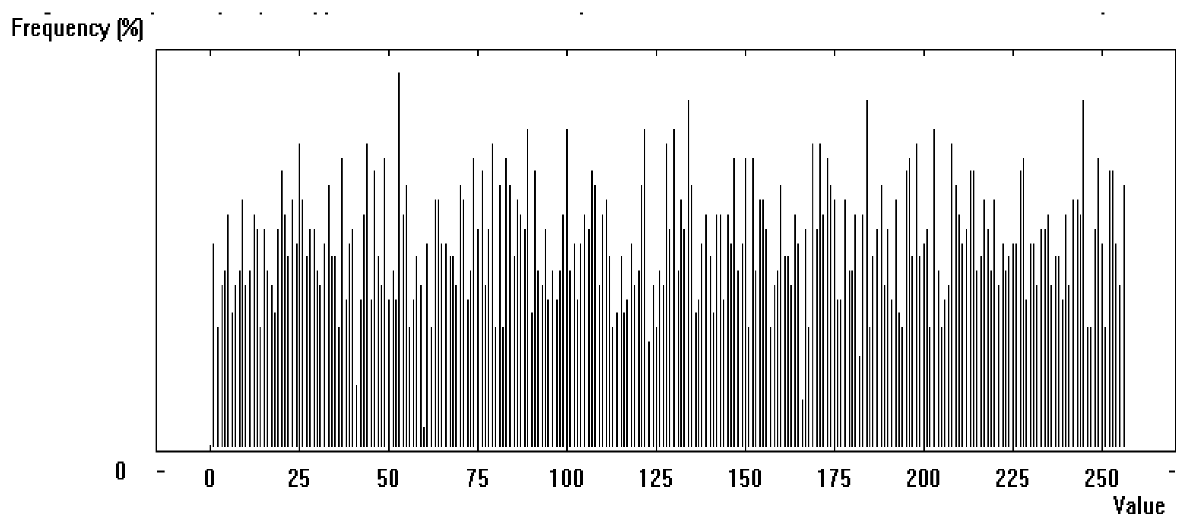


Figure 7.78: Graphical representation of frequency distribution spectrum of characters for the encrypted stream using DHLPSCT for *.com* file

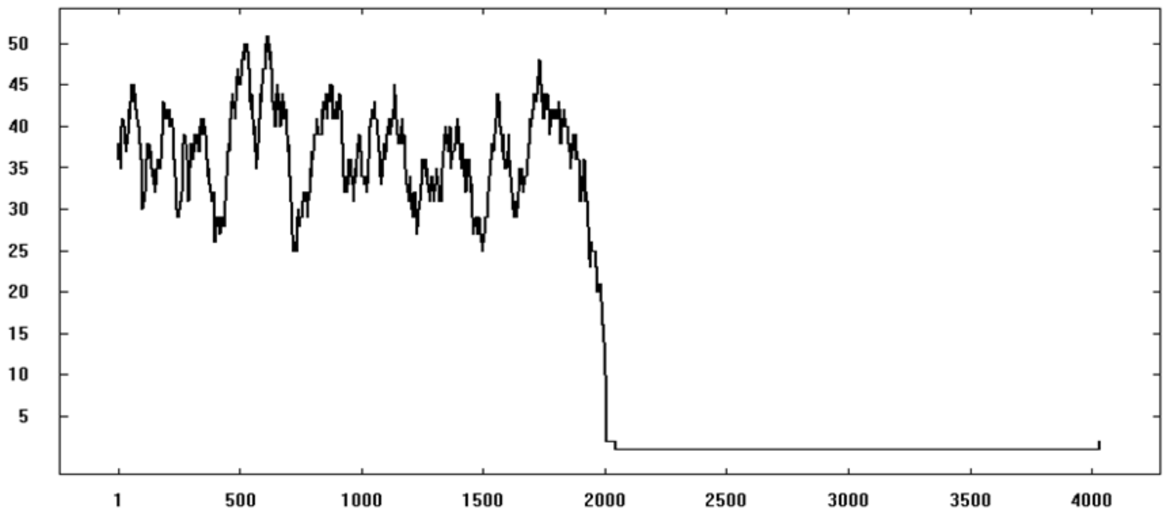


Figure 7.79: Floating frequency of the input *.com* source stream

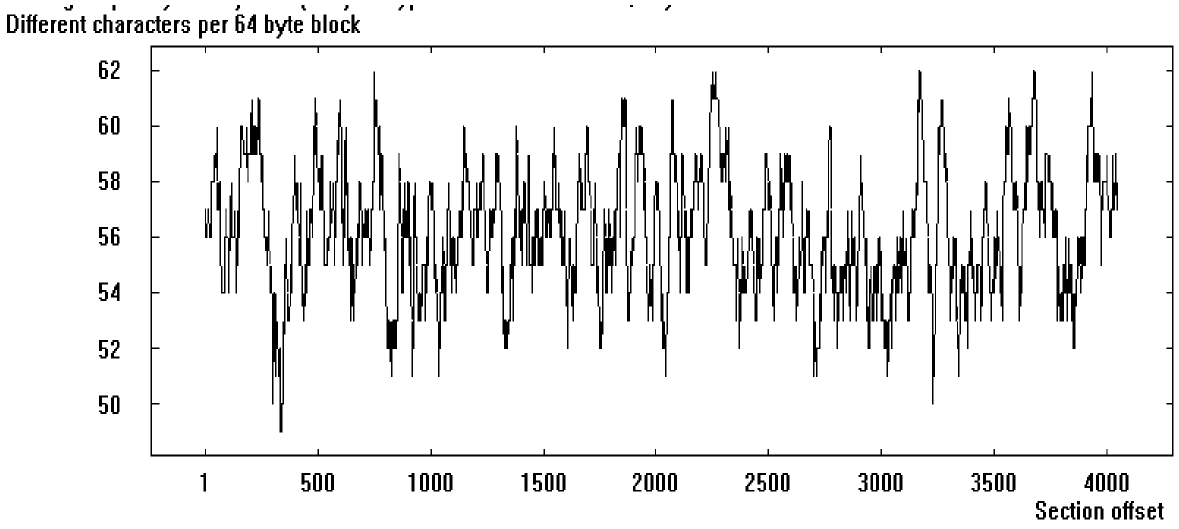


Figure 7.80: Floating frequency of the encrypted stream using DHLPSCT for *.com* file

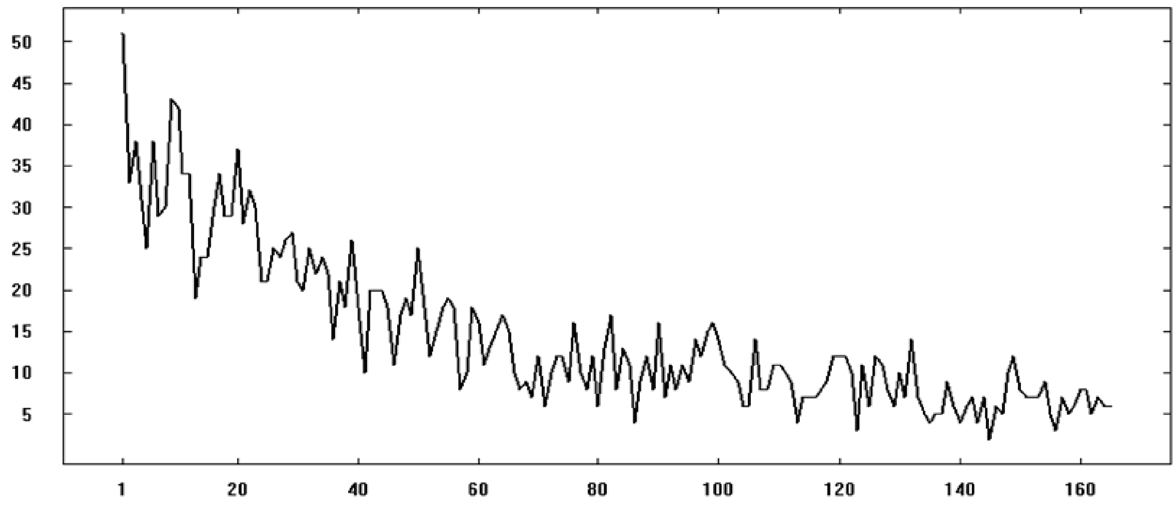


Figure 7.81: Autocorrelation of the input *.com* source stream

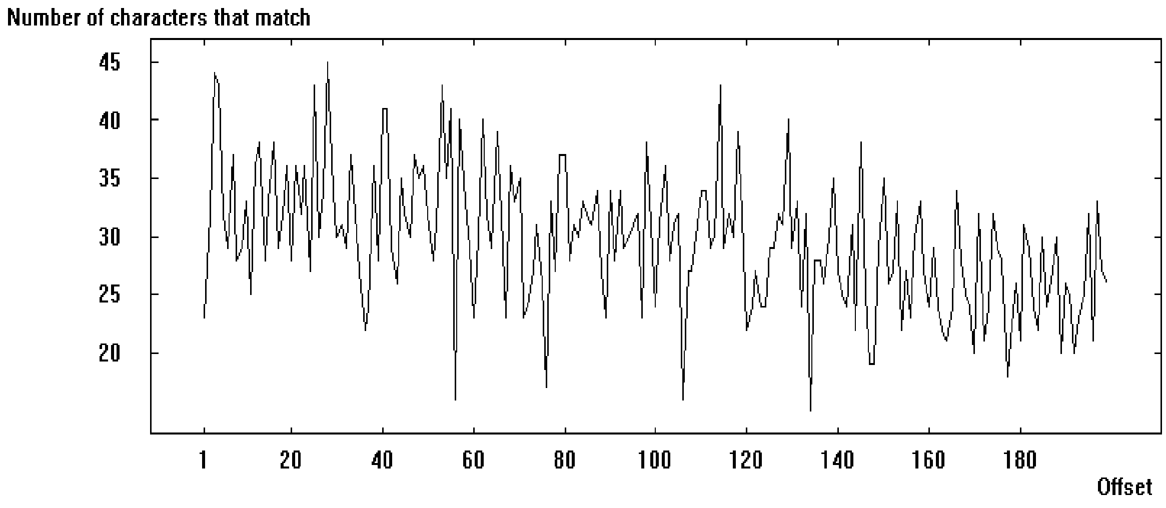


Figure 7.82: Autocorrelation of the encrypted stream using DHLPSCT for *.com* file

7.7.3 *.exe* file

Analysis of character frequencies of twenty source files of *.exe* type has been performed using CDHLPSCT. Figure 7.83 shows the spectrum of frequency distribution of characters for the input source stream. Figure 7.84 shows the spectrum of frequency distribution of encrypted characters using CDHLPSCT for the same input source stream. It has been observed that frequencies of characters are widely distributed in CDHLPSCT encrypted *.exe* file.

Analysis of entropy of twenty source files of *.exe* type has been performed using CDHLPSCT. The entropy of a source thus indicates its characteristic distribution. It measures the average amount of information which one can obtain through observation of the source or, conversely, the indeterminacy which prevails over the generated messages when one cannot observe the source. The entropy for the input source stream is 7.85. Whereas the entropy of encrypted characters using CDHLPSCT for the same input source stream is 7.99. From the figures it is observed that entropy of CDHLPSCT encrypted characters is near to eight which indicate the high degree of security.

Analysis of floating frequencies of twenty source files of *.exe* type has been performed using CDHLPSCT. The floating frequency of a document is a characteristic of its local information content at individual points in the document. The floating frequency specifies how many different characters are to be found in any given 64-character long segment of the document. Figure 7.85 shows the spectrum of floating frequencies of characters for the input source stream. Figure 7.86 shows the spectrum of floating frequencies of encrypted characters using CDHLPSCT for the same input source stream. From the figures it is observed that floating frequencies of CDHLPSCT encrypted characters indicates the high degree of security.

Analysis of autocorrelation of twenty source files of *.exe* type has been performed using CDHLPSCT. The autocorrelation of a document is an index of the similarity of different sections of the document. Figure 7.87 shows the spectrum of autocorrelation of characters for the input source stream. Figure 7.88 shows the spectrum of autocorrelation of encrypted characters using CDHLPSCT for the same input source stream. From the figure it is observed that autocorrelation of CDHLPSCT encrypted characters indicate the high degree of security.

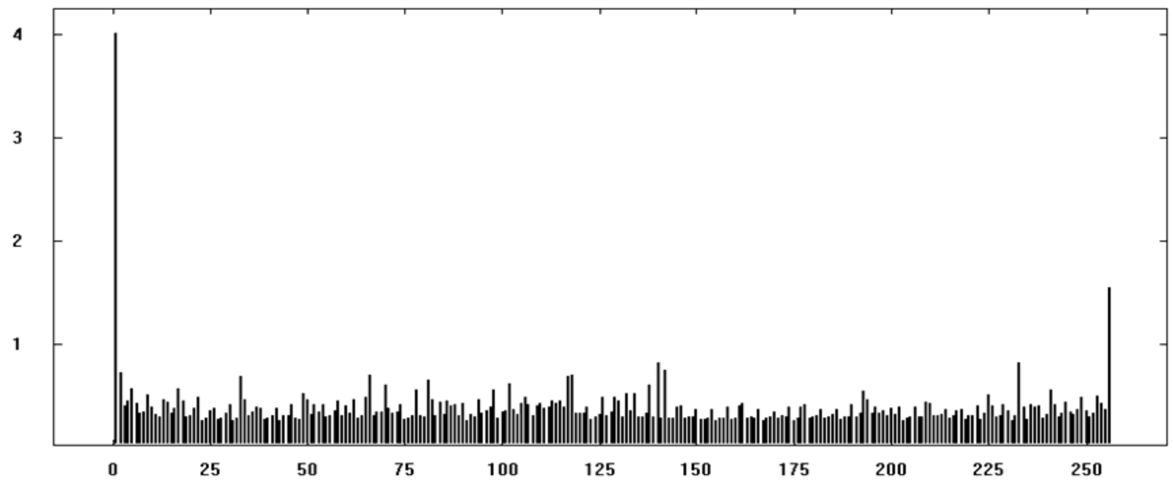


Figure 7.83: Graphical representation of frequency distribution spectrum of characters for the input *.exe* source stream

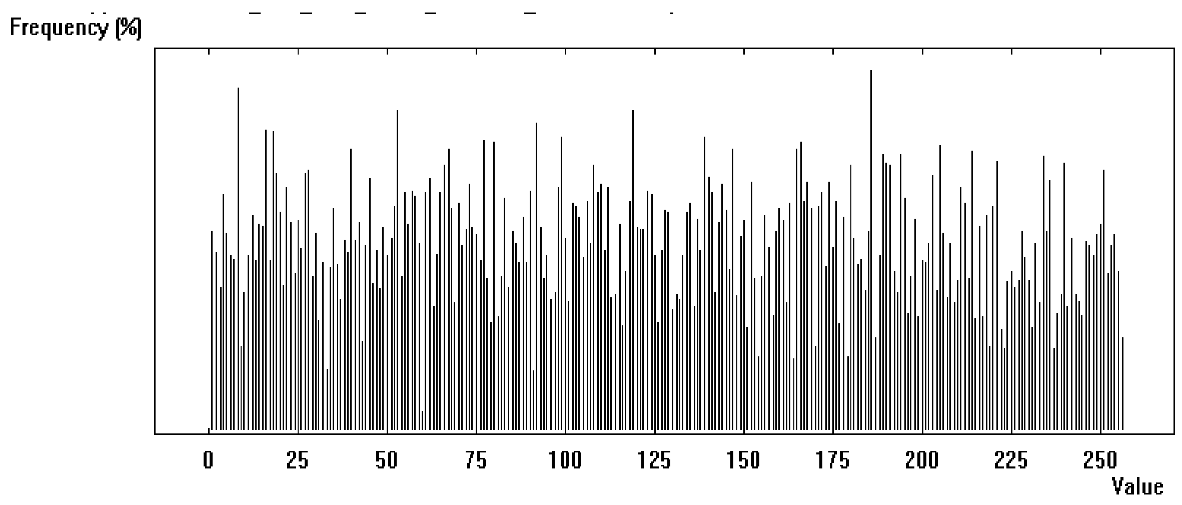


Figure 7.84: Graphical representation of frequency distribution spectrum of characters for the encrypted stream using CDHLPSCT for *.exe* file

