Data Modeling and Analytics on Neural Computing and Cryptography For Wireless Communication

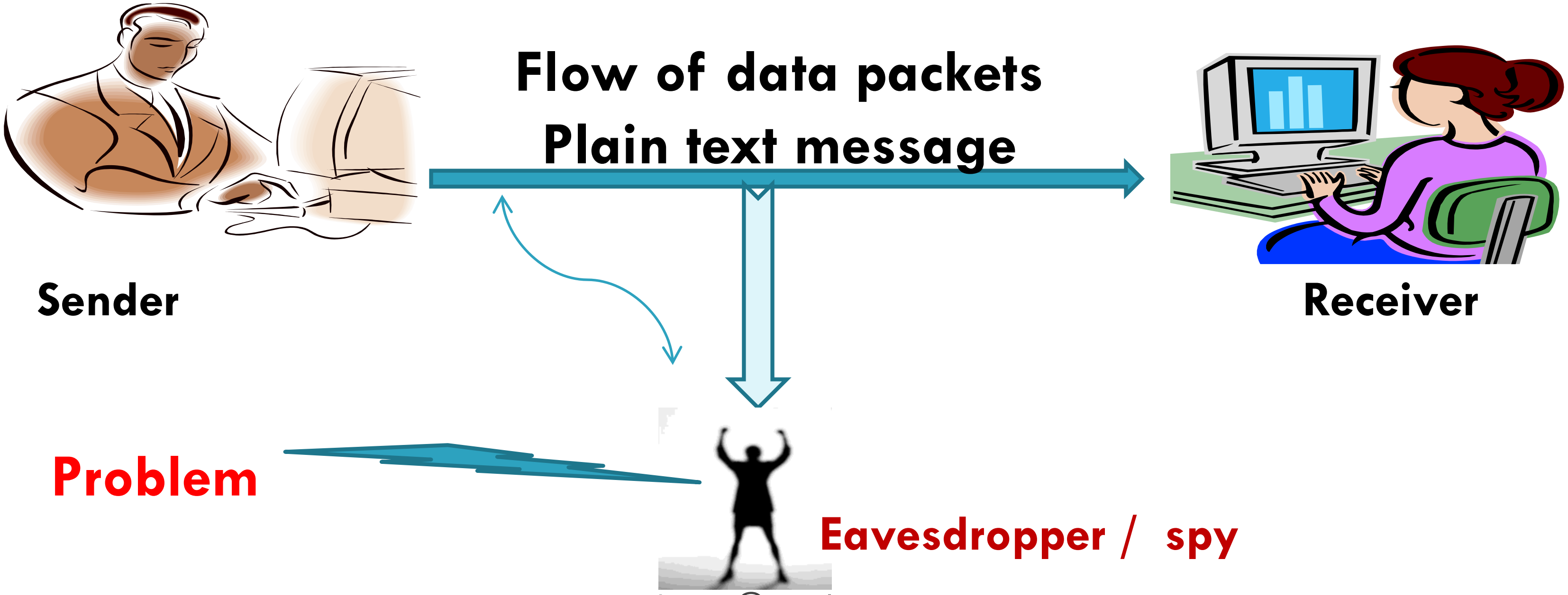
By

Prof. (Dr.) Jyotsna Kumar Mandal

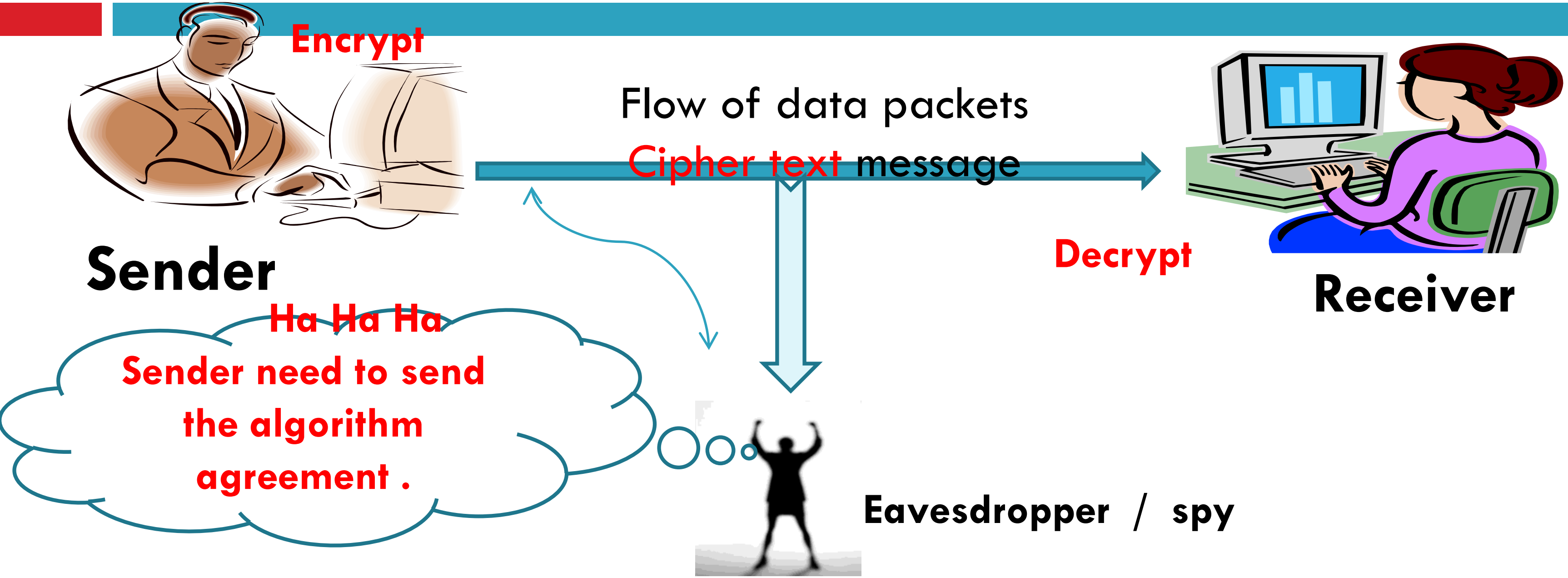
Department of Computer Science and Engineering
University of Kalyani,
Kalyani, West Bengal, India
jkm.cse@gmail.com

COMMUNICATION

Communication Through Network



Communication…….

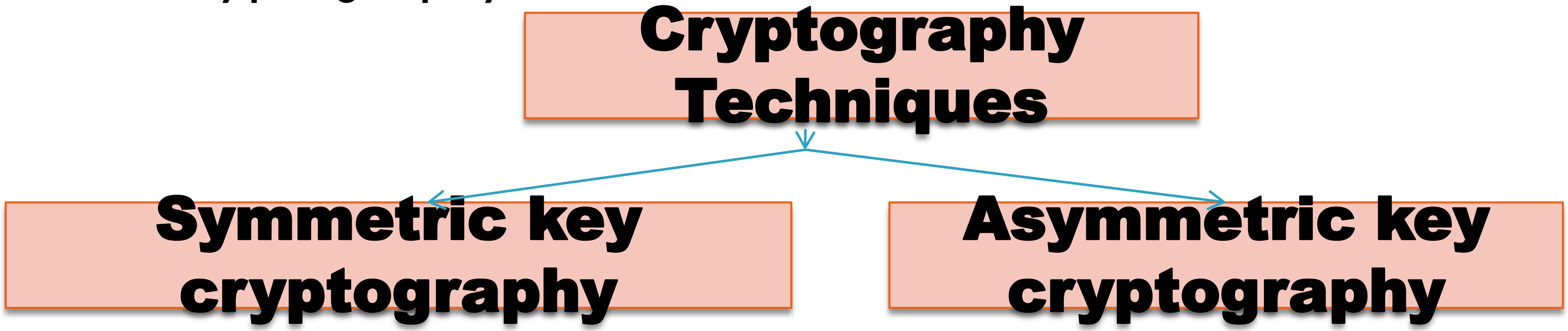


Note:- The decryption algorithm must be the same as the encryption algorithm. Otherwise decryption would not be able to retrieve the original message.

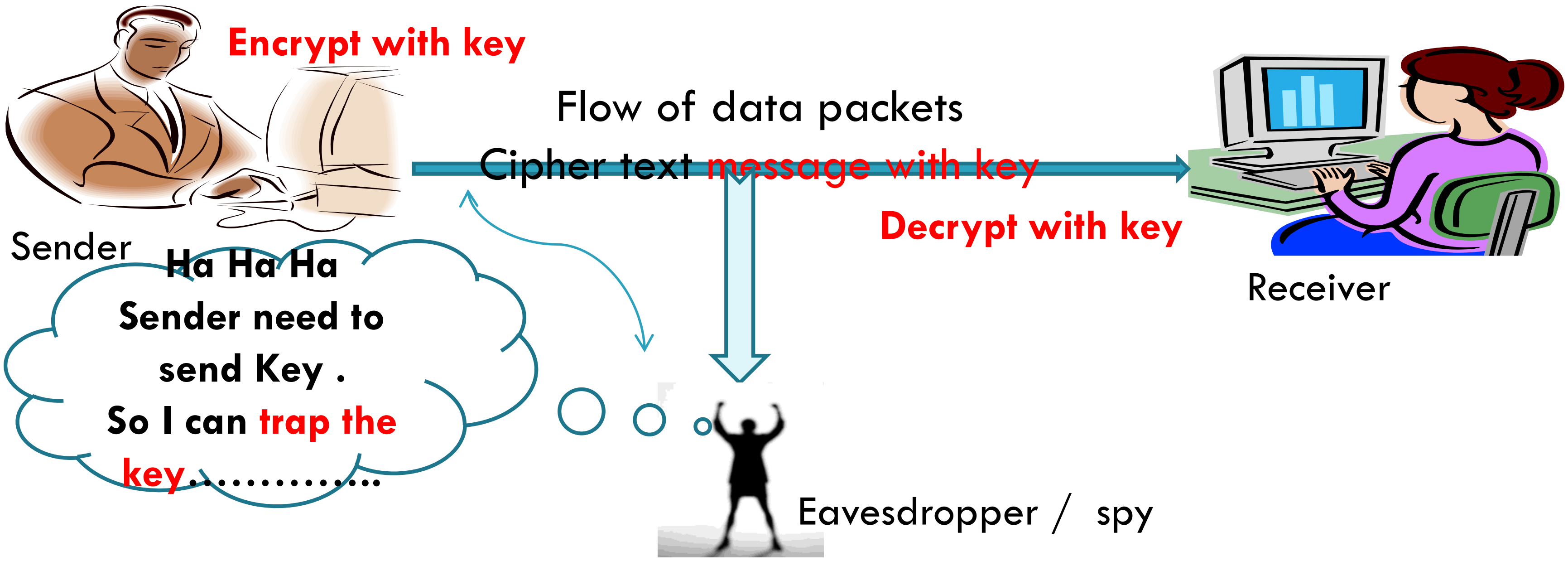
Jkm.cse@gmail.com

Cryptography

In general , the algorithm used for encryption and decryption process is usually known to everybody. However, it is the key used for encryption and decryption that makes the process of cryptography secure.

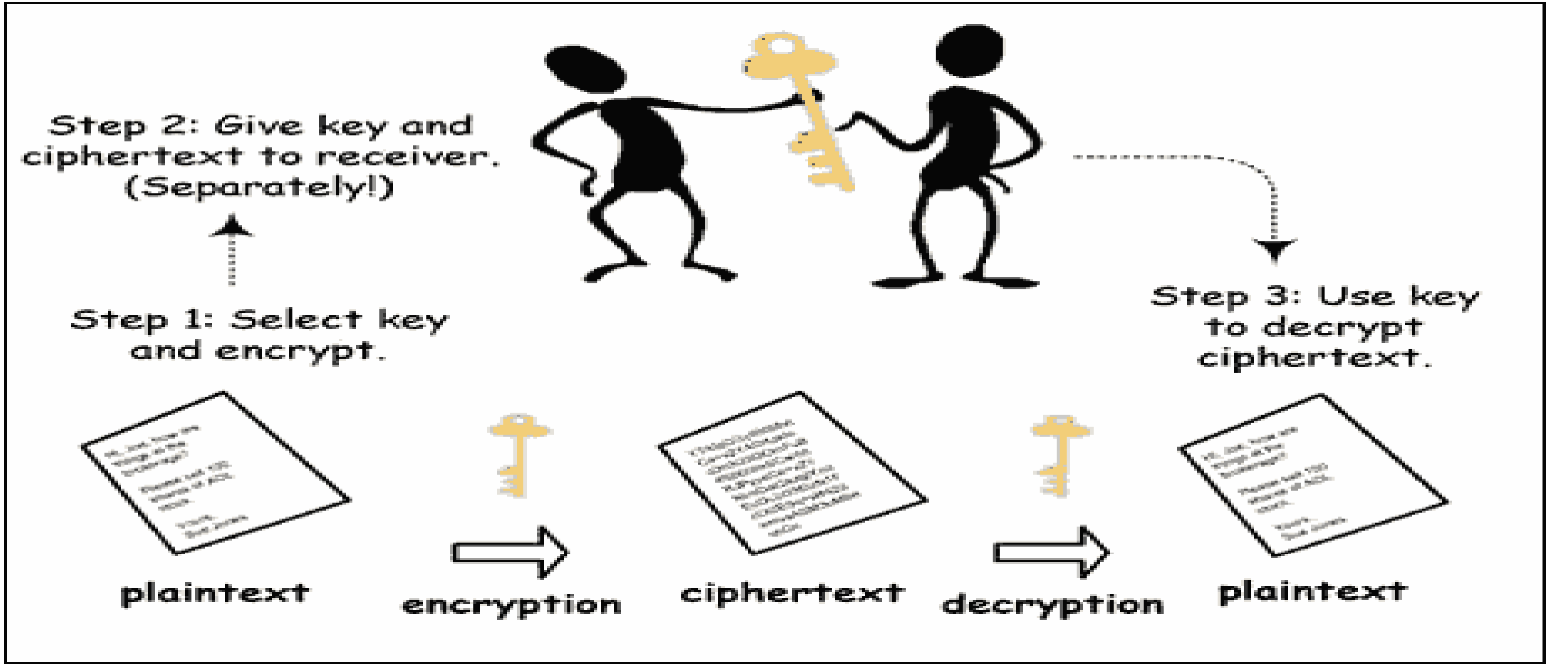


Communication……. With the concept of key

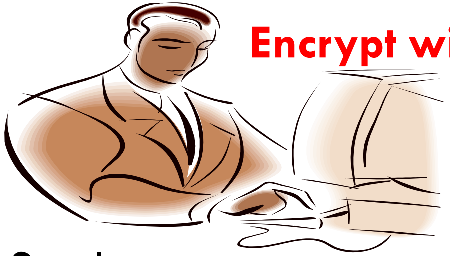


Note:- The sender and the receiver using same key ---------- Symmetric key cryptography

Applications of Symmetric Algorithms



Communication……. With the concept of key



Encrypt with private key

Flow of data packets

Cipher text message

Decrypt with public key

Sender

Receiver

Ha Ha Ha
What u think!!! , I cant trap …
(I have middle man concept)

Eavesdropper / spy

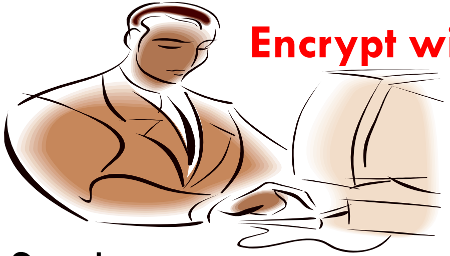
Diffie Hellman Key exchange

Note:- The sender and the receiver using different key ----------

Asymmetric key cryptography

Jkm.cse@gmail.com

Communication……. With the concept of key



Encrypt with private key

Sender

Flow of data packets

Cipher text message

Decrypt with public key

Receiver

Ha Ha Ha

What u think!!! , I

cant trap …

Diffie Hellman Key

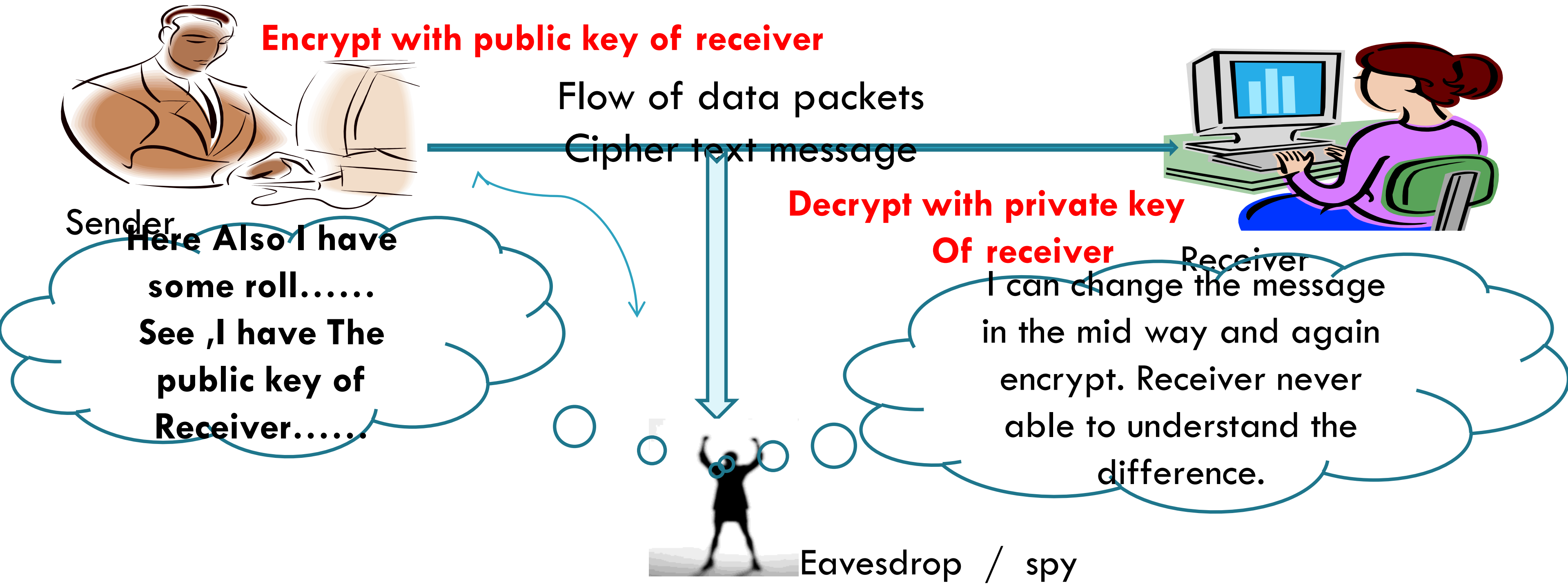
The public key of sender is public to all, So any one can decrypt message