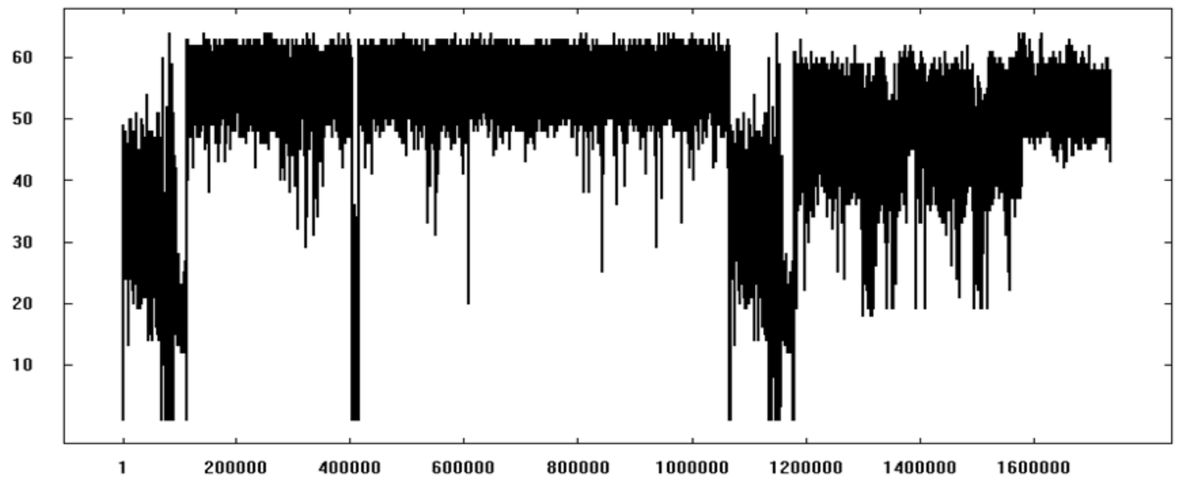


Figure 7.85: Floating frequency of the input *.exe* source stream

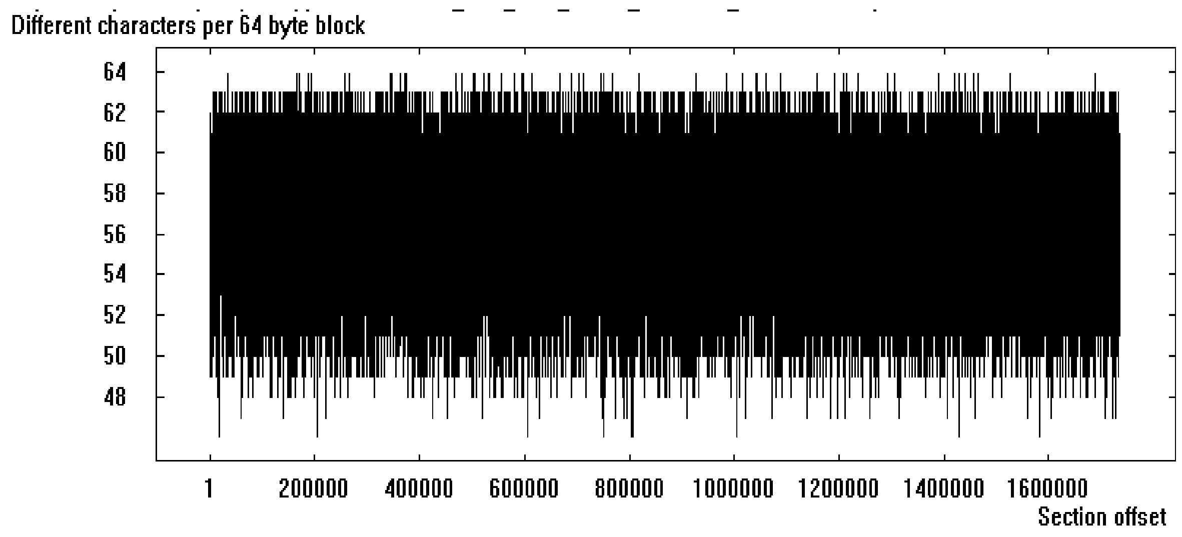


Figure 7.86: Floating frequency of the encrypted stream using CDHLPSCT for *.exe* file

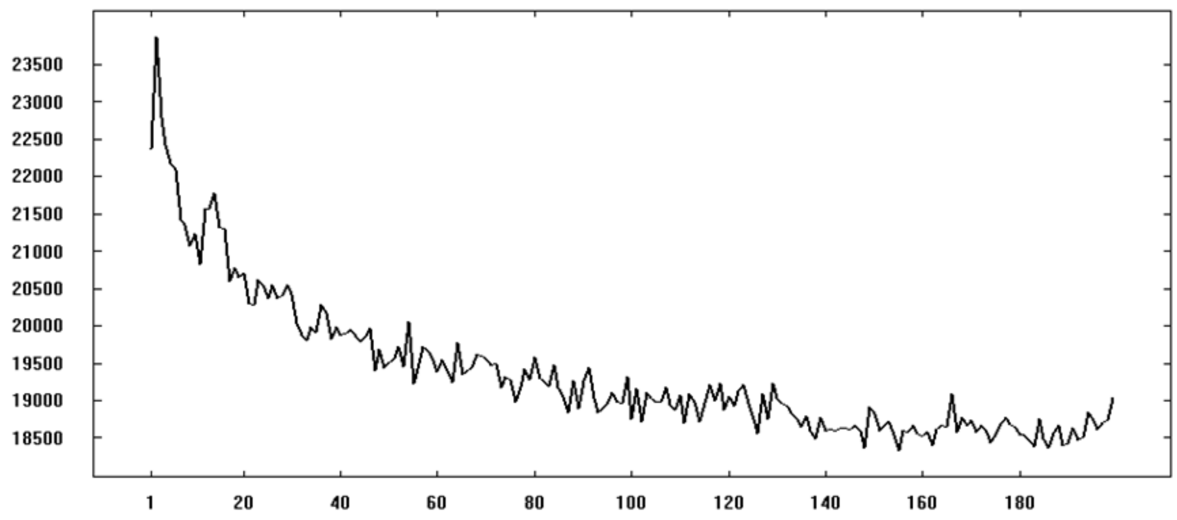


Figure 7.87: Autocorrelation of the input *.exe* source stream

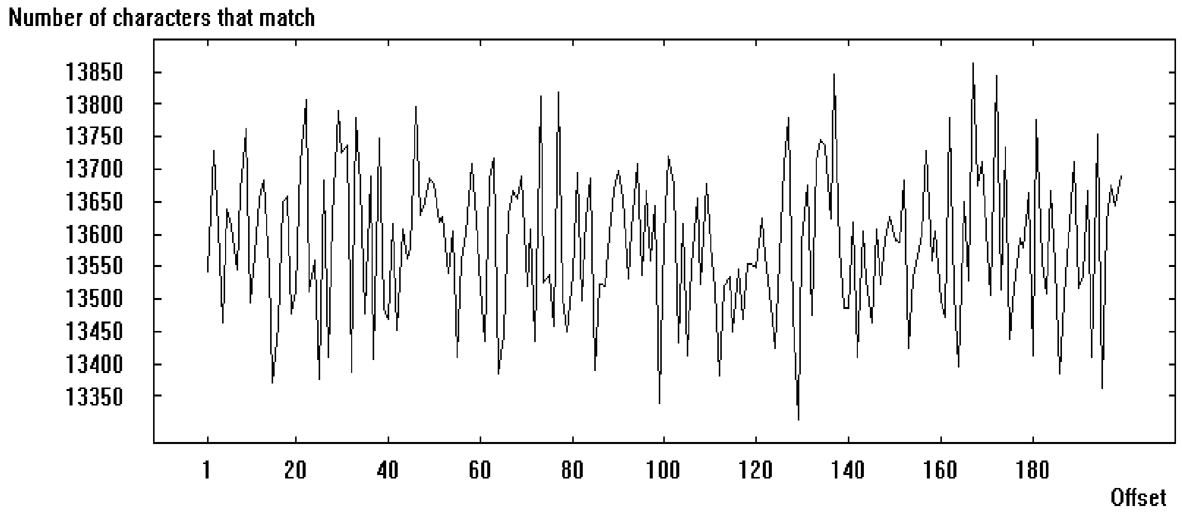


Figure 7.88: Autocorrelation of the encrypted stream using CDHLPSCT for *.exe* file

7.7.4 *.cpp* file

Analysis of character frequencies of twenty source files of *.cpp* type has been performed using CTHLPSCT. Figure 7.89 shows the spectrum of frequency distribution of characters for the input source stream. Figure 7.90 shows the spectrum of frequency distribution of encrypted characters using CTHLPSCT for the same input source stream. It has been observed that frequencies of characters are widely distributed in DHLPSCT encrypted *.cpp* file.

Analysis of entropy of twenty source files of *.cpp* type has been performed using CTHLPSCT. The entropy of a source thus indicates its characteristic distribution. It measures the average amount of information which one can obtain through observation of the source or, conversely, the indeterminacy which prevails over the generated messages when one cannot observe the source. The entropy for the input source stream is 4.06. Whereas the entropy of encrypted characters using CTHLPSCT for the same input source stream is 7.99. From the figures it is observed that entropy of CTHLPSCT encrypted characters is near to eight which indicate the high degree of security.

Analysis of floating frequencies of twenty source files of *.cpp* type has been performed using CTHLPSCT. The floating frequency of a document is a characteristic of its local information content at individual points in the document. The floating frequency specifies how many different characters are to be found in any given 64-character long segment of the document. Figure 7.91 shows the spectrum of floating frequencies of characters for the input source stream. Figure 7.92 shows the spectrum of floating frequencies of encrypted characters using CTHLPSCT for the same input source stream. From the figures it is observed that floating frequencies of CTHLPSCT encrypted characters indicates the high degree of security.

Analysis of autocorrelation of twenty source files of *.cpp* type has been performed using CTHLPSCT. The autocorrelation of a document is an index of the similarity of different sections of the document. Figure 7.93 shows the spectrum of autocorrelation of characters for the input source stream. Figure 7.94 shows the spectrum of autocorrelation of encrypted characters using CTHLPSCT for the same input source stream. From the figure it is observed that autocorrelation of CTHLPSCT encrypted characters indicate the high degree of security.

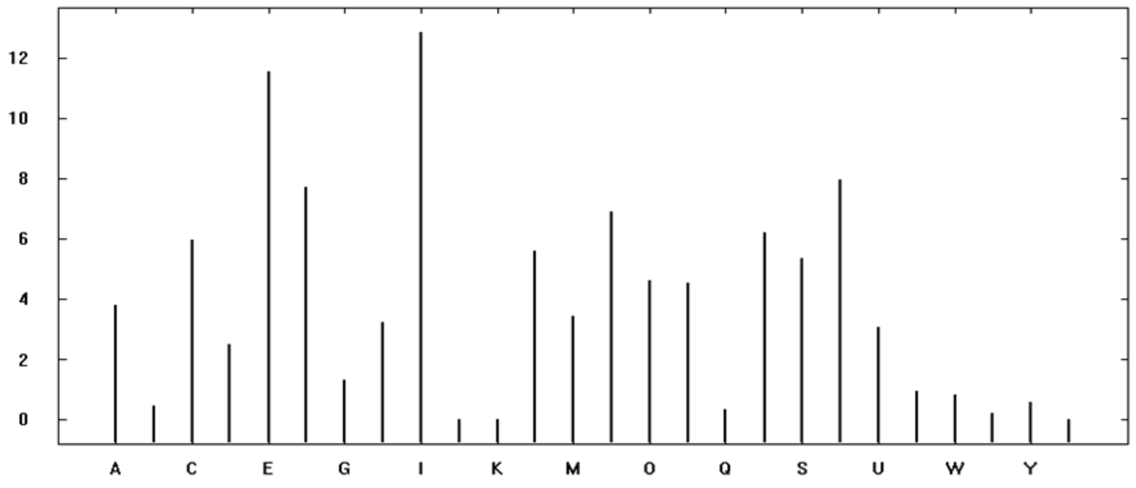


Figure 7.89: Graphical representation of frequency distribution spectrum of characters for the input *.cpp* source stream

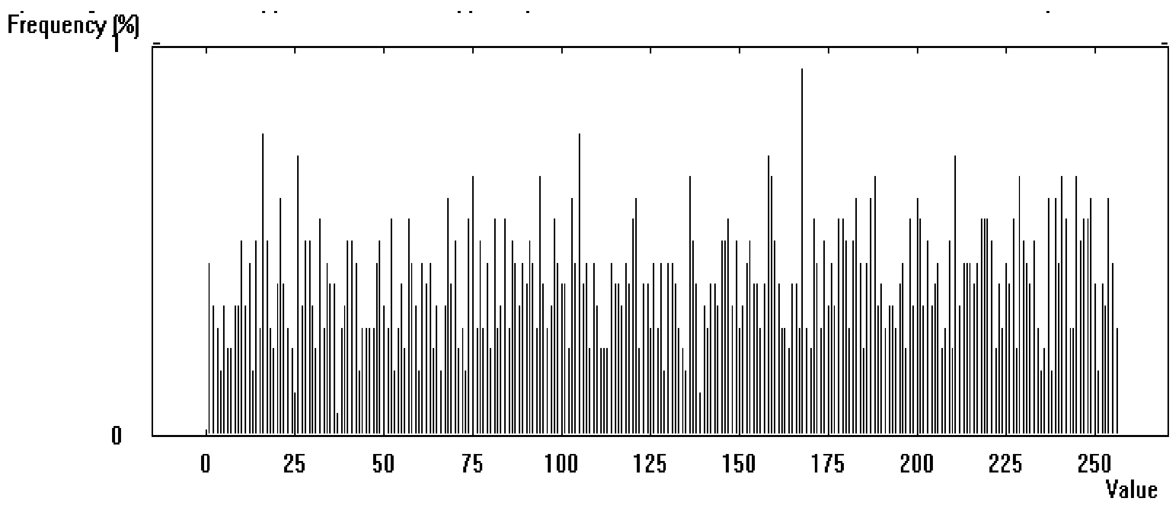


Figure 7.90: Graphical representation of frequency distribution spectrum of characters for the encrypted stream using CTHLPSCT for *.cpp* file

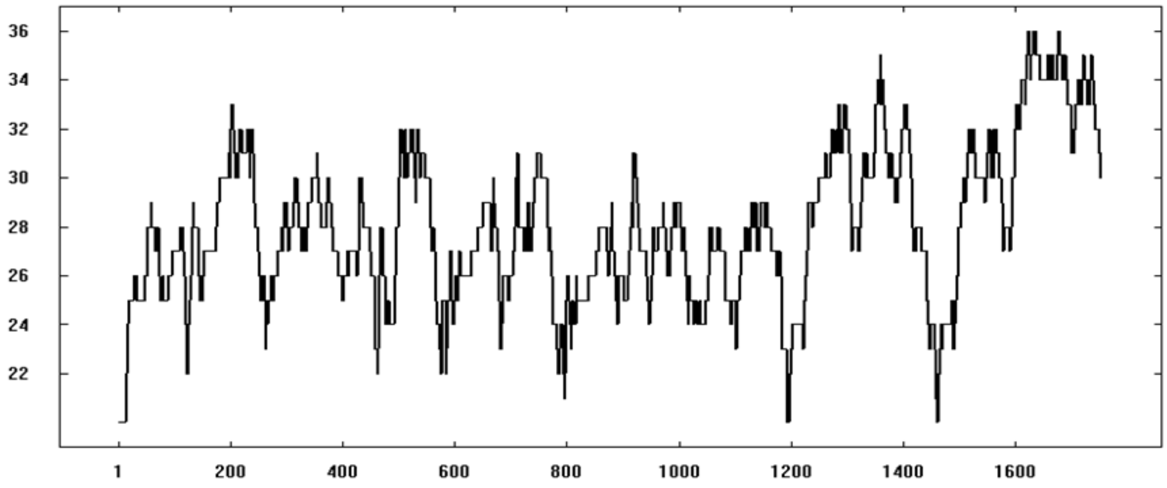


Figure 7.91: Floating frequency of the input *.cpp* source stream

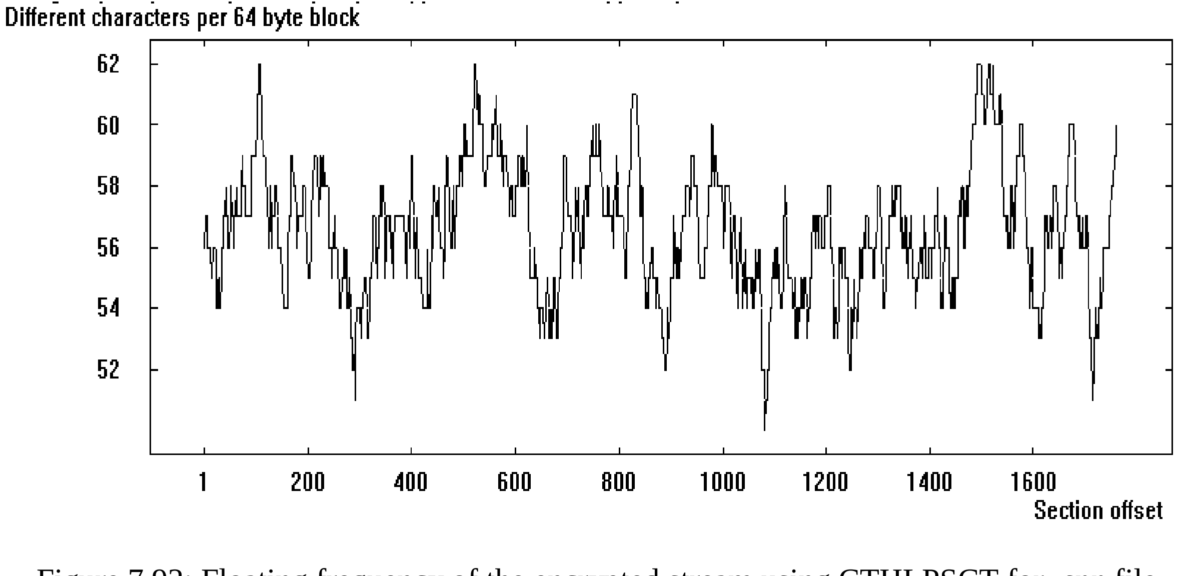


Figure 7.92: Floating frequency of the encrypted stream using CTHLPSCT for *.cpp* file

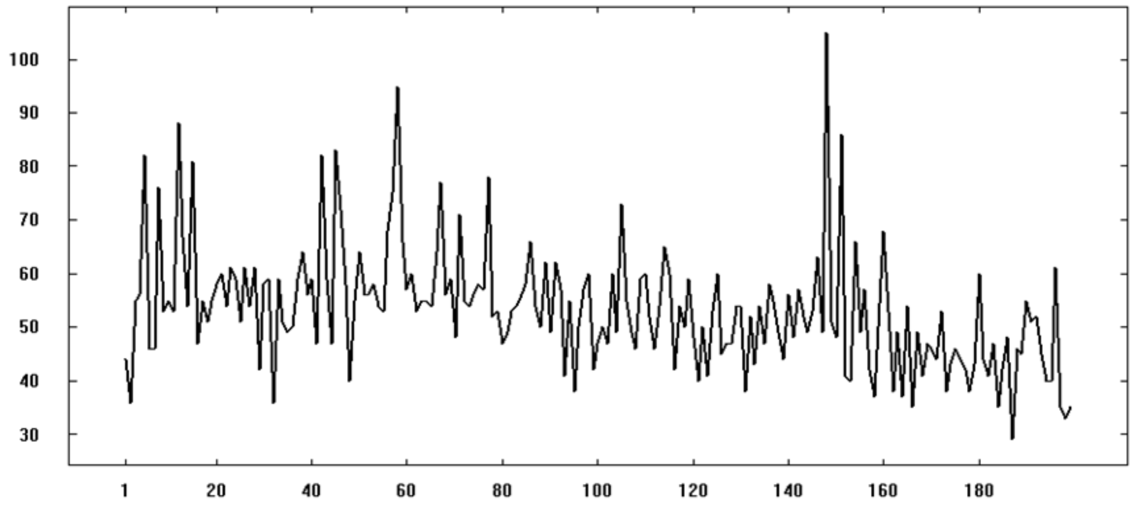


Figure 7.93: Autocorrelation of the input *.cpp* source stream

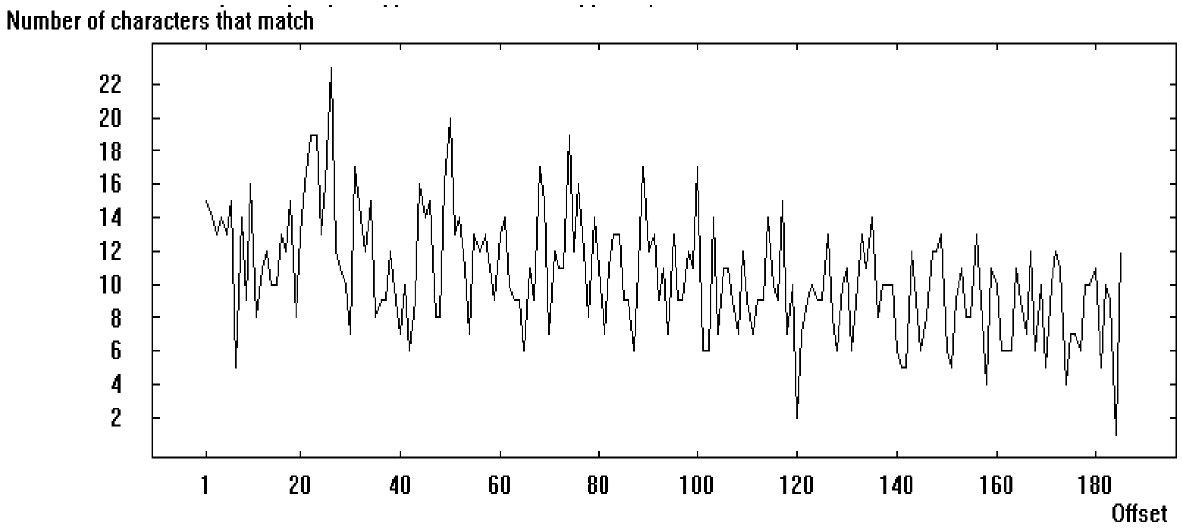


Figure 7.94: Autocorrelation of the encrypted stream using CTHLPSCT for *.cpp* file

7.7.5 *.txt* file

Analysis of character frequencies of twenty source files of *.txt* type has been performed using CGTHLPSCT. Figure 7.95 shows the spectrum of frequency distribution of characters for the input source stream. Figure 7.96 shows the spectrum of frequency distribution of encrypted characters using CGTHLPSCT for the same input source stream. It has been observed that frequencies of characters are widely distributed in CGTHLPSCT encrypted *.txt* file.

Analysis of entropy of twenty source files of *.txt* type has been performed using CGTHLPSCT. The entropy of a source thus indicates its characteristic distribution. It measures the average amount of information which one can obtain through observation of the source or, conversely, the indeterminacy which prevails over the generated messages when one cannot observe the source. The entropy for the input source stream is 4.20. Whereas the entropy of encrypted characters using CGTHLPSCT for the same input source stream is 7.99. From the figures it is observed that entropy of CGTHLPSCT encrypted characters is near to eight which indicate the high degree of security.

Analysis of floating frequencies of twenty source files of *.txt* type has been performed using CGTHLPSCT. The floating frequency of a document is a characteristic of its local information content at individual points in the document. The floating frequency specifies how many different characters are to be found in any given 64-character long segment of the document. Figure 7.97 shows the spectrum of floating frequencies of characters for the input source stream. Figure 7.98 shows the spectrum of floating frequencies of encrypted characters using CGTHLPSCT for the same input source stream. From the figures it is observed that floating frequencies of CGTHLPSCT encrypted characters indicates the high degree of security.

Analysis of autocorrelation of twenty source files of *.txt* type has been performed using CGTHLPSCT. The autocorrelation of a document is an index of the similarity of different sections of the document. Figure 7.99 shows the spectrum of autocorrelation of characters for the input source stream. Figure 7.100 shows the spectrum of autocorrelation of encrypted characters using CGTHLPSCT for the same input source stream. From the figure it is observed that autocorrelation of CGTHLPSCT encrypted characters indicate the high degree of security.

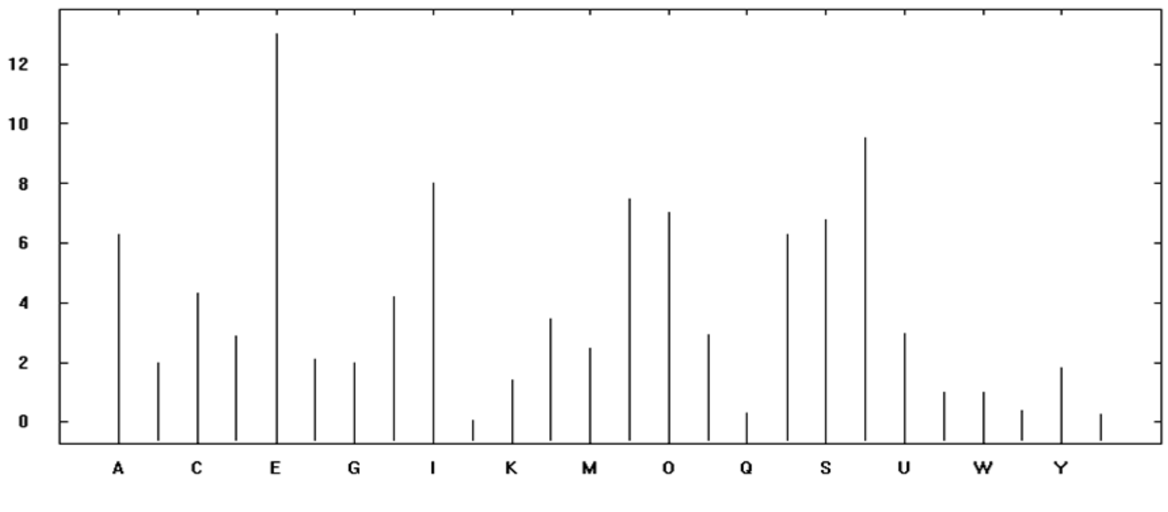


Figure 7.95: Graphical representation of frequency distribution spectrum of characters for the input *.txt* source stream

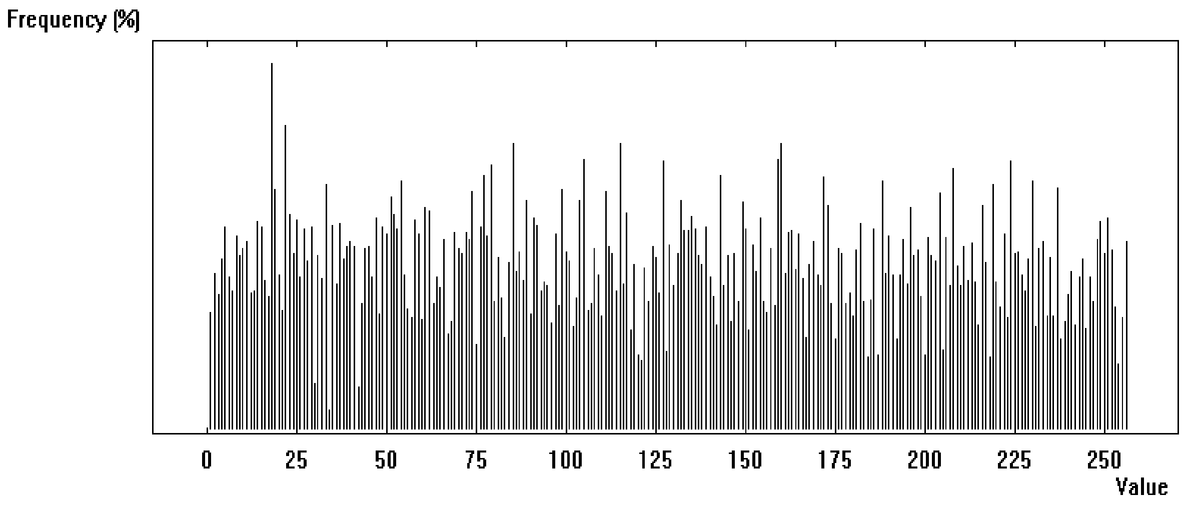


Figure 7.96: Graphical representation of frequency distribution spectrum of characters for the encrypted stream using CGTHLPSCT for *.txt* file

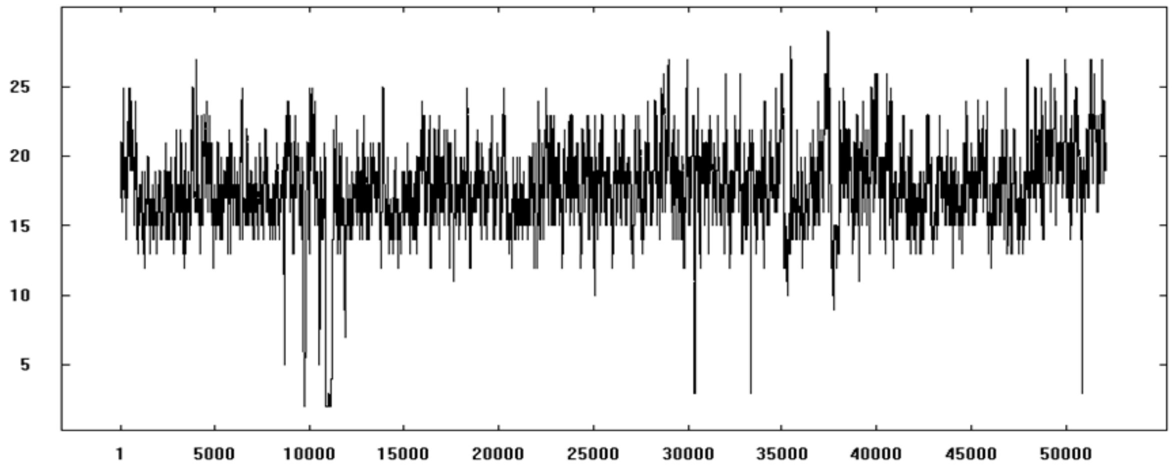


Figure 7.97: Floating frequency of the input *.txt* source stream

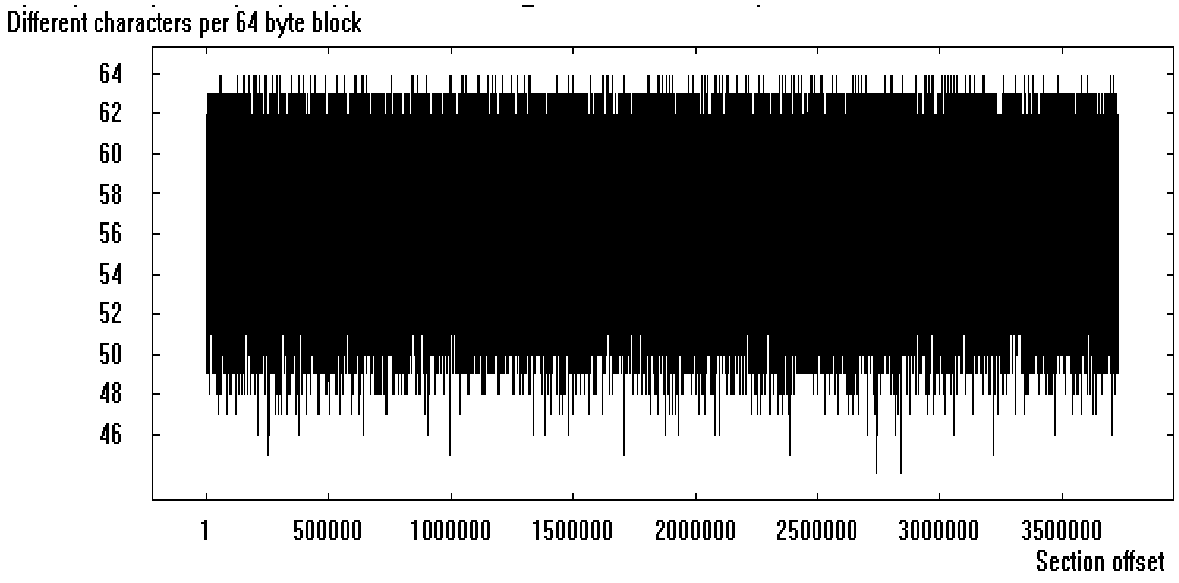


Figure 7.98: Floating frequency of the encrypted stream using CGTHLPSCT for *.txt* file

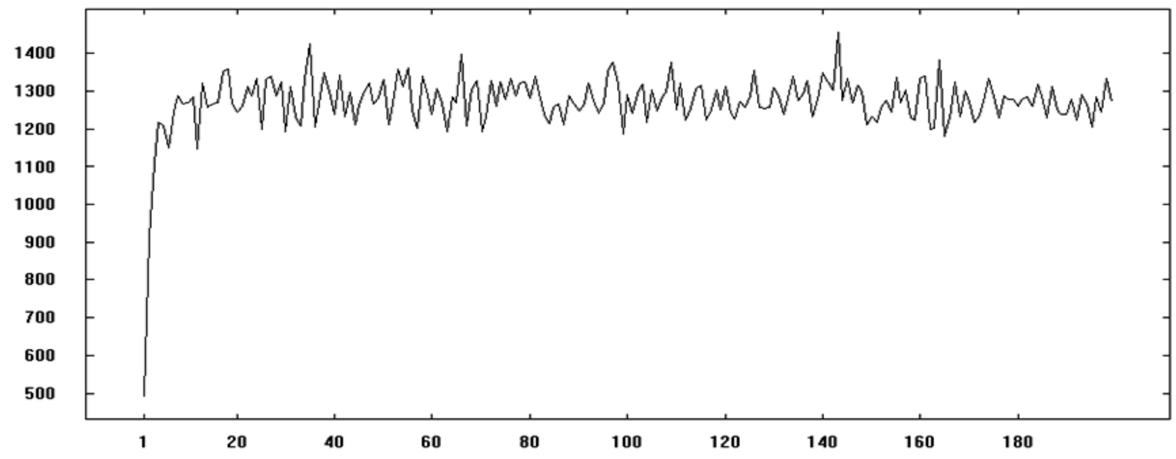


Figure 7.99: Autocorrelation of the input *.txt* source stream

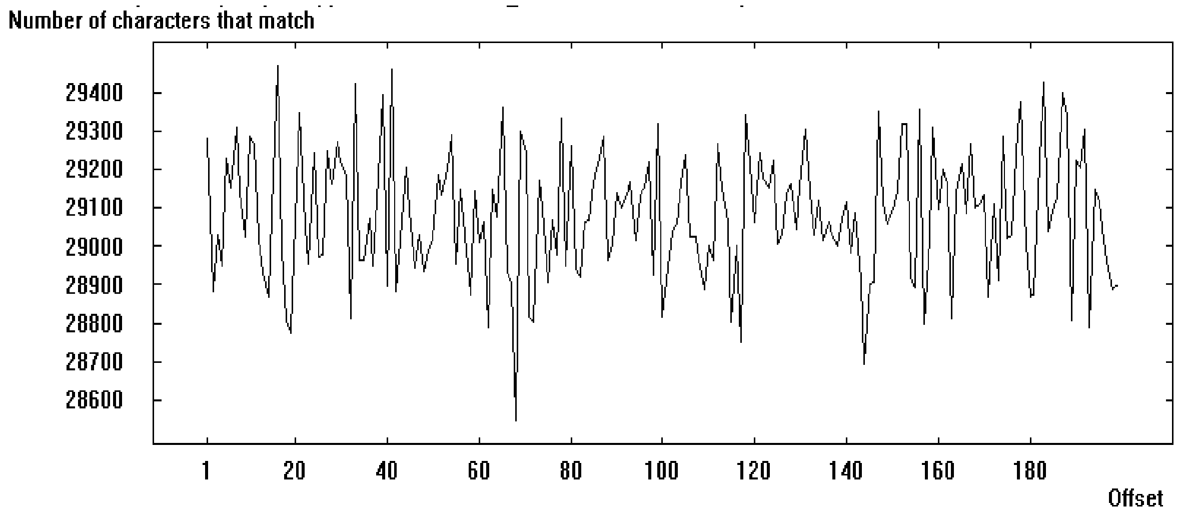


Figure 7.100: Autocorrelation of the encrypted stream using CGTHLPSCT for *.txt* file