Note:- The sender and the receiver using different key ----------

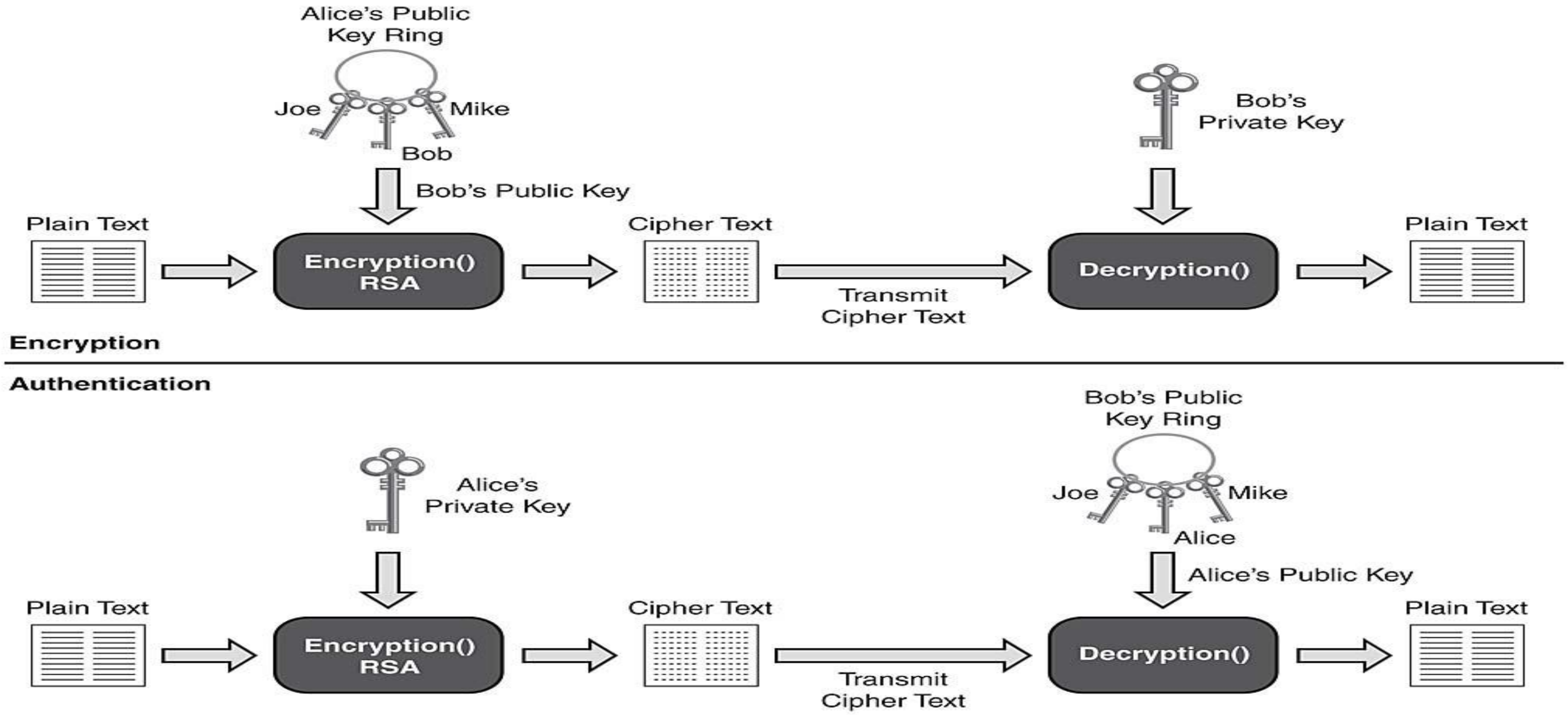
Asymmetric key cryptography

Jkm.cse@gmail.com

Communication…….



Applications of Asymmetric Algorithms



Digital Signatures

- A signature is a technique for non-repudiation based on the public key cryptography.
- The creator of a message can attach a code, the signature, which guarantees the source and integrity of the message.

AUTHENTICATION

MESSAGE DIGEST

Problem Domain

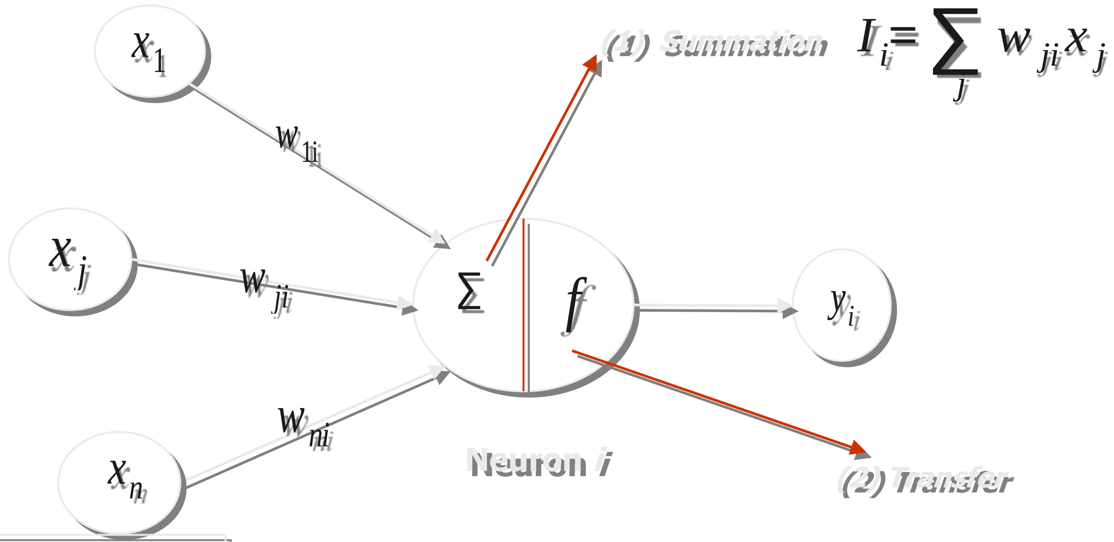
- In the present scenario, existing cryptographic technique depend on the exchange of keys through insecure public channel which are used to encrypt and decrypt the information exchange. This is vulnerable in terms of security.
- Using these key sender and receiver perform reasonably complex mathematical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

Objective

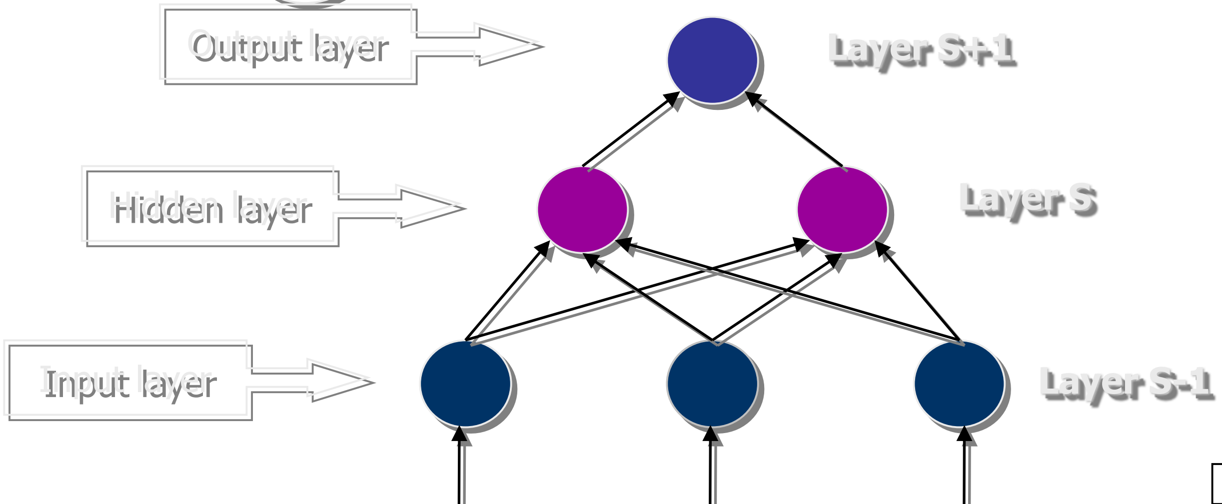
- The objectives is to enhance the security of the wireless communication system in such a way that the instead of exchanging the whole session key, Neural computing based synchronization technique is used to construct a cryptographic key exchange protocol for generating the identical session key at sender and receiver.

- 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 can be perform to synchronize group of n party to form a synchronized grouped session key.