7.7 Analysis

Analyzing all the results given in section 7.2, 7.3, 7.4,7.5, 7.6 and 7.7 following are the salient features based on comparison of proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT, CGTHLPSCT, RSA, TDES, AES, TPM, PPM, RC4 and Vernam Cipher.

a) *CGTHLPSCT outperform over all other proposed techniques and existing TPM and PPM techniques and has passed the entire 15 statistical test. This confirms the robustness and randomness of the synchronized group session key.*

b) *CGTHLPSCT uses Particle Swam Intelligence (PSI) based encryption/decryption approach. In this PSI based technique the number of keys to be stored is less than AES, RC4, Vernam Cipher and other proposed techniques*

c) *In general AES takes minimum times and TDES takes the maximum times for encryption and decryption process compare to other techniques. For TDES the encryption and decryption time both are two to three times more than that of proposed techniques.*

d) *Very little bit difference observed between the encryption and decryption times for all proposed techniques, which indicate that the computational complexity for all the process is approximately similar.*

e) *The graphs show that the encryption/decryption time increase with the increase of source file sizes. For larger file size the slope of the curve are higher.*

f) *128/192/256 bit Session key synchronization time (in cycle) for all the proposed and existing techniques in the increasing sequence of CGTHLPSCT, CTHLPSCT, CDHLPSCT, DHLPSCT, KSOMSCT and TPM, PPM. This is quite affordable in terms of resources available in wireless communication*

g) *It has been shown that in group synchronization phase CGTHLPSCT consumes less amount of memory compared to other techniques because it needs only* $nlog(n-1)$ *amount of synchronizations compared to* $n(n-1)$ *synchronizations steps in others.*

h) *The increasing order sequence of relative time spent in GC in group synchronization phase is CGTHLPSCT, CTHLPSCT, CDHLPSCT, DHLPSCT, KSOMSCT, TPM and PPM.*

i) *Avalanche, Strict Avalanche and bit Independence are cryptographic test methods which measures the degree of security. Results indicate that the Avalanche, Strict Avalanche values for proposed technique are at with other techniques. Proposed CGTHLPSCT has the maximum average Bit Independence values for .dll, .exe, .txt and .doc files compare to proposed KSOMSCT, DHLPSCT, CDHLPSCT, CTHLPSCT and existing RSA, TDES, AES techniques. These results indicate that the degree of security of the proposed CGTHLPSCT is very high and is comparable with that of other standard technique.*

j) *Proposed techniques needs less amount of threads for generating group session key compared to existing TPM and PPM. The increasing order sequence of thread required in group synchronization phase is CGTHLPSCT, CTHLPSCT, CDHLPSCT, DHLPSCT, KSOMSCT, TPM and PPM.*

k) *Proposed CGTHLPSCT needs only* $nlog(n-1)$ *compare to* $n(n-1)$ *synchronizations in other techniques.*

l) *In source files, some characters appear with very high and very low frequencies and some characters appear with zero frequency. In encrypted files all characters with ASCII values ranging from 0 to 255 appear with certain frequencies and all these characters are approximately equally distributed over a certain range. Since the frequency spectrum is smoother, so the degree of security of proposed technique is good is comparable with that of standard available cryptographic technique.*

m) *If the length of the session key get increased then the increased of average synchronization steps is linear. Linear computational complexity can be easily handled in wireless communication.*

n) *For all proposed and existing techniques, calculated Chi-Square values are than the tabulated Chi-Square values. This indicates the high degree of non-homogeneity between source and encrypted files. In case of .dll, .exe, .txt, .doc, files proposed CGTHLPSCT has maximum average Chi-Square value among all other proposed techniques.*

o) $3D$ *KSOMSCT takes more iteration to train the map in compared to* $2D$ *KSOMSCT. So, the energy consumption is more in* $3D$ *KSOMSCT than* $2D$. *For this reason* $2D$ *KSOMSCT is the best alternative in wireless communication where resource constrains (in terms of energy, memory) is a vital issues for generation of session key.*

Chapter 8

Proposed Model

8.1 Introduction

In this chapter a model of cryptographic technique through cascaded implementation embodied with proposed techniques has been discussed. The approach of cascaded implementation is an attempt to integrate the independent techniques which are discussed and analyzed in earlier chapters. The technique proposed in this chapter introduces a new dimension in the endeavor of ensuring secured session key generation and exchange for encryption/decryption to the maximum possible level.

Section 8.2 represents a brief description of the proposed technique. Section 8.3 deals with the detailed analysis of the results. Conclusions are drawn in section 8.4. Future scopes are described in section 8.5.

8.2 The Model

Five independent secured session key generation techniques proposed in various chapters, termed as KSOFM (say S_1), DHLP (say S_2), CDHLP (say S_3), CTHLP (say S_4), CGTHLP (say S_5) and five independent secured encryption/decryption techniques proposed in various chapters, termed as Fractal Triangle based encryption/decryption (say E_1), Simulated Annealing based encryption/decryption (say E_2), Genetic Algorithm based encryption/decryption (say E_3), ACI based encryption/decryption (say E_4), PSI based encryption/decryption (say E_5) have been integrated to generate the cascaded model. Number of cascaded stages, say n, is selected randomly which forms the part of the composite key of this model. This model is based on cascaded implementation of n number of techniques which are chosen randomly from among five key generation techniques and five encryption/decryption techniques with or without repetition of the same. Repetition of the same technique in consecutive cascading stage is not allowed. No technique be implemented more than t number of times where $t < n$. It may so happen that one or more out of the five key generation techniques and five encryption/decryption techniques for cascading not implement at all. The plaintext is the input stream of the first encryption technique of the sequence and the output stream generated from the nth stage of cascading in the cipher text. Whenever a technique is selected for encryption, a session key generation technique also gets selected. As a result n numbers of different encryption/decryption sub keys get generated for this implementation. A session key also gets generated using any of the session key generation techniques among five techniques. This session key helps to transmit the information regarding value of n (number of cascading stages), the order of encryption/ decryption techniques for n cascading stages ($say\ E_iE_jE_k\ldots where\ i \neq j\ , j \neq k, \ldots$) and the context of n encryption/decryption keys ($K_1, k_2, K_3, \ldots, K_n$). During decryption, the cipher text is considered as binary bit stream and passes through each of n decryption techniques in exactly the reverse order of the sequence followed during encryption. The final output stream generated from nth stage of cascading reproduced the plaintext. At any intermediate stages of this technique, the output stream of the technique of that stage is the input stream to the next cascading stage.

Section 8.2.1 describes the generation of session key of the proposed cascaded implementation and that of encryption and decryption process of the same described in the section 8.2.2 and 8.2.3 respectively.

8.2.1 Session Key Generation

The detailed mechanisms of session key generation for individual cryptographic techniques have been discussed in respective chapters. Proposed model has n number of cascading stages. At each stage, the input binary bit stream P_i passes through the encryption/decryption key generator to generate the corresponding encryption/decryption key K_i where $i \in N,$ the set of first n natural numbers. A session key S_i is generated for the proposed model and this session key used to transmit the following information to the other party.

i. *The value of n (number of cascading stages)*

ii. *The order of encryption techniques for n cascading stages ($say\ E_iE_jE_k\ldots where\ i \neq j\ , j \neq k,$ ..., and every $E_i \in \{E_1, E_2, E_3, E_4, E_5\}$)*

iii. *The information of n number of encryption/decryption keys ($say\ K_1, K_2, K_3, \ldots, K_n$) which are generated at the corresponding cascading stage of encryption using the input binary bit stream for that stage.*

The key space of the session key S is very large. The value of n can be represented by a character having ASCII value from 1 to 255. At each time a session key generation technique is selected for the whole process out of five different session key generation technique $(S_1, S_2, S_3, S_4, S_5)$ randomly based on some constraints and at each cascading stage a cryptographic technique is selected out of five different encryption/decryption technique $(E_1, E_2, E_3, E_4, E_5)$ randomly based on some constraints. For each session key generation technique only three bits are required to store (the three bits combinations range 000 to 111 is sufficient to store the session key generation technique index 1 to 5). For each encryption/ decryption technique only three bits are required to store (the 3 bit combinations range 000 to 111 is sufficient to store the encryption/ decryption technique index 1 to 5). So $(3 \times n)$ number of bits i.e. $\left(\frac{3\times n}{8}\right)$ number of characters are required to store the sequence of cryptographic techniques for n cascading stages. In various chapters, five different, independent encryption/decryption key generation techniques have been discussed in detail. All of these techniques generate $128/192/256$ bits encryption/decryption keys. Proposed model has n number of cascading stages and at each stage of encryption an encryption key is generated for that corresponding technique. Each encryption key has a length of $128/192/256$ bits So, the length of n number of encryption/decryption keys is $(128 \times n)$ to $(256 \times n)$ number of bits In various chapters, five different, independent session key generation techniques have been discussed in detail. All of these techniques generate $128/192/256$ bits tuned session keys. This tuned session key get *Exclusive-OR* with the session key produced by this proposed model and transmitted to the other party. The receiving party has the same tuned session key, using this tuned session key receiving party perform the *Exclusive-OR* operations on the receiving stream to get back the session key of the proposed model. The proposed model has a session key of length [(value of number of cascading stages in bits) + (three bits combinations of encryption/ decryption technique index) + (length of n number of encryption/decryption keys in bits) + (length of n number of session keys in bits)] i.e. $[\,8 + (3 \times n) + (128 \times n) + (128 \times n)\,]$ bits to $[\,8 + (3 \times n) + (256 \times n) + (256 \times n)]$ number of bits. So, $\frac{[\,8+(3\times n)+(128\times n)+(128\times n)]}{8} = \left[1 + \frac{(3\times n)}{8} + 16n + 16n\right] = 32n$ to $\frac{[\,8+(3\times n)+(256\times n)+(256\times n)]}{8} = \left[1 + \frac{(3\times n)}{8} + 32n + 32n\right] = 64n$ numbers of characters this confirms a huge variability of the key space in terms of randomness.

8.2.2 Encryptor Module

The detailed discussion on encryption techniques of all schemes (E_1, E_2, E_3, E_4, E_5) have been made individually into its respective chapters. Proposed model has n number of cascaded stages where n is a finite random integer. The plaintext is a binary bit stream which is the input for the first chosen encryption technique. The output stream of n^{th} technique is the cipher text. At any intermediate stages of this approach, the output stream of the encryption technique is made input to the next cascading stage. The sequence of encryption techniques is selected randomly. The assumption of the model is that the consecutive repetition of same technique is not permitted. No technique be implemented more than t number of times where $t < n$. One or more out of the five available techniques for cascading (E_1, E_2, E_3, E_4, E_5) may not be implemented at all. In maiden stage any one out of five techniques can be chosen in five ways and then for remaining stages $(n-1)$ times, at each cascading stage any one out of four (since consecutive repetition of same technique is not allowed) techniques can be chosen in four ways. So, there are as many as $5 \times 4^{n-1}$ ways to choose a cascading sequence. Now, $5 \times 4^{n-1} = \frac{5}{4} \times 4^n = 1.25 \times 4^n$, which means that the formation of session key is order of 4^n ways which is a huge one. It also indicates that the key space of the session key is very large. The most importantly, it is to be noted that the session key is used only once for each transmission. So there is a time stamp of minimum span which expires automatically at end of transmission. By notation, the sequence of encryption techniques for n cascading stages is represented as $E_i E_j E_k \ldots E_u E_v E_w$ $where, i \neq j, j \neq k, \ldots, u \neq v, v \neq w.$

Input : Source stream i.e. plaintext

Output : Encrypted stream i.e. cipher text

Method : The process takes binary stream and generates encrypted bit stream through cascaded encryption operations.

```
Step 1.   The input stream, say $P_0$ is taken as a stream with finite number of binary bits
Step 2.   Obtain the number of cascaded stages, say n, randomly
Step 3.   Set $i = 0$ and initialize $T_0 = E_0$ (i.e. Null)
Step 4.   The encryption key $K_{i+1}$ is generated using the binary bit stream $P_i$
Step 5.   Select $T_{i+1} \in \{ E_1, E_2, E_3, E_4, E_5\}$ randomly in such a way that $T_{i+1} \neq T_i$
Step 6.   The bit stream $P_i$ is encrypted into $P_{i+1}$ using the encryption technique $T_{i+1}$ and the key encryption $K_{i+1}$
Step 7.   Set $i = i + 1$. If $i < n$ then go to step 8 else go to step 9
Step 8.   $P_i$ is the input stream for the next cascading stage and go to step 4
Step 9.   $P_i \cong P_n$ is the final output of the encryptor module i.e. $P_n$ is the cipher text.
```

Figure 8.1 shows the flowchart of encryptor module for the proposed model.

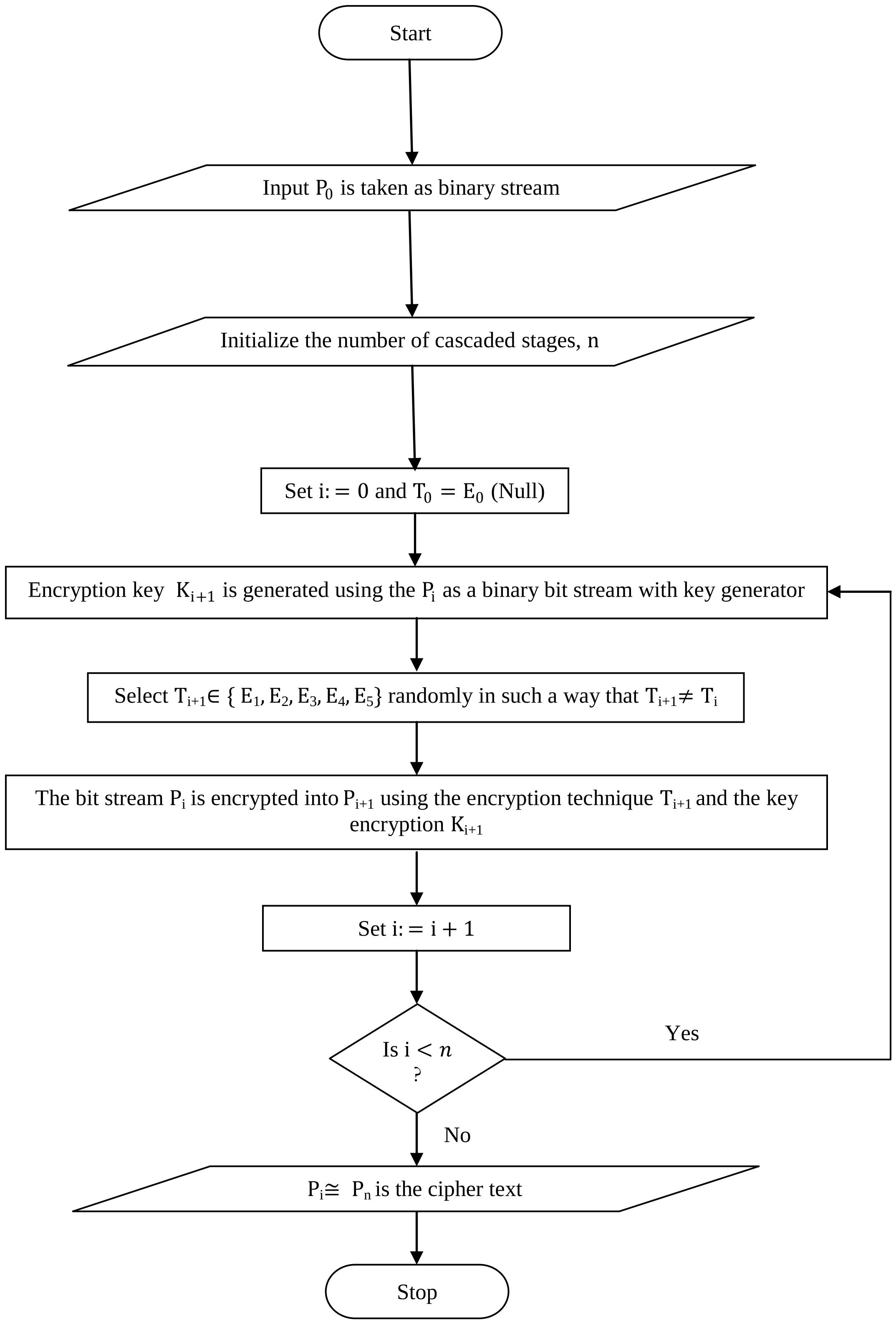

Start
Input P_0 is taken as binary stream
Initialize the number of cascaded stages, n
Set $i := 0$ and $T_0 = E_0$ (Null)
Encryption key K_{i+1} is generated using the P_i as a binary bit stream with key generator
Select $T_{i+1} \in \{E_1, E_2, E_3, E_4, E_5\}$ randomly in such a way that $T_{i+1} \neq T_i$
The bit stream P_i is encrypted into P_{i+1} using the encryption technique T_{i+1} and the key encryption K_{i+1}
Set $i := i + 1$
Is $i < n$?
Yes
No
$P_i \cong P_n$ is the cipher text
Stop