(1) Summation
$I_i = \sum_j w_{ji} x_j$
x_1
w_{1i}
x_j
w_{ji}
Σ
f
y_i
x_n
w_{ni}
Neuron *i*
(2) Transfer

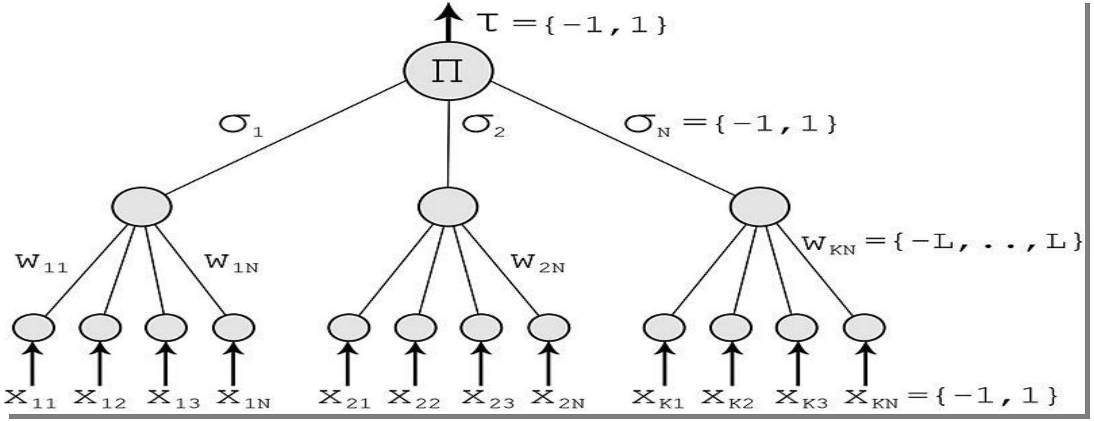


Output layer
Layer S+1
Hidden layer
Layer S
Input layer
Layer S-1

Key Generation using Neural Network

Tree Parity Machines

Tree Parity Machines, which are used by partners and attackers in neural cryptography,are multi-layer feed-forward networks.



K - the number of hidden neurons,

N - the number of input neurons connected to each hidden neuron, total (K*N) input neurons.

L - the maximum value for weight {-L..+L}

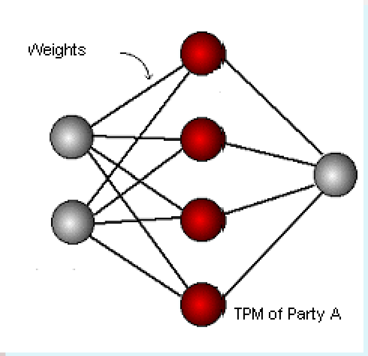
Here K = 3 and N = 4.

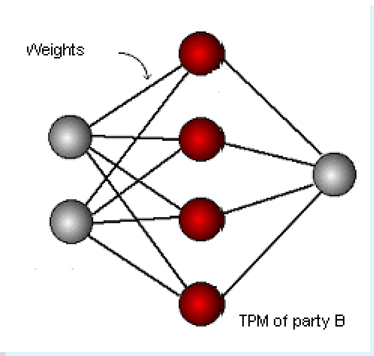
Neural Synchronization Scheme

Each party (A and B) uses its own (Same) tree parity machine.

- Synchronization of the tree parity machines is achieved in these steps

1. Initialize random weight values

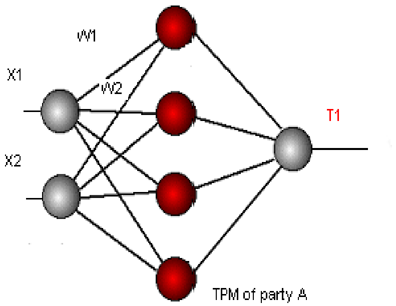




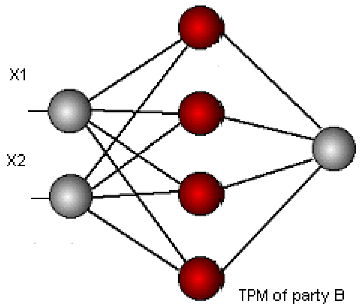
Neural Synchronization Scheme

2. Execute these steps until the full synchronization is achieved

2.1. Generate random input vector X



$$x_{ij} \in \{-1, +1\}$$



2.2. Compute the values of the hidden neurons

$$\sigma_i = \mathrm{sgn}(\sum_{j=1}^{N} w_{ij}x_{ij})$$

Signum is a simple function, which returns -1,0 or 1:

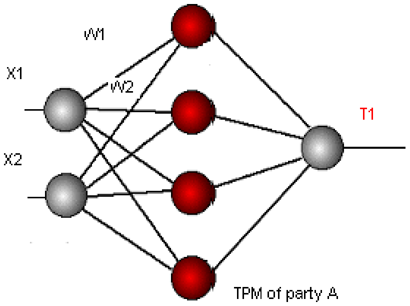
$$\mathrm{sgn}(x) = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$

Neural Synchronization Scheme

2.3. Compute the value of the output neuron

$$\tau = \prod_{i=1}^{K} \mathrm{SIGN}[\sum_{j=1}^{N} w_{i,j}\, x_{i,j}]$$

2.4. Compare the values of both tree parity machines



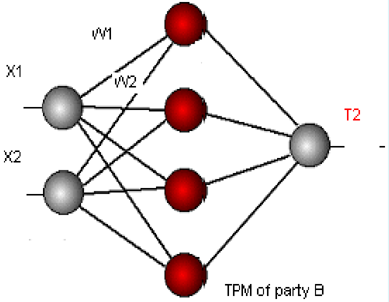
2.4.1 Outputs are others: go to 2.1

Output(A) ≠ Output(B)

2.4.2 Outputs are same:

Output(A) = Output(B)

one of the suitable learning rules is applied to the weights



How do we update the weights?

We update the weights only if the final output values of the neural machines are equal.

One of the following learning rules can be used for the synchronization:

- Hebbian learning rule:

$$w_i^+ = w_i + \sigma_i x_i \Theta(\sigma_i \tau)\Theta(\tau^A \tau^B)$$

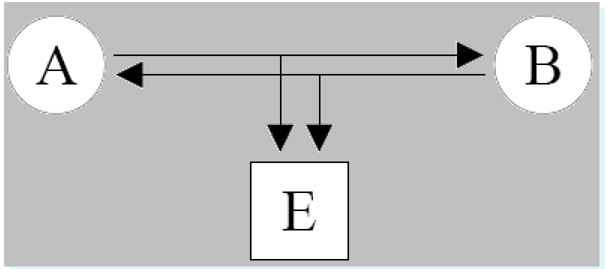
- Anti-Hebbian learning rule:

$$w_i^+ = w_i - \sigma_i x_i \Theta(\sigma_i \tau)\Theta(\tau^A \tau^B)$$

- Random walk:

$$w_i^+ = w_i + x_i \Theta(\sigma_i \tau)\Theta(\tau^A \tau^B)$$

Learning with own tree parity machine



In each step there are 3 situations possible:

1. **Output(A) ≠ Output(B):** None of the parties updates its weights.
2. **Output(A) = Output(B) = Output(E):** All the three parties update weights in their tree parity machines.
3. **Output(A) = Output(B) ≠ Output(E):** Parties A and B update their tree parity machines, but the attacker can not do that. Because of this situation his learning is slower than the synchronization of parties A and B.

Desirable Features of the Cryptographic Techniques

- *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*

Needs of Statistical Test

- **Selecting and testing random and pseudorandom number generators.** The outputs of such generators may be used in many cryptographic applications, such as the generation of key material.
- **Generators suitable for use in cryptographic applications may need to meet stronger requirements than for other applications.** In particular, their **outputs must be unpredictable in the absence of knowledge of the inputs.**
- These tests may be useful as a first step in determining **whether or not a generator is suitable for a particular cryptographic application.**
- However, no set of statistical tests can absolutely certify a generator as appropriate for usage in a particular application, i.e., **statistical testing cannot serve as a substitute for cryptanalysis.**

Randomness

- A random bit sequence could be interpreted as the result of the flips of an unbiased "fair" coin with sides that are labeled "0" and "1," with each flip having a probability of exactly ½ of producing a "0" or "1."
- Furthermore, the flips are independent of each other: the result of any previous coin flip does not affect future coin flips.
- The unbiased "fair" coin is thus the perfect random bit stream generator, since the "0" and "1" values will be randomly distributed (and [0,1] uniformly distributed).
- All elements of the sequence are generated independently of each other, and the value of the next element in the sequence cannot be predicted, regardless of how many elements have already been produced.

Unpredictability

- **Forward unpredictability-** In the case of PRNGs, if the seed is unknown, the next output number in the sequence should be unpredictable in spite of any knowledge of previous random numbers in the sequence. This property is known as forward unpredictability.

- It should also not be feasible to determine the seed from knowledge of any generated values (i.e., backward unpredictability is also required).

- No correlation between a seed and any value generated from that seed should be evident; each element of the sequence should appear to be the outcome of an independent random event whose probability is 1/2.

- To ensure forward unpredictability, care must be exercised in obtaining seeds.

Random Number Generators (RNGs)

- An RNG uses a nondeterministic source (i.e., the entropy source), along with some processing function to produce randomness.
- The entropy source typically consists of some physical quantity, such as the noise in an electrical circuit, the timing of user processes (e.g., key strokes or mouse movements), or the quantum effects in a semiconductor. Various combinations of these inputs may be used.
- The output of any RNG needs to satisfy strict randomness criteria as measured by statistical tests in order to determine that the physical sources of the RNG inputs appear random.
- For cryptographic purposes, the output of RNGs needs to be unpredictable.
- In addition, the production of high-quality random numbers may be too time consuming, making such production undesirable when a large quantity of random numbers is needed. To produce large quantities of random numbers, pseudorandom number generators may be preferable.

Pseudorandom Number Generators (PRNGs)

- The second generator type is a pseudorandom number generator (PRNG). Inputs to PRNGs are called **seeds. In contexts in which unpredictability is needed, the seed itself must be random and unpredictable.**
- **Hence, by default, a PRNG should obtain its seeds from the outputs of an RNG; i.e., a PRNG requires a RNG as a companion.**
- The outputs of a PRNG are typically deterministic functions of the seed; i.e., all true randomness is confined to seed generation. The deterministic nature of the process leads to the term "pseudorandom."
- If a pseudorandom sequence is properly constructed, each value in the sequence is produced from the previous value via transformations that appear to introduce additional randomness. A series of such transformations can eliminate statistical auto-correlations between input and output. Thus, the outputs of a PRNG may have better statistical properties and be produced faster than an RNG.

CHAOS THEORY

- **Chaos is a ubiquitous phenomenon existing in deterministic nonlinear systems which exhibit highly sensitivity to initial conditions and have random like behaviours.**
- **Discovered by Edward N. Lorenz in 1963.**
- **Actually, Chaos theory is a field of study in mathematics which has applications in several disciplines including**
  - **Meteorology**
  - **Physics**
  - **Engineering**
  - **Economics**
  - **Biology**
  - **Philosophy.**

DEFINITION OF DISCRETE CHAOS

Consider a discrete dynamical system in the general form of

$x_{k+1} = f(x_k),\ f : I \rightarrow I,\ x_0 \in I,$ **(1)**

where f is a continuous map on the interval I = [0, 1]. This system is said to be chaotic if the following conditions are satisfied:

- **Sensitive to initial conditions:**

$\exists \delta > 0\ \forall x0 \in I,\ \varepsilon > 0\ \exists n \in N,\ y0 \in I : |x0 - y0| < \varepsilon \Rightarrow |f\ n\ (x0) - f\ n\ (y0)| > \delta.$

- **Topological transitivity:**

$\forall I1, I2 \subset I\ \exists x0 \in I1,\ n \in N : f\ n\ (x0) \in I2.$ **(3)**

- **Density of periodic points in I:**

Let $P = p \in I | \exists n \in N : f\ n\ (p) = p$ be the set of periodic points of f . Then P is dense in I: P = I.

FEATURES OF CHAOS

- Sensitive dependency on initial conditions and system parameters.
- Pseudo-random property
- Nonperiodicity
- Topological transitivity
- Ergodicity

RANDOM BIT GENERATION USING CHAOTIC MAP :

To create a chaotic stream cipher, a random bit stream is to be generated using chaotic system.

Pseudo Noise (PN) Sequences :

A pseudo random bit generator (PRBG) is a deterministic algorithm, which uses a truly random binary sequence of length k as input called seed and produces a binary sequence of length l>>k, which is called pseudorandom sequence. This pseudorandom sequence appears to be random. The output of a PRBG is not truly random; infact the number of possible output sequences is at most a small fraction (/) of all possible binary sequences of length l. The basic idea is to take a small truly random sequence of length k and expand it to a sequence of much larger length l in such a way that an adversary cannot efficiently distinguish between output sequence of PRBG and truly random sequence of length.

SKEW TENT MAP

The skew tent map is ergodic and has uniform invariant density function in its definition interval. It is the simplest kind of one-dimensional chaotic map which is defined as:

$$x_{i+1} = F(\alpha, x_i) = \begin{cases} \dfrac{x_i}{\alpha} & x_i = [0, \alpha) \\ \dfrac{1 - x_i}{1 - \alpha} & x_i = (\alpha, 1] \end{cases}$$

where α and x_i are system parameter and initial condition of the map respectively. It is a non-invertible transformation of unit interval onto itself and contains only one system parameter α, which determines position of the top of the tent in the interval [0,1]. A sequence computed by iterating *F(α, x)*, is expansionary everywhere in the interval [0,1] and

CHAOS BASED PN SEQUENCE

A **Chaotic tent map based Bit Generator** has been used where two skew tent maps are chosen. They are piecewise linear chaotic maps. The pseudorandom bit sequence is generated by comparing the outputs of both the chaotic logistic maps. The system parameter for both the chaotic maps is kept same and is in the chaotic regime.

Let $f_1(x_0, \alpha)$ and $f_2(y_0, \alpha)$ be two piecewise linear chaotic maps such that:

$$x_{i+1} = f_1(\alpha, x_i),$$
$$y_{i+1} = f_2(\alpha, y_i),$$

where α is the system parameter and is same for chaotic tent maps, x_i and y_i are the initial conditions and x_{i+1} and y_{i+1} are their new corresponding states.

CHAOS BASED PN SEQUENCE

The chaotic tent map produces the binary sequences by comparing the outputs of the piecewise linear chaotic maps in the following way:

$$g(x_{n+1}, y_{n+1}) = \begin{cases} 0 & \text{if } \; x_{n+1} < y_{n+1} \\ 1 & \text{otherwise} \end{cases}$$

Analysis of Randomness Of Bit Streams:

For the analysis of randomness of bit streams, five sets of parameter values

$(\alpha_1, x_1, y_1)=(0.48999, 0.5006841, 0.538167586)$,
$(\alpha_2, x_2, y_2)=(0.49045, 0.6410089, 0.505410089)$,
$(\alpha_3, x_3, y_3)=(0.49493, 0.4417689, 0.754193089)$,
$(\alpha_4, x_4, y_4)=(0.49951, 0.5166892, 0.273417389)$ and
$(\alpha_5, x_5, y_5)=(0.49999, 0.1996892, 0.738567389)$

have been used for 262144 sized binary sequences.

So, $N=262144$

MONOBIT TEST:

The purpose of this test is to determine whether the frequency of 0's and 1's in binary sequences generated by the skew tent map are approximately same. Let n_0 , n_1 denote the number of 0's and 1's in binary sequences respectively. It is calculated using the formula[18] :

$$\chi^2 = \frac{(n_0 - n_1)^2}{n}$$

χ^2 **which approximately follow a distribution with one degree of freedom.**

Results of Monobit Test

Size	Parameter	Numbers in binary sequences n_0	n_1	Calculated value	Critical value at α=0.05
N=262144	(α_1, x_1, y_1)	130948	131196	0.234619	3.8414588207
	(α_2, x_2, y_2)	131026	131118	0.032288	
	(α_3, x_3, y_3)	131311	130833	0.871597	
	(α_4, x_4, y_4)	131164	130980	0.129150	
	(α_5, x_5, y_5)	131207	130937	0.278091	

SERIAL TEST

The purpose of this test is to determine whether the number of occurrences of 00, 01, 10, and 11 as sub sequences of s are approximately the same. Let n_0 , n_1 denote the number of 0's and 1's in s, respectively, and let n_{00} , n_{01} , n_{10} , n_{11} denote the number of occurrences of 00, 01, 10, 11 in s, respectively. The sum of the number of occurrences of 00, 01, 10 and 11 is (n − 1) as the sub sequences are allowed to overlap. The statistic used is

$$\chi^2 = \frac{4}{n-1}\left(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2\right) - \frac{2}{n}\left(n_0^2 + n_1^2\right)$$

which approximately follows a distribution with 2 degrees of freedom if n ≥ 21.

Result of Serial Test

Size	Parameter	Calculated χ^2 value	Critical χ^2 value at $\alpha = 0.05$
N=262144	(α_1, x_1, y_1)	75.622020	5.991464547
	(α_2, x_2, y_2)	82.892036	
	(α_3, x_3, y_3)	17.994071	
	(α_4, x_4, y_4)	2.156072	
	(α_5, x_5, y_5)	2.819054	

POKER TEST

Let m be a positive integer such that $n/m \geq 5*(2^m)$ and let $k = n/m$ where n is the size of binary sequence. The binary sequence is divided into *k* non-overlapping parts, each of length *m* and n_i is the number of occurrence of i^{th} type of sequences of length *m*, where $1 \leq i \leq 2^m$. The poker test determines whether the sequences of length *m* each appear approximately the same number of times in s.