Figure 8.1: Pictorial representation of the flow chart of encryption for the proposed cascaded model

8.2.3 Decryptor Module

The detailed discussion on decryption techniques of all schemes $(D_1, D_2, D_3, D_4, D_5)$ have been discussed individually into its respective chapters. Proposed model has n number of cascaded stages where n has selected at random during decryption. Processing the information of the session key S, the value of n (number of cascading stages), the order of the encryption technique for n cascading stages $(E_iE_jE_k\dots E_uE_vE_w\, where, i \neq j, j \neq k, \dots, u \neq v, v \neq w)$ and the information of n number of encryption/decryption keys $(K_1, K_2, \dots, K_n)$ are fetched during decryption. Order of decryption is fetched from the session key S which is exactly reverse of the sequence of encryption and initialized into the variables $\mathrm{T}_1, \mathrm{T}_2, \mathrm{T}_3, \dots, \mathrm{T}_n$ (i.e. T_1 is the first decryption technique, T_2 is the second decryption technique and so on) where $\mathrm{T}_i \in \{\mathrm{D}_1, \mathrm{D}_2, \mathrm{D}_3, \mathrm{D}_4, \mathrm{D}_5\} \forall \mathrm{i} \in \mathrm{N}$, the set of first n natural numbers. The order of the decryption is exactly the reverse of the sequence followed during encryption i.e. $D_wD_vD_u\dots D_kD_jD_i$ and the decryption key which will be used during decryption in the order of $K_n, K_{n-1}, \dots, K_2, K_1$. The first decryption technique D_w considers the input cipher text P_n as a binary bit stream. The final output stream generated from the final stage of cascading using the decryption technique D_i reproduced the plaintext. At the intermediate stages of this approach, the output stream of any decryption technique is the input stream to the next cascading stage.

The decryption algorithm is described as follows:

Input : *Encrypted stream i.e. cipher text and the session key S*

Output : *Source stream i.e. plaintext*

Method : *The process takes encrypted binary stream and generates decrypted bit stream through cascaded decryption operations.*

Step 1. *The stream containing the information of the session key S obtained to get the information about the decryption key*

Step 2. *The value of n (number of cascading stages) and the decryption keys $K_1, K_2, K_3, \dots, K_n$ are extracted from the session key S and used for decryption*

Step 3. *The order of decryption is fetched from the session key S which is exactly reverse of the sequence of encryption and initialized into the* *variables $T_1, T_2, T_3, \ldots, T_n$ (i.e. T_1 is the first decryption technique, T_2 is the second decryption technique and so on) where $T_i \in \{ D_1, D_2, D_3, D_4, D_5\} \forall i \in N$, the set of first n natural numbers*

Step 4. *The input stream, say P_n, is taken as a stream with finite number of binary bits.*

Step 5. *Set $i = n$*

Step 6. *Input bit stream P_i is decrypted into P_{i-1} using the decryption technique T_{n-i+1} and the decryption key K_i*

Step 7. *Set i=i-1. If i>0 then go to step 8 else go to step 9*

Step 8. *P_i is the input stream for the next cascading stage and goes to step 6*

Step 9. *$P_i \cong P_0$ is the final output of the decryptor module i.e. P_0 is the plaintext.*

Figure 8.2 shows the flowchart of decryptor module for the proposed model.

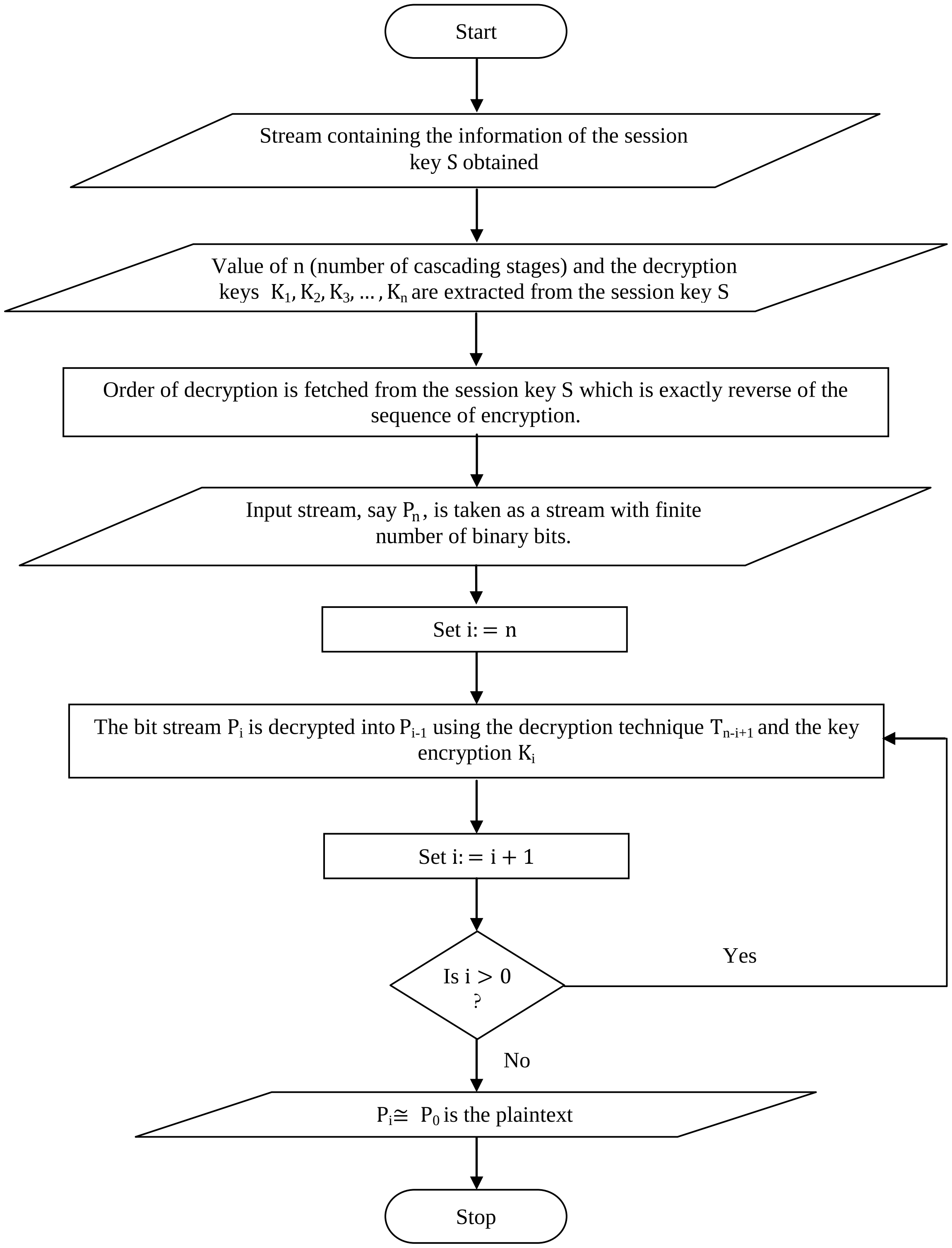

Start
Stream containing the information of the session key S obtained
Value of n (number of cascading stages) and the decryption keys $K_1, K_2, K_3, \ldots, K_n$ are extracted from the session key S
Order of decryption is fetched from the session key S which is exactly reverse of the sequence of encryption.
Input stream, say P_n, is taken as a stream with finite number of binary bits.
Set $i := n$
The bit stream P_i is decrypted into P_{i-1} using the decryption technique T_{n-i+1} and the key encryption K_i
Set $i := i + 1$
Is $i > 0$?
Yes
No
$P_i \cong P_0$ is the plaintext
Stop

Figure 8.2: Pictorial representation of the flow chart of decryption for the proposed cascaded model

8.3 Analysis

Extensive analysis has been made with a huge variability of the value of n, the number of cascading stages. Out of eighty samples files (four different types and twenty files for each type) only twelve files, arbitrarily chosen three from each type, taken for testability of the model and result are generated.

The encryption and decryption times taken are the differences between processor clock stick at the starting of execution and at the end of execution respectively. Since the CPU clock ticks taken as time, there might be a slight variation in actual time which is insignificant and may be ignored. Proposed model has n number of cascading stages. So the encryption and decryption time of the proposed approach are near equal to the cumulative sums of n number of encryption and decryption times respectively of the individual cryptographic techniques. Therefore the encryption and decryption times are larger than that of any individual method.

Comparison between the source and encrypted bytes has been performed and changes of bits within encrypted bytes has been observed for a change of single bit in the original message byte for the entire or a relative large number of bytes. Detail concept of Avalanche, Strict avalanche and Bit independence test has been discussed in chapter 7. The values of three above mentioned tests are based on pure numbers and this has no units. The calculated Avalanche, Strict avalanche and Bit independence values are very high which may indicate good security of the proposed approach. There are no significant differences observed between the calculated avalanche, Strict avalanche and Bit independence values for the Proposed approach using cascaded implementation and that for any individual technique.

Spectrum of the frequency distribution of the encrypted characters generated using the proposed approach are analyzed and it is observed that characters with ASCII values ranging from 0 to 255 appeared all with near equal frequencies which may indicate that it is very hard to regenerate the original file for a cryptanalyst. Difference between high and low value of frequencies in the frequency distribution curves is very small. So the spectrum of frequency distribution generated using the proposed approach are nearly smoother which may indicate that the degree of security of the proposed approach is good. No remarkable differences in the spectrum of frequency distribution have been observed for the cascaded implementation.

Chi-Square value is calculated from the character frequencies using the formula devised by Karl Pearson which is called “Pearsonian Chi-Square”. The higher the Chi-Square values the more deviation from the original message. In chapter 7 the detail concept of the test of non-homogeneity has been discussed. The calculated Chi-Square values for all the sample files using the proposed approach are very large compare to tabulated one which may indicate that the degree of security of the proposed approach using cascaded implementation is good. There is no noticeable difference has been observed from calculated Chi-Square values, which confirms the high degree of non-homogeneity of the encrypted stream with respect to the source stream.

Cryptographic algorithms are possible to break without keys where a cryptanalyst try all possible keys until get the success. The security of a cryptographic scheme depends on how much effort along with its time stamp is required for the cryptanalyst to break it. But it is always very difficult to estimate the amount of effort required to decrypt the cipher text successfully. In other word, an encryption scheme may be defined as unconditionally secured if the cipher text generated by the scheme does not contain enough information to determine uniquely the corresponding plaintext, no matter how much cipher text is available.

The complexity of any symmetric encryption algorithm is generally compared with the Brute-force attack. Brute-force approach simply involves computing every possible key until an intelligible translation of the cipher text into plaintext is obtained. On average, half of all possible keys must be tried to achieve success. Table 8.1 shows how much time is involved for various key spaces. The 56-bit key size is used with the DES algorithm and 168-bit key size is used for triple DES (i.e. TDES) algorithm. The minimum key size specified for AES is 128 bits. For each key size, the results are shown assuming that it takes 1µs and 10^6 µs to perform a single decryption respectively.

Table 8.1
Time involved for various key spaces

Key Size	Number of Alternate Keys	Time required at 1 decryption/ μs	Time required at 10^6 decryption/ μs
32 bits	$2^{32} = 4.3 \times 10^9$	2^{31}μs= 35.8 minutes	2.15 milliseconds
56 bits	$2^{56} = 7.2 \times 10^{16}$	2^{55}μs= 1142 years	10.01 hours
128 bits	$2^{128} = 3.4 \times 10^{38}$	2^{127}μs= 5.4×10^{24} years	5.4×10^{18} years
168 bits	$2^{168} = 3.7 \times 10^{50}$	2^{167}μs= 5.9×10^{36} years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	2×10^{26}μs= 6.4×10^{12} years	6.4×10^6 years

In section 8.2.1 of this chapter, the length of the session key of the proposed model has been discussed and it has approximately $64n$ number of characters (any character with ASCII value from 0 to 255). Therefore number of alternate keys = 256^{64n}. Since on an average half of all possible keys must be tried to achieve success, so total time required at 1 decryption/ μs=0.5×256^{64n} μs = $0.5 \times 2^{8\times 64n}$ μs = 0.5×2^{512n} μs = $2^{(512n-1)}$ μs. Table 8.2 shows how much time is involved for exhaustive search of the session key S for the proposed model with various n values. Analyzing the data of this table it may be concluded that the proposed model is highly secured from Brute-force attack.

Table 8.2
Average time required for exhaustive key search

n values	Number of Alternate Keys	Time required at 1 decryption/ μs	Time required at 10^6 decryption/ μs
1	$256^{64\times 1} = 2^{512}$	2^{511}μs= 2.13×2^{140} years	2.13×2^{134} years
2	$256^{64\times 2} = 2^{1024}$	2^{1023}μs= 2.85×2^{294} years	2.85×2^{288} years
3	$256^{64\times 3} = 2^{1536}$	2^{1535}μs= 3.82×2^{448} years	3.82×2^{442} years
4	$256^{64\times 4} = 2^{2048}$	2^{2047}μs= 5.12×2^{602} years	5.12×2^{596} years
5	$256^{64\times 5} = 2^{2560}$	2^{2559}μs= 6.87×2^{756} years	6.87×2^{750} years
6	$256^{64\times 6} = 2^{3072}$	2^{3071}μs= 9.21×2^{910} years	9.21×2^{904} years
7	$256^{64\times 7} = 2^{3584}$	2^{3583}μs= 1.23×2^{1065} years	1.23×2^{1059} years
8	$256^{64\times 8} = 2^{4096}$	2^{4095}μs= 1.66×2^{1219} years	1.66×2^{1213} years

Let T be the average time in years required at 10^6 decryptions per μs for exhaustive search of the session key for the proposed model. If n, number of cascading stages, is plotted along X-axis and $log_{10}T$ along Y-axis then the generated curve is a straight line which is shown in figure 7.3. Extending this straight line along positive X-axis, it may predict the required average time T in years for any large value of n. Since the slope value of that straight line is very high and the value of T is plotted as $log_{10}T$, so the value of T is increased sharply with the increase of n.

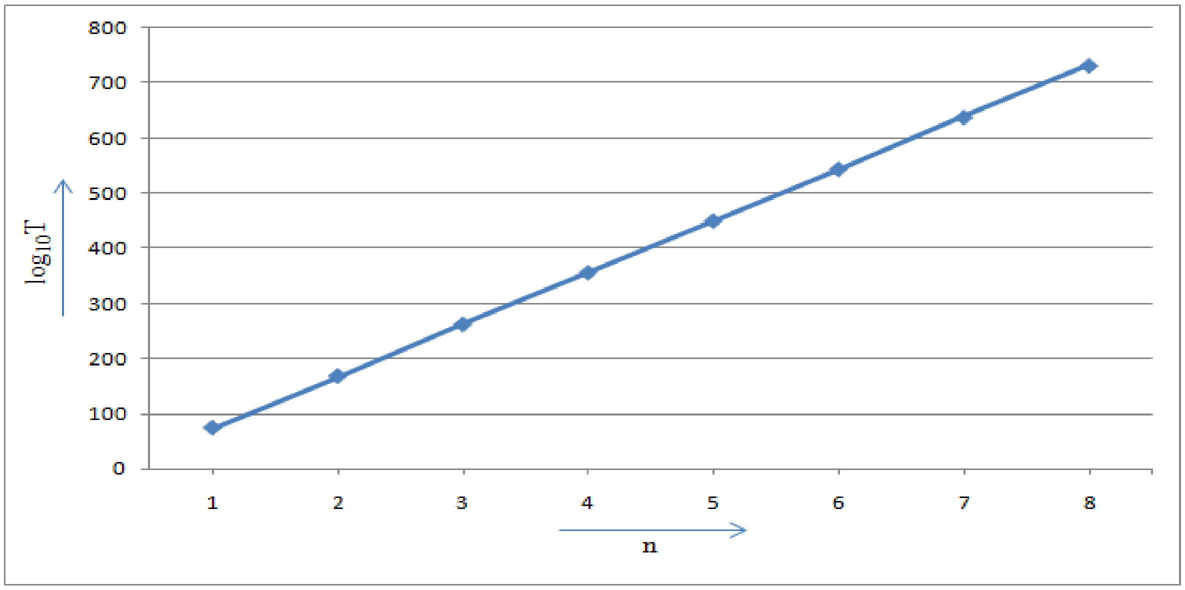


Figure.8.3: Graphical representation of average time T in years (T in logarithmic scale as $\log_{10}T$) against n, number of cascading stages

An ideal encryption procedure should be sensitive with the secret key. It indicates that the change of a single bit in the secret key should produce a completely different cipher stream and the decryption with a slightly different key fails completely. It is observed that the proposed model generates an entirely different cipher stream with the change of a single bit randomly in the key K. It is also noticed that the model totally fails to decrypt the cipher stream into plaintext with a slightly different secret key. From this point of view, it may be concluded that the proposed model is highly key sensitive.