χ^2 is calculated using the formula:

$$\chi^2 = \frac{2^m}{k}\left(\sum_{i=1}^{2^m} n_i^2\right) - k$$

Result of Poker Test

Size	Block length (m) in bits	df (2^m-1)	Calculated value					Critical value at α=0.05
			(α_1, x_1, y_1)	(α_2, x_2, y_2)	(α_3, x_3, y_3)	(α_4, x_4, y_4)	(α_5,x_5,y_5)	
N=262144	2	3	34.572021	40.347107	10.828308	0.262146	0.330139	3.841459
	3	7	62.963425	65.408991	7.895343	6.631545	8.185384	14.06714
	4	15	58.479004	76.179688	27.362793	10.141113	24.706055	24.99579

Maps

Recurrence Relation

jkm.cse@gmail.com

$X_{n+1} = \mu_2 X_n$ for $X_n < 1/2$

$\mu_2 X_n$ for $1/2 <= X_n$

$Y_{n+1} = \mu_2 Y_n$ for $Y_n < 1/2$

$\mu_2 Y_n$ for $1/2 <= Y_n$

Random Bit Generator

$G(X_{n+1}, Y_{n+1}) = 0$ if $X_{n+1>=} Y_{n+1}$

1 if $X_{n+1>=} Y_{n+1}$

SKEW TENT MAPS

jkm.cse@gmail.com

$X_{n+1} = P = X_i \; / \alpha$ for $X_i = [0, \alpha]$

$P' = 1 - X_i \; / (1-\alpha)$ for $X_i = [\alpha, 1]$

Binary Bit Generator

$G_{i+1} = 0$ **if P < P' Else 1**

CROSS COUPLED MAP

jkm.cse@gmail.com

$X_{n+1} = X_i \ /\alpha$ for $X_i = [0, \alpha]$

$Y_{n+1} = 1 - Y_i \ /(1-\alpha)$ for $X_i = [\alpha, 1]$

Random Bit Generator

$G(X_{n+1}, Y_{n+1}) = 0$ if $X_{n+1} >= Y_{n+1}$

1 if $X_{n+1} >= Y_{n+1}$

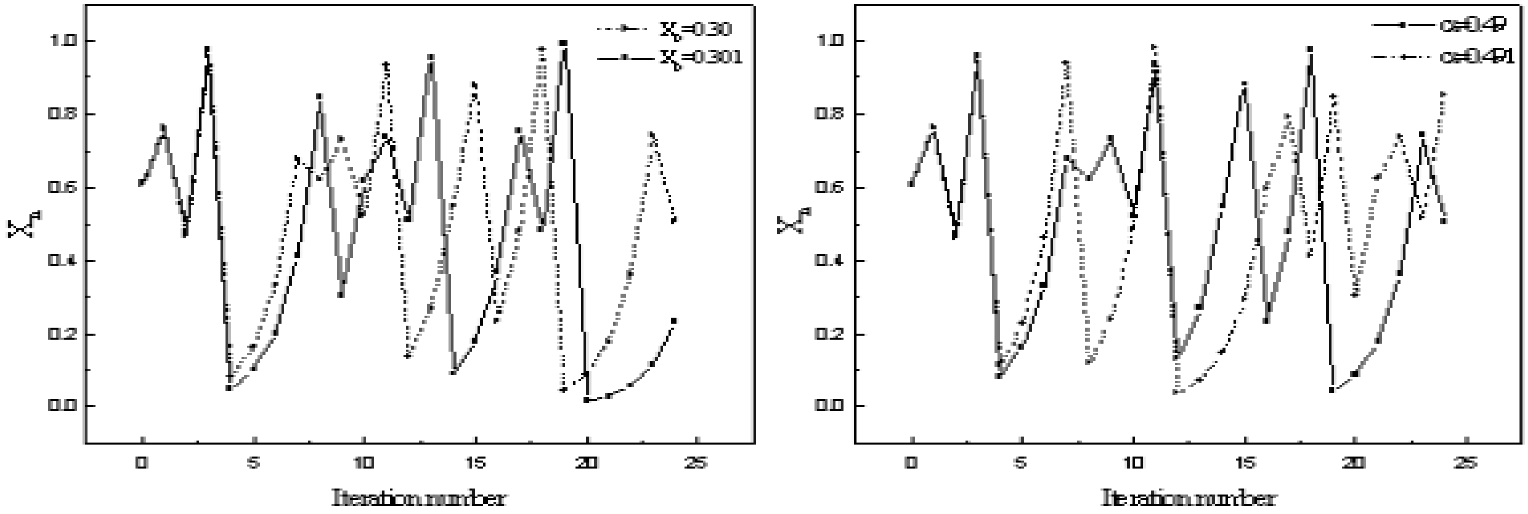


Figure 2: Shows the sensitivity of chaotic solution of skew tent map on initial condition and system parameter (*α*).

ARNOLD'S CAT MAP

Arnold’s Cat Map :-

An image is hit with a transformation that apparently randomizes the original organization of its pixels. However, if iterated enough times, as though by magic, the original image reappears.

If we let X = $\begin{bmatrix} x \\ y \end{bmatrix}$ be a *n x n* matrix of some image, Arnold's cat map is the transformation

$$\Gamma \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x + y \\ x + 2y \end{bmatrix} mod\, n$$

where mod is the modulo of the $\begin{bmatrix} x + y \\ x + 2y \end{bmatrix}$

and n. For example, 11.23 mod 1 = .23 or 150 mod 100= 50 or $\begin{bmatrix} 153 \\ 184 \end{bmatrix}$ mod 100 = $\begin{bmatrix} 53 \\ 84 \end{bmatrix}$ Since the signs of both

arguments are the same sign in this exercise, the modulo will simply be the remainder of the long division of

$$\begin{bmatrix} x + y \\ x + 2y \end{bmatrix} mod\, n$$

To better understand the mechanism of the transformation Γ, let us decompose it into its elemental pieces.

1. Shear in the x-diection by a factor of 1.

$$\begin{bmatrix} x \\ y \end{bmatrix} \to \begin{bmatrix} x+y \\ y \end{bmatrix}$$

2. Shear in the y-direction by a factor of 1.

$$\begin{bmatrix} x \\ y \end{bmatrix} \to \begin{bmatrix} x \\ x+y \end{bmatrix}$$

3. Evaluate the modulo.

$$\begin{bmatrix} x \\ y \end{bmatrix} \to \begin{bmatrix} x \\ y \end{bmatrix} \bmod n$$

Let the period be Π

Experimentally, no elegant model could be developed for the relationship between the period of an image and *n, its number of rows or columns. In general, it may be claimed that as the value of n* increases, the period tends to increases. However, this is not always true.

1. $\prod(n) = 3n$ if and only if $n = 2 \bullet 5^k$ for $k = 1, 2, \ldots$

2. $\prod(n) = 2n$ if and only if $n = 5^k$ or $n = 6 \bullet 5^k$ for $k = 1, 2, \ldots$

3. $\prod(n) \leq \frac{12n}{7}$ for all other choices of n

Consider the ordinary pixel $\begin{bmatrix} 16 \\ 10 \end{bmatrix}$ of the 124 x 124 image considered previously. It takes the following path:

$$\Gamma \begin{bmatrix} 16 \\ 10 \end{bmatrix} \rightarrow \begin{bmatrix} 16+10 \\ 16+2X10 \end{bmatrix} mod\ 124$$

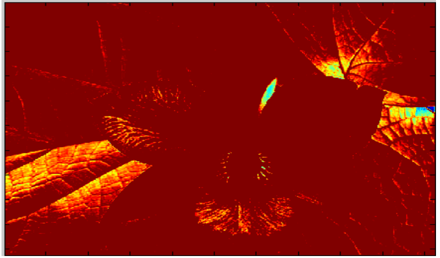
$$= \begin{bmatrix} 26 \\ 36 \end{bmatrix} \rightarrow \begin{bmatrix} 62 \\ 98 \end{bmatrix} \rightarrow \begin{bmatrix} 36 \\ 10 \end{bmatrix} \rightarrow \begin{bmatrix} 46 \\ 56 \end{bmatrix} \rightarrow \begin{bmatrix} 102 \\ 34 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 12 \\ 46 \end{bmatrix} \rightarrow \begin{bmatrix} 58 \\ 104 \end{bmatrix} \rightarrow \begin{bmatrix} 38 \\ 18 \end{bmatrix} \rightarrow \begin{bmatrix} 56 \\ 74 \end{bmatrix} \rightarrow \begin{bmatrix} 6 \\ 80 \end{bmatrix}$$

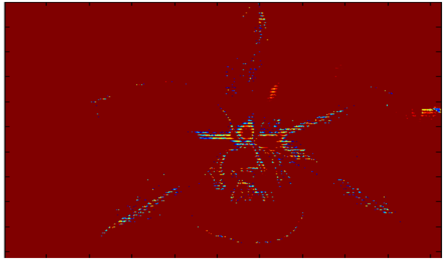
$$\rightarrow \begin{bmatrix} 86 \\ 42 \end{bmatrix} \rightarrow \begin{bmatrix} 4 \\ 46 \end{bmatrix} \rightarrow \begin{bmatrix} 50 \\ 96 \end{bmatrix} \rightarrow \begin{bmatrix} 22 \\ 118 \end{bmatrix} \rightarrow \begin{bmatrix} 16 \\ 10 \end{bmatrix}$$

Experimental Results : -

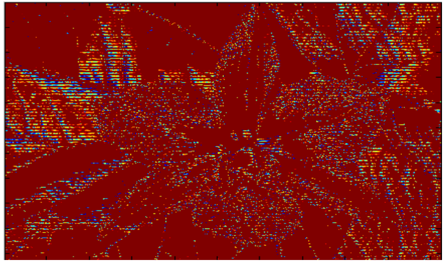
the following *512 x 512 image of the* Flower was iterated with the transformation Γ