8.4 Conclusions

The approach of cascaded implementation is very simple and logical. The analysis of results also indicates enhanced security of this approach. The strength of this proposed approach through cascaded implementation is not highlighted through the metrics for evaluation. No universal conclusion can be drawn from that above discussion regarding this approach. The real strength of the proposed approach lies in the possible formation of a large key space. The key space increases drastically with allowing the much more cascading stages. The proposed model is highly secured from Brute-force attack. Other strength of this proposed model is the adoption of complexity based on energy and resource available in the wireless communication, infrastructure for computing in a node or mesh in wireless communication. For a wireless network having low energy, the number of cascading stages be less. So, the model is very much suitable for the security of the system where energy and resource is one of the main constraints. One of the most important features of the proposed model is that the model is idle to trade-off between security and performance of light weight devices having very low processing capabilities or limited computing power. The proposed model is applicable to ensure very high security for file transmission in any form in any size.

Some of the salient features of proposed technique can be summarized as follows:

a) *Session key generation and exchange – The session key can be formed in order of 4^n ways which is a vast one and the length of session key is approximately $64n$ number of characters where n is the number of cascading stages. It indicates that the key space of the session key is very large. Again identical tuned session key can be generate after the tuning of network in both sender and receiver side using any of the proposed key generation techniques. This tuned session key can be used to encrypt session key generated by the proposed model for transmission to the other party which provides another level of security. Since the session key is used only once for each transmission, so there is a minimum time stamp which expires automatically at the end of each transmission of information.*

b) *Degree of security – Proposed technique does not suffers from cipher text only Attack, known plaintext attack, chosen plaintext attack, Chosen cipher text only attack, brute* *force attack. The number of alternate keys for the proposed model is approximately* $256^{64\times n}$*. So, model is highly secured from Brute-force attack.*

c) *Variable block size – Encryption algorithm can work with any block length and thus not require padding, which result identical size of files both in original and encrypted file. So, proposed technique has no space overhead.*

d) *Variable size key –variable size session key with high key space can be used in different session. Since the session key is used only once for each transmission, so there is a minimum time stamp which expires automatically at the end of each transmission of information. Thus the cryptanalyst will not be able guess the session key for that particular session.*

e) *Complexity – Proposed technique has the flexibility to adopt the complexity based on infrastructure, resource and energy available for computing in a node or mesh through wireless communication. So, the proposed technique is very much suitable in wireless communication.*

f) *Non-homogeneity – Measures of central tendency, dispersion and Chi-Square value have been performed between the source and corresponding cipher streams generated using proposed technique. All the measures indicate that the degree of non-homogeneity of the encrypted stream with respect to the source stream is good.*

g) *Floating frequency – In this proposed technique it is observed that floating frequencies of encrypted characters are indicates the high degree of security of proposed technique.*

h) *Entropy – In this proposed technique it is observed that entropy of encrypted characters is near to eight which indicate the high degree of security of proposed technique.*

i) *Correlation between source and encrypted stream – The cipher stream generated through proposed technique is negligibly correlated with the source stream. Therefore the proposed technique may effectively resist data correlation statistical attack.*

j) *Key sensitivity – Proposed method generates an entirely different cipher stream with a small change in the key and technique totally fails to decrypt the cipher stream with a slightly different secret session key.*

k) *Trade-off between security and performance – The proposed technique may be ideal for trade-off between security and performance of light weight devices having very low processing capabilities or limited computing power in wireless communication.*

8.5 Future Scope

The work presented in this thesis leaves further investigations in some areas. Some of the apparent future investigations may be:

a) *Intermixing of other popular soft computing based approach – Some well-known and popular soft computing based approaches like Artificial Immune System (AIS), Differential evolution, Support Vector Machine (SVM), Fuzzy Logic can be intermixed for generation of session key by tuning in wireless communication.*

b) *Effectiveness- Comparisons of the proposed model with other well-known and popular cryptographic algorithms like Blowfish, RC2, RC5, TEA, XTEA, IDEA, Serpent etc. can be performed to ensure the effectiveness further of the proposed technique.*

c) *Differential Analysis - It may be possible to find out a meaningful relationship between the source stream and encrypted stream making a slight change such as modifying a single bit of the encrypted stream. If one minor change in the source stream can cause a significant change in the encrypted stream then this differential attack would become very inefficient and practically useless, To resist the differential attack differential analysis on the encrypted stream is necessary.*

d) *Encryption Quality – A measure of encryption quality of the proposed model may be expressed as the deviation between the source stream and the encrypted stream. The encryption quality is also a function of secret key length.*

Inspite of various limitations and scope of future upgradability, there are good potential in each of individual proposed soft computing based cryptographic techniques and also in the proposed model. Incorporation of tuning of networks over public channel for generation of session key introduced a novel idea out of which more security may be obtained. In the proposed model, a huge variability of key space has been introduced which is most sensitive to the cipher text with the minimal change. The proposed model is highly flexible to adopt the complexity of any light weight computing system and idle to trade-off between security and performance of light weight device in wireless communication having limited resources. From the study, incorporation of proposed techniques the security of wireless communication may be enhanced. As a result, the proposal of the thesis may be useful for the researchers and stakeholders.

References

[1] Feistel, H. (1973). *Cryptography and Computer Privacy*. Scientific American, May 1973, Vol. 228 No. 5, pp.15-23.

[2] Rivest, R. L. (1990). *Cryptology*. In A. Jan Van Leeuwen (Ed.), *Handbook of Theoretical Computer Science*, chapter 13, pp.717-755, Elsevier / MIT Press.

[3] Stinson, D. R. (1995). *Cryptography, Theory and Practice*, CRC Press.

[4] Bellare, Mihir, Rogaway, & Phillip (2005). *Introduction*. *Introduction to Modern Cryptography*, pp.10.

[5] Menezes, A.J., Vanstone, S.A., & Van Oorschot, P.C. (1996). *Handbook of Applied Cryptography*, In: *Applied Cryptography*, CRC Press, Boca Raton.

[6] Cryptography. Retrieved August 04 2012, from http://en.wikipedia.org/wiki/Cryptography

[7] Kahn, D. (1967). *The Codebreakers*, ISBN 0-684-83130-9.

[8] Encryption, Retrieved August 05 2012, from http://en.wikipedia.org/wiki/Encryption

[9] Encryption Basics, EFF Surveillance Self-Defense Project. (n.d.). Retrieved Nov 06 2013, from https://ssd.eff.org/tech/encryption.

[10] Goldreich, Oded. (2004). *Foundations of Cryptography*: *Volume 2, Basic Applications*. Vol. 2. Cambridge university press.

[11] Kahate, A. (2010). *Cryptography and Network Security*, 2nd edition, Tata McGraw Hill.

[12] Cipher, Retrieved August 05 2012, from http://en.wikipedia.org/wiki/Cipher

[13] Schneier, B. (1995). *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 2nd edition. Wiley, New York.

[14] Stallings, W. (2003). *Cryptography and Network Security: Principles and Practices*, 3rd edition, Pearson Education.

[15] Menezes, A.J., Vanstone, S.A., & Van Oorschot, P.C. (1996) *Handbook of Applied Cryptography*, CRC Press, ISBN 0-8493-8523-7, October 1996 (Fifth printing, August 2001).

[16] Cryptography Key, Retrieved August 06 2012, from http://en.wikipedia.org/wiki/Key_(cryptography)

[17] Diffie, W., & Hellman, M. (1976). Multi-user cryptographic techniques. In *Proceedings of the AFIPS* Proceedings 45, June 8 1976, pp.109-112.

[18] Kahn, D. (1979). *Cryptology Goes Public*, 58 Foreign Affairs 141, 151 (fall 1979), pp.153.

[19] Diffie, W., & Hellman, M. (1976). New directions in cryptography, *IEEE Trans. Inform. Theory, 22*(6), pp.644-654.

[20] Rivest, R., Shamir, A., & Adleman, L. (1978). A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM, 21*(2), pp.120-126. Previously released as an *MIT Technical Memo* in April 1977, and published in *Martin Gardner's Scientific American Mathematical* recreations column.

[21] "Security Policy and Key Management: Centrally Manage Encryption Key". Retrieved August 13 2013, from Slideshare.net.

[22] "Key Management System". Retrieved January 17 2014, from Bell ID.

[23] Key generation. Retrieved December 10 2011, from http://en.wikipedia.org/wiki/ Key_generation

[24] Key-agreement protocol. Retrieved December 12 2011, from http://en.wikipedia.org/wiki/ Key-agreement_protocol

[25] Key exchange. Retrieved December 12 2011, from http://en.wikipedia.org/wiki/ Key_exchange

[26] Attack model. Retrieved December 15 2011, from http://en.wikipedia.org/wiki/Attack_model

[27] Man-In-The-Middle Attack Retrieved December 15 2011, from http://en.wikipedia.org/wiki/ Man-in-the-middle_attack

[28] Zadeh, Lotfi, A. (1994). Fuzzy Logic, Neural Networks, and Soft Computing, *Communication of the ACM, 37*(3), pp.77-84.

[29] Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford Univ. Press.

[30] Bäck, T., Fogel, D., & Michalewicz, Z. (1997). *Handbook of Evolutionary Computation*, Oxford Univ. Press.

[31] Banzhaf, W., Nordin, P., Keller, R., & Francone, F. (1998). *Genetic Programming - An Introduction*. Morgan Kaufmann, San Francisco.

[32] Eiben, A.E., Smith, J.E. (2003). *Introduction to Evolutionary Computing*, Springer.

[33] Ashlock, D. (2006). *Evolutionary Computation for Modeling and Optimization*, Springer, ISBN 0-387-22196-4.

[34] Spillman, R., Janssen, M., Nelson, B., & Kepner, M. (1993). Use of Genetic Algorithms in the Cryptanalysis of Simple Substitution Ciphers. *Cryptologia, XVII*(1), pp.31-43.

[35] Spillman, R. (1993). Cryptanalysis of Knapsack Ciphers Using Genetic Algorithms. *Cryptologia, XVII*(4), pp.367-377.

[36] Clark, J.A. (2003). Nature-Inspired Cryptography: Past, Present and Future. In *Proceedings of Conference on Evolutionary Computation*, December 8-12, 2003, Canberra, Australia.

[37] Kennedy, J., & Eberhart, R. C. (1995). Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, Piscataway, NJ.

[38] Kennedy, J. (1999). Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC 1999),* Piscataway, NJ. pp.1931-1938.

[39] Colorni, A., Dorigo et, M., Maniezzo, V. (1991). Distributed Optimization by Ant Colonies. In *Proceedings of actes de la première conférence européenne sur la vie artificielle*, Paris, France, pp.134-142, Elsevier Publishing.

[40] Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms*, (Doctoral dissertation), Politecnico di Milano, Italy.

[41] Bafghi, A.G., & Sadeghiyan, B. (2003). Differential Model of Block Cipher with Ant Colony Technique. In *Proceedings of Workshops on Coding, Cryptography and Combinatorics*, Yellow Mountain.

[42] Kirkpatrick, S., Gelatt Jr, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science, 220*(4598), 671-680, Bibcode:1983Sci...220..671K. DOI: 10.1126/science.220.4598.671. JSTOR 1690046. PMID 17813860.

[43] Černý, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45, 41-51, DOI: 10.1007/BF00940812.

[44] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics, 21* (6), 1087, Bibcode:1953JChPh..21.1087M. DOI: 10.1063/1.1699114.

[45] Bagnall, A.J. (1996). *The Applications of Genetic Algorithms in Cryptanalysis*. (M.Sc. Thesis), School of Information System, University of East Anglia.

[46] Clark, A. (1998). *Optimization Heuristics for Cryptology*. (Doctoral dissertation), Faculty of Information Technology, Queensland University of Technology, Australia.

[47] Glover, F. (1986). Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research, 13*(5), 533-549, DOI: 10.1016/0305-0548(86)90048-1.

[48] Glover, F. (1989). Tabu Search - Part 1. *ORSA Journal on Computing, 1*(2), 190-206, DOI: 10.1287/ijoc.1.3.190.

[49] Glover, F. (1990). "Tabu Search - Part 2". *ORSA Journal on Computing, 2*(1), 4-32. DOI: 10.1287/ijoc.2.1.4.

[50] Simon, H. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

[51] Hassoun, M. (1995). *Fundamentals of Artificial Neural Networks.* MIT Press.

[52] Osowski, S. (1996). *Neural Networks in Algorithmic Approach.* WNT (in Polish).

[53] Bornholdt, S., & Schuster, H. (2003). *Theory of Interacting Neural Networks*, Wiley VCH.

[54] Rutkowski, L. (2006). *Methods and Techniques of Artificial Intelligence*. PWN, Warsaw (in Polish).

[55] Saad, D. (1998). *On-line learning in neural networks*. Cambridge University Press, Cambridge.

[56] Kanter, I., & Kinzel, W. (2002). *Neural Cryptography*, cond-mat/0208453.

[57] Kinzel, W. & Kanter, I. (2002). Neural cryptography. In *Proceedings of 9th International Conference on Neural Information Processing, ICONIP '02*, November 18-22 2002, Vol. 3, pp.1351-1354.

[58] Huntsberger, D.V. (1961). *Elements of Statistical Inference*. Allyn and Bacon.

[59] Spiegel, M. R., & Stephens, L. J. (2008). *Theory and Problems of Statistics*, 3rd edition, Tata McGraw-Hill Publishing Company Limited, Thirteenth reprint.

[60] Maurer, U. (1993). Secret key agreement by public discussion from common information. *IEEE Trans. Inform. Theory, 39*(3), 733-742.

[61] Delgado-Restituto, M., de Ahumada, R.L., & Rodriguez-Vazquez, A. (1995). Secure communication through switched-current chaotic circuits, *IEEE International Symposium on Circuits and Systems, ISCAS '95., Vol.3*, 2237-2240, Apr 30-May 3 1995.

[62] Dourlens, S. (1995). *Neuro-Cryptography*. (MSc Thesis), Dept. of Microcomputers and Microelectronics, University of Paris, France.

[63] Caponetto, R., Lavorgna, M., & Occhipinti, L. (1996). Cellular neural networks in secure transmission applications. In *Proceedings of IEEE Fourth International Workshop on Cellular Neural Networks and their Applications, CNNA-96*, June 24-26 1996, pp.411 – 416.

[64] Metzler, R., Kinzel, W., & Kanter, I. (2000). Interacting neural networks. *Phys. Rev. E 62*(3), 2555-2565.

[65] Kinzel, W., Metzler, R., & Kanter, I. (2000). Dynamics of interacting neural networks. *J. Phys. A: Math. Gen. 33*(14), L141-L147.

[66] Kanter, I., Kinzel, W., & Kanter, E. (2002). Secure Exchange of Information by Synchronization of Neural Networks. *Europhys. Lett. 57*(1), 141-147.

[67] Rosen-Zvi, M., Kanter, I., & Kinzel, W. (2002). *Cryptography based on Neural Networks: Analytical Results*. cond- mat/0202350.

[68] Klimov, A., Mityagin, A., & Shamir, A. (2002). Analysis of Neural Cryptography. In Zheng, Y. (Ed.), *ASIACRYPT 2002, (LNCS)*, (Vol. 2501, pp. 288-298) Heidelberg, Germany: Springer.

[69] Mislovaty, R., Perchenok, Y., Kanter, I., & Kinzel, W. (2002). Secure key exchange protocol with an absence of injective functions. *Phys. Rev. E 66*.

[70] Rosen-Zvi, M., Klein, E., Kanter, I., & Kinzel, W. (2002). Mutual learning in a tree parity machine and its application to cryptography. *Phys. Rev. E 66*(6), 066 135-1–066 135-13.

[71] Kinzel, W., & Kanter, I. (2002). *Interacting neural networks and cryptography*. *Advances in Solid State Physics*.

[72] Kanter, I., & Kinzel, W. (2003). The Theory of Neural Networks and Cryptography. In *Proceeding of the XXII Solvay Conference on Physics*, The Physics of Communication, 2003, 631-644.

[73] Kinzel, W., & Kanter, I. (2003). Disorder generated by interacting neural networks: application to econophysics and cryptography. *J. Phys. A: Math. Gen. 36*(43), 11 173–11 186.

[74] Mislovaty, R., Klein, E., Kanter, I., & Kinzel, W. (2003). Public Channel Cryptography by Synchronization of Neural Networks and Chaotic Maps. *Phys. Rev. Lett. 91, 118701*.

[75] Shacham, L.N., Klein, E., Mislovaty, R., Kanter, I., & Kinzel, W. (2004). Cooperating attackers in neural cryptography. *Phys. Rev. E 69*(6), 066 137-1–066 137-4.

[76] Mislovaty, R., Klein, E., Kanter, I., & Kinzel, W. (2004). Security of neural cryptography. In *Proceeding of the IEEE 11th International Conference on Electronics, Circuits and Systems*, ICECS 2004, December 13-15 2004, pp. 219-221.

[77] Ruttor, A., Reents, G., & Kinzel, W. (2004). Synchronization of random walks with reflecting boundaries. *J. Phys. A: Math. Gen. 37*, 8609-8618.

[78] Ruttor, A., Kinzel, W., Shacham, L., & Kanter, I. (2004). Neural cryptography with feedback. *Phys. Rev. E 69*(4), 046 110-1–046 110-7.

[79] Volkmer, M., & Schaumburg, A. (2004). Authenticated tree parity machine key exchange. *arXiv preprint* cs/0408046.

[80] Volkmer, M., & Wallner, S. (2004). A Low-Cost Solution for Frequent Symmetric Key Exchange in Ad-hoc Networks. In *Proceedings of the 2nd German Workshop on Mobile Ad-hoc Networks (WMAN)*, 2004, pp. 128–137.

[81] Ruttor, A., Kinzel, W., & Kanter, I. (2005). Neural cryptography with queries. *J. Stat. Mech. 2005*(1), pp. 1-14.

[82] Klein, E., Mislovaty, R., Kanter, I., Ruttor, A., & Kinzel, W. (2005). Synchronization of Neural Networks by Mutual Learning and its Application to Cryptography. In *Advances in Neural Information Processing Systems*, MIT Press, Cambridge, 2005, Vol. 17, pp. 689-696.

[83] Kanter, I. (2005). The theory of neural networks: learning from examples, time-series and cryptography. In *Proceedings of the IEEE International Workshop on VLSI Design and Video Technology*, May 28-30 2005, pp. xviii.

[84] Kotlarz, P., & Kotulski, Z. (2005). On Application of Neural Networks for S-Boxes Design. In Szczepaniak, P.S., Kacprzyk, J., Niewiadomski, A. (Eds.), *AWIC 2005, LNCS (LNAI)*, (Vol. 3528, pp. 243-248), Heidelberg, Germany: Springer.

[85] Volkmer, M., & Wallner, S. (2005a). Tree parity machine rekeying architectures. *IEEE Trans. Comput. 54*(4), pp. 421-427.

[86] Volkmer, M., & Wallner, S. (2005b). A Key Establishment IP-Core for Ubiquitous Computing. In *Proceedings of the 1st Int. Workshop on Secure and Ubiquitous Networks*, SUN 2005, IEEE Computer Society, Los Alamitos, Denmark, pp. 241-245.

[87] Volkmer, M., & Wallner, S. (2005c). Lightweight Key Exchange and Stream Cipher based solely on Tree Parity Machines. In *ECRYPT Workshop on RFID and Lightweight Crypto*, Graz University of Technology, Austria, July 2005, pp. 102-113.

[88] Volkmer, M., & Wallner, S. (2005d). Tree parity machine rekeying architectures for embedded security. *Cryptology ePrint Archive*, Inst. Comput. Technol., Hamburg Univ. Technol., Hamburg, Germany, Rep. 2005/235.

[89] Batina, L., Lano, J., Mentens, N., Ors, S.B., Preneel, B. & Verbauwhede, I. (2005). *Energy, performance, area versus security tradeoffs for streamciphers*. Catholic University Leuven.

[90] Chen, T. & Cai, J. (2005). A Novel Remote User Authentication Scheme Using Interacting Neural Network. *Advances in Natural Computation, (LNCS)*, (Vol. 3610, pp. 1117-1120), Heidelberg, Germany: Springer.

[91] Chen, T., Chen, B., & Cai., J. (2005). A Novel Identity-Based Key Issuing Scheme Based on Interacting Neural Network. *Advances in Neural Networks, (LNCS)*, (Vol. 3497, pp. 637-642), Heidelberg, Germany: Springer.

[92] Gross, N., Klein, E., Rosenbluh, M., Kinzel, W., Khaykovich, L., & Kanter, I. (2005). A framework for public-channel cryptography using chaotic lasers. *Phys. Rev. E 73*(6), 066 214-1–066 214-4.

[93] Klein, E., Kanter, R.M.I., & Kinzel, W. (2005). Public-channel cryptography using chaos synchronization. *Phys. Rev. E 72*(1), 016 214-1–016 214-4.

[94] Castro, J.C.H., & Viñuela, P. I. (2005). Evolutionary Computation in computer security and cryptography. *New Generation Computing*, *23*(3), 193-199.

[95] Godhavari, T., Alamelu, N.R., & Soundararajan, R. (2005). Cryptography Using Neural Network. *INDICON, 2005 Annual IEEE*, December 11-13 2005, pp. 258-261.

[96] Klein, E., Gross, N., Rosenbluh, M., Kinzel, W., Khaykovich, L., & Kanter, I. (2006). Stable isochronal synchronization of mutually coupled chaotic lasers. *Phys. Rev. E 73*(6), 066 214-1–066 214-4.

[97] Ruttor, A., Kinzel, W., Naeh, R., & Kanter, I. (2006). Genetic attack on neural cryptography. *Phys. Rev. E 73*(3), 036 121-1–036 121-8.

[98] Ruttor, A. (2006). *Neural Synchronization and Cryptography*. (Doctoral dissertation), Würzburg.

[99] Ruttor, A., Kinzel, W., & Kanter, I. (2007) Dynamics of neural cryptography, *Phys. Rev. E 75*(5), 056 104-1–056 104-4.

[100] Mu□ehlbach, S., & Wallner, S. (2007). Secure and Authenticated Communication in Chip- Level Microcomputer Bus Systems with Tree Parity Machines. In *Proceedings of the IEEE IC-SAMOS*, Greece, pp. 201-208.

[101] Saballus, B., Volkmer, M. & Wallner, S. (2007). Secure Group Communication in Ad-Hoc Networks using Tree Parity Machine. In *Proceedings of the Communication in Distributed Systems (KiVS)*, ITG-GI Conference, February 26-March 2 2007, pp. 1-12.

[102] Patra, G. K., Anil Kumar V., Thangavelu, R. P. (2007). A New Concept of Key Agreement Using Chaos-Synchronization Based Parameter Estimation, Information Systems Security. *(LNCS)*, (Vol. 4812, pp. 263-266), Heidelberg, Germany: Springer.

[103] Laskari, E. C., Meletiou, G. C., Stamatiou, Y. C., & Vrahatis, M. N. (2007). Cryptography and Cryptanalysis Through Computational Intelligence. *Computational Intelligence in Information Assurance and Security Studies in Computational Intelligence*, *57*, 1-49.

[104] Arvandi, Maryam, Alireza, & Sadeghian (2007). Chosen Plaintext Attack against Neural Network-Based Symmetric Cipher. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN).*

[105] Liu, Niansheng, & Donghui, G. (2007). Security analysis of public-key encryption scheme based on neural networks and its implementing. *Computational Intelligence and Security*, 443-450, Berlin Heidelberg: Springer.

[106] Hen, Tieming, & Rongrong, J. (2007). Designing Security Protocols Using Novel Neural Network Model. In *Proceedings of the IEEE Third International Conference on Natural Computation (ICNC),* Vol. 1.

[107] Li, P. & Yi, Z. (2007). Compound Attack on Synchronization Based Neural Cryptography, In *Proceedings of the Advances in Cognitive Neurodynamics ICCN 2007*, pp. 1019-1023.

[108] Chen, T., Huang, S.H. (2008). Tree Parity Machine-based One-time Password Authentication Schemes. In *Proceedings of the International Joint Conference on Neural Networks*, Hong Kong, June 1-6 2008.

[109] Dong, H., & Yu Yan W. (2008a). Secure Authentication on WiMAX with Neural Cryptography. In *Proceedings of the International Conference on Information Security and Assurance (ISA)*, April 24-26 2008, pp. 366–369.

[110] Dong, H., & Yu Yan W. (2008b). Security Research on WiMAX with Neural Cryptography, In *Proceedings of the International Conference on Information Security and Assurance (ISA)*, April 24-26 2008, pp. 370 – 373.

[111] Dong, H., Hansheng, L. (2008). Privacy Research on Ubicomp Computing with Neural Cryptography, In *Proceedings of the The 3rd International Conference on Grid and Pervasive Computing Workshops*, GPCWorkshops '08, May 25-28 2008, pp. 335-340.

[112] Yunpeng, Z., Tongtong, X., Zhengjun, Z., Chunyan, M., & Xiaobin, C. (2008). The Improvement of Public Key Cryptography Based on Chaotic Neural Networks, In *Proceedings of the Eighth International Conference on Intelligent Systems Design and Applications (ISDA '08), Vol. 3*, November 26-28 2008, pp. 326 - 330.

[113] Shouhong, W., Hai, W. (2008). Password Authentication Using Hopfield Neural Networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 38*(2), pp. 265-268.

[114] Tieming, C., Hangzhou, & Huang, S.H. (2008). Tree parity machine-based One-Time Password authentication schemes. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN 2008)*, June 1-8 2008, pp. 257-261.

[115] Arvandi, M., Wu, S., & Sadeghian, A. (2008). On the use of recurrent neural networks to design symmetric ciphers, *Computational Intelligence Magazine,* IEEE 3.2.

[116] Dong, H., & Zhong, S. (2009). A New Service-Based Computing Security Model with Neural Cryptography, In *Proceedings of the Second Pacific-Asia Conference on Web Mining and Web-based Application (WMWA '09)*, June 6-7 2009, pp. 154-156.

[117] Reyes, O.M., Kopitzke, I., & Zimmermann, K. H. (2009). Permutation parity machines for neural synchronization. *Phys. A: Math. Gen. 42*, 195002.

[118] Reyes, O.M., & Zimmermann, K. H. (2009). Permutation parity machines for neural cryptography. *J. Phys. A: Math. Theor. 42*, 195 002-1–195 002-20.

[119] Schmitzer, B., Kinzel, W., & Kanter, I. (2009). Pulses of chaos synchronization in coupled map chains with delayed transmission. *Phys. Rev. E 80*, 047203.

[120] Wallner S. (2009). Designing Low-Cost Cryptographic Hardware for Wired- or Wireless Point-to-Point Connections. *Advances in Information Security and Its Application Communications in Computer and Information Science, 36*, pp. 1-10.

[121] Lian, Shiguo, Jinsheng, S., & Zhiquan, W. (2009). One-way hash function based on neural network. *arXiv preprint arXiv:* 0707.4032.

[122] Allam, A.M., & Abbas, H. M. (2010). On the improvement of neural cryptography using erroneous transmitted information with error prediction. *IEEE Trans. Neural Networks 21*(12), pp. 1915-1924.

[123] Ahmad, S., Beg, M. R., Abbas, Q., Ahmad, J. & Atif, S. M. (2010). Comparative Study between Stream Cipher and Block Cipher using RC4 and Hill Cipher. *International Journal of Computer Applications (IJCA), 1*(25), 9-12, February 2010.

[124] Revankar, P., Gandhare, W.Z., & Rathod, D. (2010). Neural Synchronization with Queries. In *Proceedings of the International Conference on Signal Acquisition and Processing (ICSAP '10)*, February 9-10 2010, pp. 371- 374.

[125] Tirdad, K., & Sadeghian, A. (2010). Hopfield neural networks as pseudo random number generators. In *Proceedings of the Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS)*, July 12-14 2010, pp. 1 - 6.

[126] Prabakaran, N., & Vivekanandan, P. (2010). A New Security on Neural Cryptography with Queries. *International Journal of Advanced Networking and Applications, 02*(01), pp. 437-444.

[127] Prabakaran, N., & Nallaperumal, E. "(2010). Neural cryptography with queries for co-operating attackers and effective number of keys. In *Proceedings of the IEEE International Conference on Communication Control and Computing Technologies (ICCCCT)*, October 7-9 2010, pp. 782-787.

[128] Chowdhury, R., & Ghosh, S. (2011). Study of Cryptology Based on Proposed Concept of Cyclic Cryptography using Cyclograph. *Research Journal of Engineering and Technology (RJET), 0*2(01), January-March, 2011.

[129] Bhattacharya, T., Hore, S., Mukherjee, A., & Chaudhuri, S. R. B. (2011). A Novel Data Encryption Technique by Genetic Crossover of Robust Biometric Key and Session Based

Password. *International Journal of Network Security & Its Applications (IJNSA), 3*(2), 111-120, March 2011.

[130] Yosh, H. (2011). The Key Exchange Cryptosystem used with Higher Order Diophantine Equations. *International Journal of Network Security & Its Applications (IJNSA), 3*(2), 43-50, March 2011.

[131] Jogdand, R. M., & Sahana, S. (2011). Design of An Efficient Neural Key Generation. *International Journal of Artificial Intelligence & Applications (IJAIA), 2*(1), pp 60-69.

[132] Allam, A.M., & Abbas, H.M. (2011). Group key exchange using neural cryptography with binary trees. *In Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, May 8-11 2011, pp. 000783-000786.

[133] Saeed, F., Qadir, A. B. A., Mughal, Y. M., & Rashid, M. (2011). A Novel Key Generation for FMET. *International Journal of Computer Science and Network Security (IJCSNS), 11*(6), 197-202, June 2011.

[134] Karas, D.S., Karagiannidis, G.K., & Schober, R. (2011). Neural network based PHY-layer key exchange for wireless communications. *In Proceedings of the IEEE 22nd International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*, September 11-14 2011, pp. 1233-1238.

[135] Li, Yantao, et al. (2011). Parallel Hash function construction based on chaotic maps with changeable parameters. *Neural Computing and Applications, 20*(8), pp. 1305-1312, Springer.

[136] Lu´ıs, F., Seoane, L.F., & Ruttor, A. (2011). Successful attack on PPM-based neural cryptography. *arXiv preprint arXiv*:1111.5792,24.

[137] Rasool, S., Sridhar, G., Kumar, K. H., & Kumar, P. R. (2011). Enhanced Secure Algorithm for Message Communication. *International Journal of Network Security & Its Applications (IJNSA), 3*(5), 33-42, September 2011.

[138] Qian, W., Chen, H., Li, Z., & Jia, C. (2011). On A Practical Distributed Key Generation Scheme Based on Bivariate Polynomials. In *proceedings of the 7th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM),* September 23-25 2011, Wuhan, China, pp. 01-04.

[139] Gajbhiye, S., Sharma, M., & Dashputre, S. (2011). A Survey Report on Elliptic Curve Cryptography. *International Journal of Electrical and Computer Engineering (IJECE), 1*(2), 195-201, December 2011.

[140] Vijayakumar, P., Vijayalakshmi, V., & Zayaraz, G. (2011). DNA Computing based Elliptic Curve Cryptography. *International Journal of Computer Applications (IJCA), 6*(4), 18-21, December 2011.

[141] Som, S., Chatergee, N. S., & Mandal, J. K. (2011). Key Based Bit Level Genetic Cryptographic Technique (KBGCT). In *proceedings of the 7th International Conference on Information Assurance and Security (IAS),* December 5-8 2011, Melaka, Malaysia, pp. 240-245.

[142] Das, D., Mukherjee, M., Choudhary, N., Nath, A., & Nath, J. (2011). An Integrated Symmetric Key Cryptography Algorithm using Generalised Modified Vernam Cipher Method and DJSA Method: DJMNA Symmetric Key Algorithm. In *proceedings of the World Congress on Information and Communication Technologies (WICT 2011),* December 11-14 2011, Mumbai, India, pp. 1199-1204.

[143] Patheja, P. S., Waoo, A. A., & Nagwanshi, S. (2011). A Hybrid Encryption Technique to Secure Bluetooth Communication. *IJCA Proceedings on International Conference on Computer Communication and Networks CSI-COMNET-2011, 1*, 26-32.

[144] Kumar, R., & Saraswat, V. K. (2011). A Research on Performance Evaluation for Some Cipher Models Based on Analytical Survey. *International Journal of Networks and Communications, 1*(1), 14-17.

[145] Gupta, V., Singh, G., & Gupta, R. (2012). Advance Cryptography Algorithm for Improving Data Security. *International Journal of Advanced research in Computer Science and Software Engineering, 2*(1), January 2012.

[146] Rayarikar, R., Upadhyay, S., & Shah, D. (2012). An Encryption Algorithm for Secure Data Transmission. *International Journal of Computer Applications (IJCA), 40*(7), 43-47. February 2012.