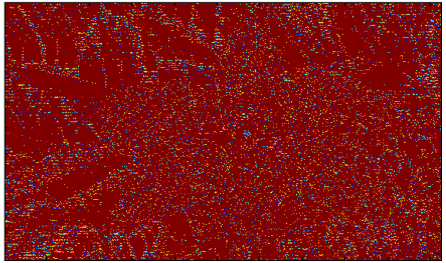
Original



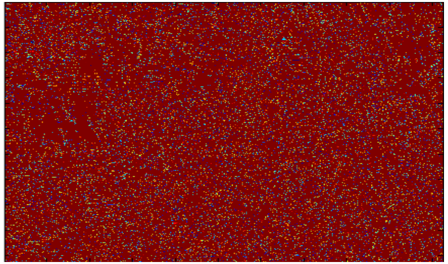
Iteration - 1



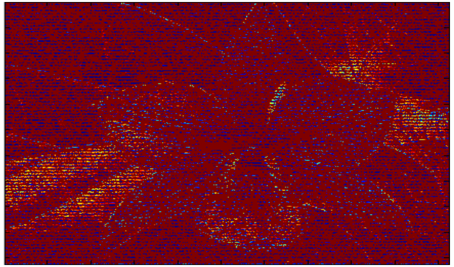
Iteration - 2



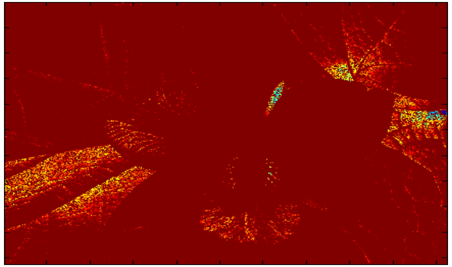
Iteration - 3



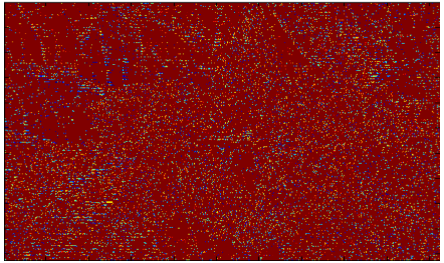
Iteration - 4



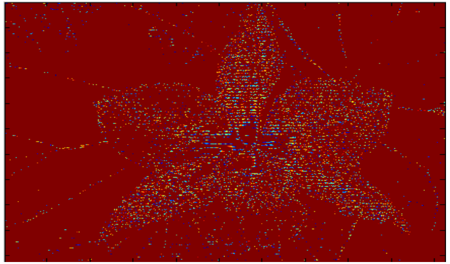
Iteration - 191



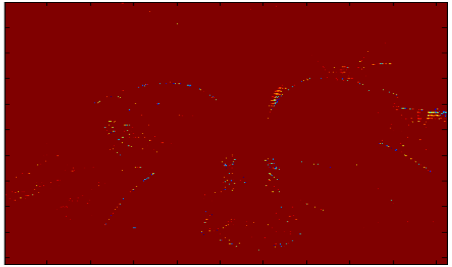
Iteration - 192



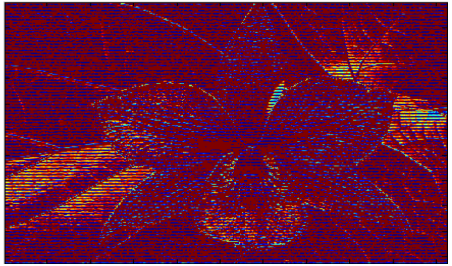
Iteration - 380



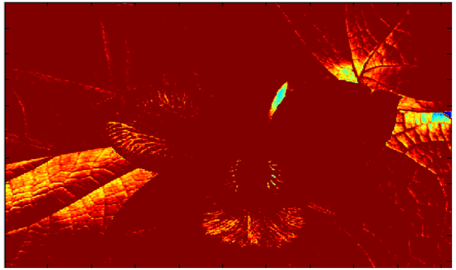
Iteration - 381



Iteration - 382



Iteration - 383

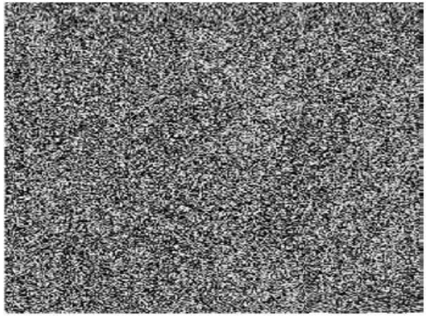


Final (Iteration 384)

IMAGE ENCRYPTION

Encryption

Decryption

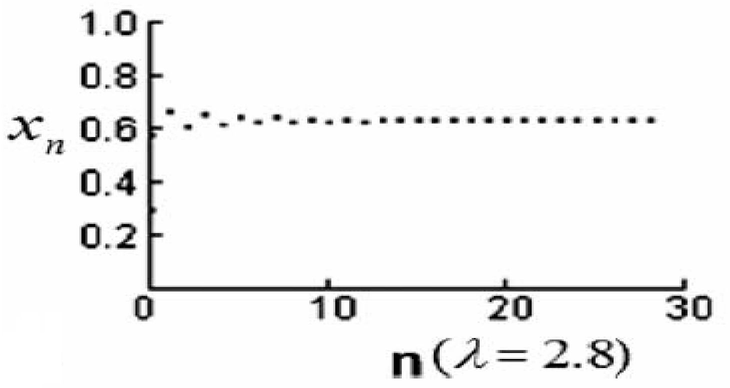


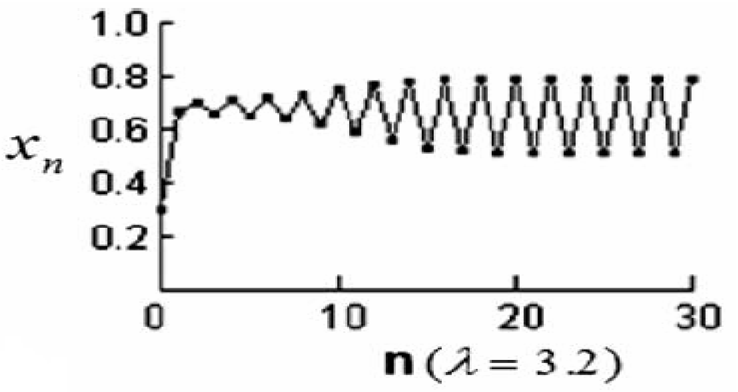
Lena

encrypted image

Logistic map

$$x_{n+1} = \lambda \times x_n \times (1 - x_n), \text{ where } \lambda \in (0,4),\ n = 0, 1, \ldots$$



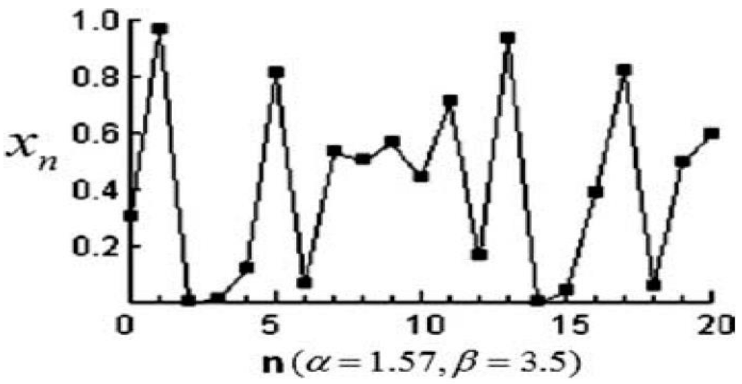


CA (Chaotic Algorithm)

$$x_{n+1} = \lambda \cdot \mathrm{tg}(\alpha x_n) \cdot (1 - x_n)^{\beta}, \text{ where } x_n \in (0,1),\ n = 0, 1, 2, \ldots$$