[147] Kaul, V., Narayankhedkar, S. K., Achrekar, S., & Agarwal, S. (2012). Security Enhancement Algorithms for Data Transmission for Next Generation Networks. In *proceedings of the International Conference and workshop on Emerging Trends in Technology (ICWET 2012),* March 2012, icwet2012 - Number 13, pp. 1-6.

[148] Nicanfar, H., & Leung, V. C. M. (2012). EIBC: Enhanced Identity-Based Cryptography, a Conceptual Design. In *proceedings of the IEEE International Systems Conference (SysCon),* March 19-22 2012, Vancouver, BC, Canada, pp. 1-7.

[149] Alani, M. M. (2012). Neuro-cryptanalysis of DES. In *proceedings of the World Congress on Internet Security (WorldCIS 2012),* June 10-12 2012, Guelph, Canada, pp. 23-27.

[150] Savari, M., & Montazerolzohour, M. (2012). All about Encryption in Smart Card. In *proceedings of the International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec 2012),* June 26-28 2012, Kuala Lumpur, Malaysia, pp. 54-59.

[151] Banerjee, S. & Ariffin, M. R. K. (2012). Chaos Synchronization Based Data Transmission with Asymmetric Encryption. *International Journal of Computer Applications (IJCA), 37*(12), pp. 6-9.

[152] Seoane, L.F., & Ruttor, A. (2012). Successful attack on permutation-parity-machine-based neural cryptography. *Phys. Rev. E 85*(2), 025 101-1–025 101-4.

[153] Urbanovich, Pavel, Marcin Plonkowski, & Konstantsin Churikov. (2012). The appearance of conflict when using the chaos function for calculating the hash code. *network 3*.

[154] Santhanalakshmi, S., Sudarshan, T., Patra, S. B., & Gopal, K. (2012). Neural Synchronization by Mutual Learning Using Genetic Approach for Secure Key Generation. *Recent Trends in Computer Networks and Distributed Systems Security,, Communications in Computer and Information Science, 335*, pp. 422-431.

[155] Winkler, M., Butsch, S., & Kinzel, W. (2012). Pulsed chaos synchronization in networks with adaptive couplings. *Phys. Rev. E 86*, 016203.

[156] Dolecki, M. (2012). Tree Parity Machine Synchronization Time-Statistical Analysis. *Mathematics, Physics and Informatics Series, Minsk 6*(153), pp. 149-151.

[157] Qing-hai, B., Wen-bo, Z., Peng, J., & Xu, L. (2012). Research on Design Principles of Elliptic Curve Public Key Cryptography and Its Implementation. In *proceedings of the International Conference on Computer Science & Service System (CSSS 2012),* August 11-13 2012, Nanjing, China, pp. 1224-1227.

[158] Abdulkader, H. & Roviras, D. (2012). Generating Cryptography Keys using Self-Organizing Maps. In *Proceedings of the International Symposium on Wireless Communication Systems (ISWCS 2012)*, August 28-31 2012, Paris, France, pp. 736-740.

[159] Santhanalakshmi, S., Sangeeta, K., & Patra, G.K. (2012). Design of Stream Cipher for Text Encryption using Soft Computing based Techniques. *International Journal of Computer Science and Network Security (IJCSNS*), *12*(12), pp. 149-152.

[160] Gutub, A.A.A., & Khan F.A.A. (2012). Hybrid Crypto Hardware Utilizing Symmetric-Key and Public-Key Cryptosystems. In *Proceedings of the International Conference on Advanced Computer Science Applications and Technologies (ACSAT 2012)*, November 26-28 2012, Kuala Lumpur, Malaysia, pp. 116-121.

[161] Shrivastava, A., & Singh, M. (2012). A Security Enhancement Approach in Quantum Cryptography. In *Proceedings of the 5th International Conference on Computers and Devices for Communication (CODEC 2012),* December 17-19 2012, Kolkata, India, pp. 1-4.

[162] Pramanik, S., & Setua, S. K. (2012). DNA Cryptography, In *Proceedings of the 7th International Conference on Electrical & Computer Engineering (ICECE 2012),* December 20-22 2012, Dhaka, Bangladesh, pp. 551-554.

[163] Mandal, S., Macdonald, G., Rifai, M. E., Punekar, N., Zamani, F., Chen, Y., Kak, S., Verma, P. K., Huck, R. C., & Sluss, J. (2013). Multi-Photon Implementation of Three-

Stage Quantum Cryptography Protocol. In *Proceedings of the International Conference on Information Networking (ICOIN 2013)*, January 28-30 2013, Bangkok, Thailand, pp. 6-11.

[164] Verma, H. K., & Singh, R. K. (2013). Enhancement of RC6 Block Cipher Algorithm and Comparison with RC5 & RC6. In *Proceedings of the 3rd IEEE International Advance Computing Conference (IACC 2013)*, February 22-23 2013, Ghaziabad, Uttar Pradesh, India, pp. 556-561.

[165] Yang, W. C., & Lee, J. X. (2013). Implementation of Stream Cipher Service in JCA. In *Proceedings of the IEEE International Symposium on Next-Generation Electronics (ISNE 201)*, February 25-26 2013, Kaohsiung, Taiwan, pp. 557-561.

[166] Dolecki, M., Kozera, R., & Lenik, K. (2013). The Evaluation of the TPM Synchronization on the Basis of their Outputs. *Journal of Achievements in Materials and Manufacturing Engineering 57*(2), pp.91-98.

[167] Dolecki, M., & Kozera, R. (2013). Threshold Method of Detecting Long-Time TPM Synchronization, *Computer Information Systems and Industrial Management, (LNCS)* (Vol. 8104, pp. 241-252), Heidelberg, Germany: Springer.

[168] Mu, N., & Liao, X. (2013). An Approach for Designing Neural Cryptography. *Advances in Neural Networks ISNN 2013, (LNCS)* (Vol. 7951, pp. 99-108), Heidelberg, Germany: Springer.

[169] Aguilar, J., & Molina, C. (2013). The Multilayer Random Neural Network, *Neural Processing Letters, 37*(2), pp. 111-133.

[170] Lei, X., Liao, X., Chen, F., & Huang, T. (2013). Two-layer tree-connected feed-forward neural network model for neural cryptography, *Phys. Rev. E 87*, 032811.

[171] Ramesh, A., & Suruliandi, A. (2013). Performance Analysis of Encryption Algorithms for Information Security. In *Proceedings of the International Conference on Circuits, Power and Computing Technologies (ICCPCT 2013)*, March 20-21 2013, Nagercoil , Tamil Nadu, India, pp. 840-844.

[172] Sircar, R. D., Sekhon, G., & Nath, A. (2013). Modern Encryption Standard (MES): Version-II. In *Proceedings of the International Conference on Communication Systems and Network Technologies (CSNT 2013)*, April 6-8 2013, Gwalior, India, pp. 506-511.

[173] Soni, R., Johar, A., & Soni, V. (2013). An Encryption and Decryption Algorithm for Image Based on DNA. In *Proceedings of the International Conference on Communication Systems and Network Technologies (CSNT 2013)*, April 6-8 2013, Gwalior, India, pp. 478-481.

[174] Mandal, B. K., Bhattacharyya, D., & Bandyopadhyay, S. K. (2013). Designing and Performance Analysis of a Proposed Symmetric Cryptography Algorithm. In

Proceedings of the International Conference on Communication Systems and Network Technologies (CSNT 2013), April 6-8 2013, Gwalior, India, pp. 453-461.

[175] Naveen J. K., Karthigaikumar P., Sivamangai N. M., Sandhya R. & Asok S. B. (2013). Hardware Implementation of DNA Based Cryptography, In *Proceedings of the IEEE Conference on Information & Communication Technologies (ICT 2013),* April 11-12 2013, JeJu Island, South Korea, pp. 696-700.

[176] Mu, N., Liao, X., & Huang, T. (2013). Approach to design neural cryptography: A generalized architecture and a heuristic rule. *Phys. Rev. E 87,* 062804.

[177] Allam, A.M., Abbas, H.M., & El-Kharashi, M.W. (2013). Authenticated key exchange protocol using neural cryptography with secret boundaries, In *Proceedings of the IEEE The 2013 International Joint Conference on Neural Networks (IJCNN),* August 4-9 2013, pp. 1-8.

[178] Jhajharia, S., Mishra, S., & Bali, S. (2013). Public key cryptography using neural networks and genetic algorithms. In *Proceedings of the IEEE Sixth International Conference on Contemporary Computing (IC3),* August 8-10 2013, pp. 137-142.

[179] Allam, A.M., Abbas, H.M., & El-Kharashi, M.W. (2013). Security analysis of neural cryptography implementation. In *Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM),* August 27-29 2013, pp. 195 -199.

[180] Akhavan, A., Azman, S., & Afshin, A. (2013). A novel parallel hash function based on 3D chaotic map. *Springer EURASIP Journal on Advances in Signal Processing*, pp. 1-12.

[181] Singh, A., & Mandal, A. (2013). Neural Cryptography for Secret Key Exchange and Encryption with AES. *International Journal of Advanced Research in Computer Science and Software Engineering, 3*(5).

[182] Adel, A., El-Zoghabi, A. H., & Yassin, H. H. H. (2013). Survey Report on Cryptography Based on Neural Network. *International Journal of Emerging Technology and Advanced Engineering, 3*(12), pp. 456-462.

[183] Lonkar, S., & Charniya, N. (2014). Neural Network based Cryptography. In *Proceedings of the International Technological Conference-2014 (I-TechCON),* January 03-04 2014, pp. 168-172.

[184] Apdullah, Y., & Yakup, K. (2014). Neural Network Based Cryptography. *Neural Network World*, *24,* pp. 177-192.

[185] Mohammed, A. (2014). Appliance of Neuron Networks in Cryptographic Systems. *Research Journal of Applied Sciences, Engineering and Technology 7*(4), pp. 740-744,

[186] Soni, V., & Tanwar, S. (2014). Secure Key Exchange using Neural Network. *International Journal of Engineering Sciences & Research Technology*, pp. 1707-1709.

[187] Dadhich, A., & Yadav, S. K. (2014). Evolutionary Algorithms, Fuzzy Logic and Artificial Immune Systems applied to Cryptography and Cryptanalysis: State-of-the-art review. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), 3*(6), pp. 2111-2120.

[188] Singla, P., Sachdeva, P., & Ahmad M. (2014). A Chaotic Neural Network Based Cryptographic Pseudo-Random Sequence Design. In *Proceedings of the IEEE Fourth International Conference on Advanced Computing & Communication Technologies (ACCT),* February 8-9 2014, pp. 301-306.

[189] Soni, V., Tanwar, M.S., & Prema, K.V. (2014). Implementation of Hash Function Based On Neural Cryptography. *International Journal of Computer Science and Mobile Computing, 3*(4), pp. 1380-1386.

[190] Chakraborty, S., Dalal, J., Sarkar, B., & Mukherjee, D. (2014). Neural synchronization based secret key exchange over public channels: A survey. In *Proceedings of the IEEE International Conference on Signal Propagation and Computer Technology (ICSPCT),* July 12-13 2014, pp. 368-375.

[191] Adel, A., El-Zoghabi, A. H., & Yassin, H. H. H. (2014). Public key Cryptography Based on Chaotic Neural Network. *International Journal of Computer Application, 4*(4) pp. 201-218.

[192] NIST statistical test. Retrieved June 27 2012, from http://csrc.nist.gov/groups/ST/toolkit/rng/ stats_tests.html.

[193] Webster, A. F. & Tavares, S. E. (1985). On the Design of S-Boxes, In *Proceedings of the Advances in Cryptology: Crypto'85,* August 18-22, 1985, Santa Barbara, California, USA, Vol. 218, pp. 523-534.

[194] Spiegel, M. R., & Stephens, L. J. (2008). *Theory and Problems of Statistics,* 3rd edition, Tata McGraw-Hill Publishing Company Limited, Thirteenth reprint 2008.

[195] Kohonen, T. (1982). Self-Organized Formation of Topologically Correct Feature Maps. *Biological Cybernetics 43* (1), 59-69. DOI: 10.1007/bf00337288.

[196] Sarkar, A., & Mandal, J. K. (2014). Soft Computing based Cryptographic Technique using Kohonen's Self-Organizing Map Synchronization for Wireless communication (KSOMSCT). *International Journal in Foundations of Computer Science & Technology (IJFCST), 4*(5), 85-100, DOI: 10.5121/ijfcst.2014.4508, ISSN 1839 – 7662.

[197] Sarkar, A., Mandal, J. K., & Patra P. (2013). Double Layer Perceptron Synchronized Computational Intelligence guided Fractal Triangle based Cryptographic Technique for Secured Communication (DLPFT). In *Proceedings of the IEEE 4th International Symposium on Electronic System Design (ISED-2013),* pp.191-195, ISBN 978-0-7695-5143-2, December 13-15 2013, NTU, Singapore, IEEE.

[198] Kohonen, T., & Honkela, T. (2007). Kohonen Network. *Scholarpedia*. Retrieved September 24 2012.

[199] Kohonen, T. Honkela, T. (2011). Kohonen network. *Scholarpedia*. Retrieved September 24 2012.

[200] Sarkar, A. (2013). Parallel Session Key Exchange and Certification by Fine Tuning of Double Layer Perceptron in Wireless communication (PKECDLP), *In Proceedings of the ICT in Present Wireless Revolution: Challenges and Issues*, The Institution of Electronics and Telecommunication Engineers Kolkata Centre, IETE Kolkata Center, Salt Lake, India, August 30-31 2013, pp. 1-9, ISBN 978-93-5126-699-0.

[201] Sarkar, A., & Mandal, J. K. (2013). Computational Intelligence Based Simulated Annealing Guided Key Generation In Wireless Communication (CISAKG). *International Journal on Information Theory (IJIT), 2*(4), 35-44, DOI: 10.5121/ijit.2014.2403, ISSN 2319 - 7609 [Online]; 2320 - 8465 [Print].

[202] Sarkar, A., & Mandal, J. K. (2014). Intelligent Soft Computing based Cryptographic Technique using Chaos Synchronization for Wireless Communication (CSCT). *International Journal of Ambient Systems and Applications (IJASA), 2*(3), 11-20, DOI: 10.5121/ijasa.2014.2302, ISSN 2320 – 9259 [Online]; 2321 – 6344 [Print].

[203] Sarkar, A., & Mandal, J. K. (2013). Genetic Algorithm Guided Key Generation in Wireless Communication (GAKG). *International Journal on Cybernetics & Informatics (IJCI), 2*(5), 9-17, DOI: 10.5121/ijci.2013.2502, ISSN 2277 - 548X [Online]; 2320 - 8430 [Print].

[204] Pecora, L. M., & Carroll, T. L. (1990). Synchronizationin chaotic systems. *Phys. Rev. Lett. 64*, 821-824.

[205] Lorenz, E. N. (1963). Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences 20*(2), 130-141, Bibcode:1963JAtS...20..130L. DOI: 10.1175/1520-0469.

[206] Sarkar, A., & Mandal, J. K. (2013). Computational Intelligence based Triple Layer Perceptron Model Coordinated PSO guided Metamorphosed based Application in Cryptographic Technique for Secured Communication (TLPPSO). In *Proceedings of the First International Conference on Computational Intelligence: Modeling, Techniques and Applications (CIMTA-2013)*, Vol.10, pp. 433-442, DOI: 10.1016/j. protcy. 2013.12.380, ISSN: 2212-0173, September 27-28 2013, Department of Computer Science & Engineering, University of Kalyani, Kalyani, India, Procedia Technology, Elsevier.

[207] Sarkar, A., & Mandal, J. K. (2014). Computational Science guided Soft Computing based Cryptographic Technique using Ant Colony Intelligence for Wireless Communication (ACICT). *International Journal of Computational Science and Applications (IJCSA), 4*(5), 61-73, DOI: 10.5121/ijcsa.2014.4505, ISSN 2200 – 0011.

[208] Sarkar, A., & Mandal, J. K. (2013). Complete Binary Tree Architecture based Triple Layer Perceptron Synchronized Group Session Key Exchange and Authentication in Wireless Communication (CBTLP). In *Proceedings of the 48th Annual Convention of CSI on theme "ICT and Critical Infrastructure*, AISC Series Springer Vol. 249, Book Subtitle Vol.2, pp. 609-615, DOI: 10.1007/978-3-319-03095-1_66, ISBN: 978-3-319-03094-4 [Print], ISBN: 978-3-319-03095-1 [Online], Series ISSN: 2194-5357, December 13-15 2013, Computer Society of India, Visakhapatnam Chapter, Visakhapatnam, India, Springer International Publishing.

[209] Sarkar, A., & Mandal, J. K. (2014). *Particle Swarm Optimization based Session Key Generation for Wireless Communication (PSOSKG).* In A. Bhattacharyya, S., & B. Dutta, P. (Eds.), *Handbook of Research on Swarm Intelligence in Engineering*, chapter 20, 701 E. Chocolate Ave., Hershey, Pennsylvania (USA): IGI GLOBAL. (Accepted)