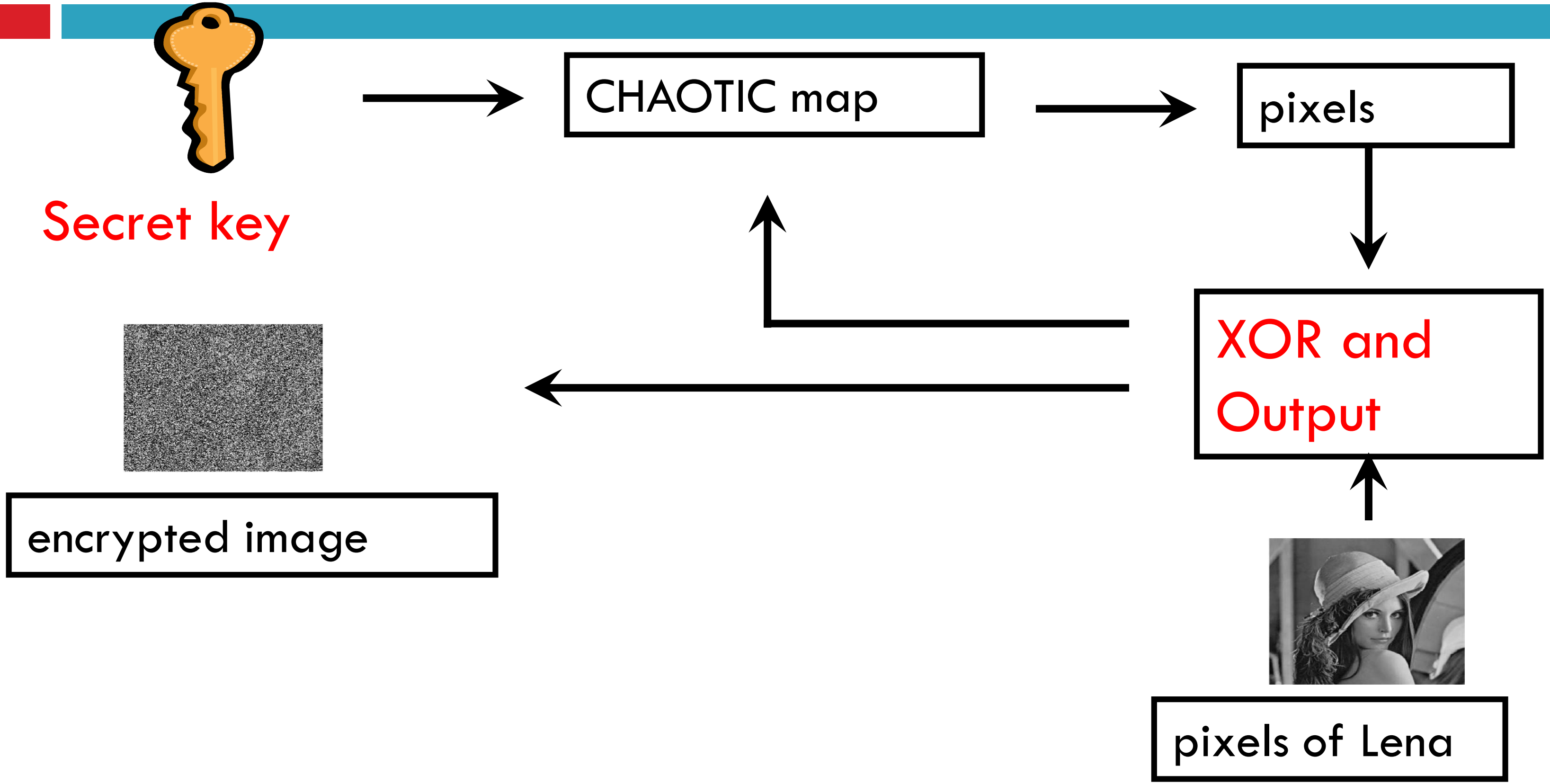
tangent function

power function



Iteration property of the CA map

ENCRYPTION


Secret key
CHAOTIC map
pixels
XOR and Output
encrypted image
pixels of Lena

Encryption algorithm

- Secret key
- $(x_0, \alpha, \beta) = (0.98765432101 2345, 1.1, 5)$
- 987 mod 256=219, 654 mod 256=142, 321 mod 256=65, 012 mod 256=12, 345 mod 256=89

5 pixels →

200	210	220	225	230

↓ XOR

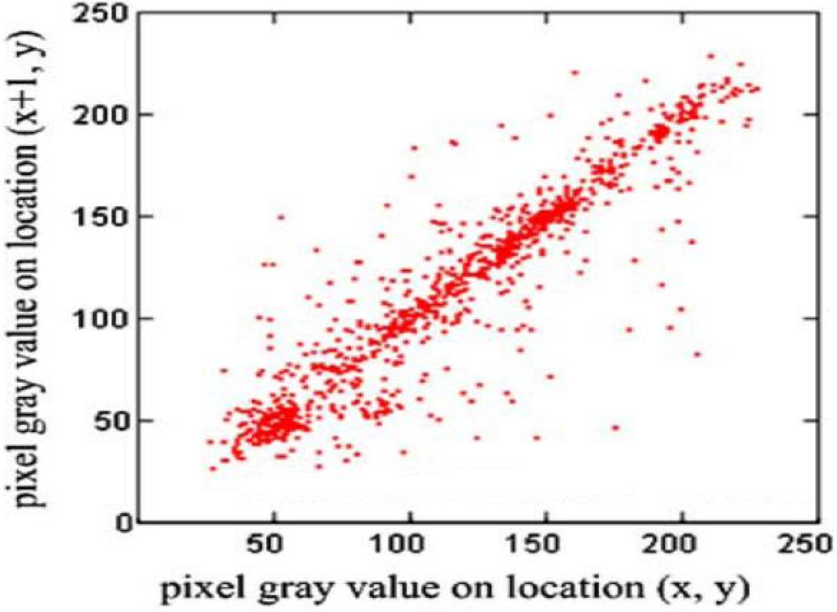
219	142	65	12	89

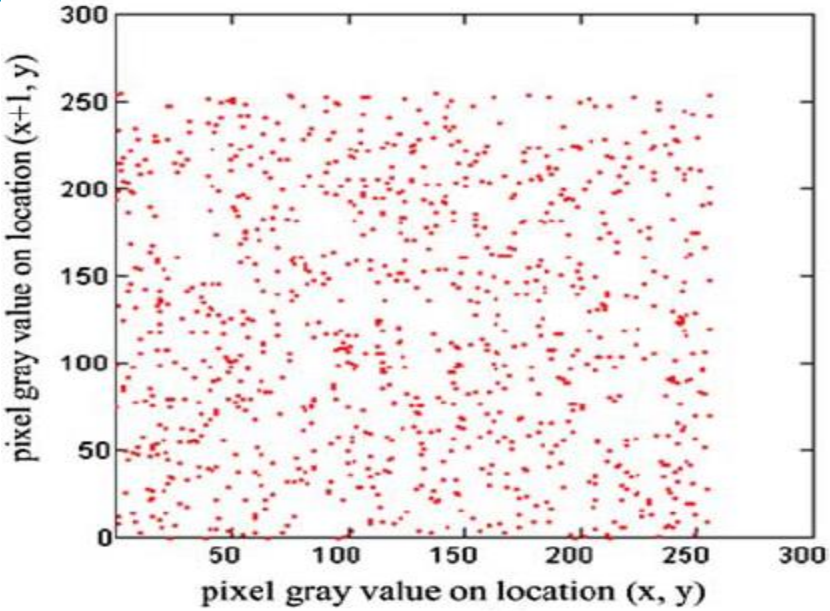
↓

19	92	157	237	191

← 5 pixels

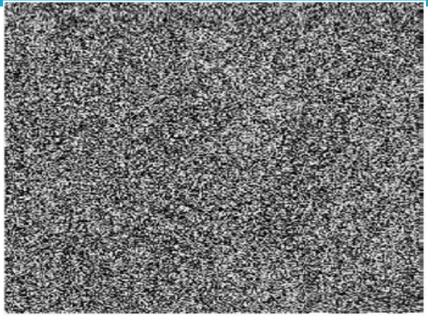
Experimental results(1/2)





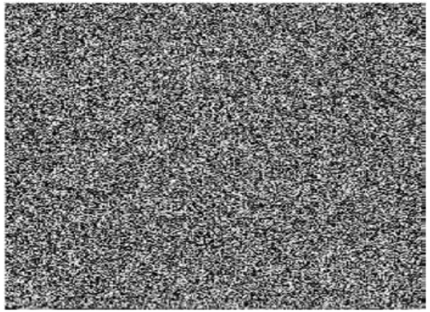
Experimental results

Encryption



encryption key $K = (x_0, \alpha, \beta) = (0.98765432101234 5, 1.1, 5)$

Decryption with wrong key



wrong key $K_1 = (x_0, \alpha, \beta) = (0.98765432101234 6, 1.1, 5)$

Testing

- **Uniformity:** At any point in the generation of a sequence of random or pseudorandom bits, the occurrence of a zero or one is equally likely, i.e., the probability of each is exactly 1/2. The expected number of zeros (or ones) is $n/2$, *where* n *= the sequence length.*

- **Scalability:** Any test applicable to a sequence can also be applied to subsequences extracted at random. If a sequence is random, then any such extracted subsequence should also be random. Hence, any extracted subsequence should pass any test for randomness.

- **Consistency:** The behavior of a generator must be consistent across starting values (seeds). It is inadequate to test a PRNG based on the output from a single seed, or an RNG on the basis of an output produced from a single physical output.

Random Number Generation Tests

The NIST Test Suite is a statistical package consisting of 15 tests that were developed to test the randomness of (arbitrarily long) binary sequences produced by either hardware or software based cryptographic random or pseudorandom number generators. These tests focus on a variety of different types of non-randomness that could exist in a sequence. Some tests are decomposable into a variety of subtests.

Advantages of Neural Network for Cryptology

Neural networks' most important property is their generalization capability. This ability ensures they produce reasonable results when they are fed with inputs not previously encountered. This makes them extremely useful for many applications. Assuming that x_k denotes inputs of a network and denotes targets of a network, it is easy to compute y_k from x_k. But if target y_k is different from input x_k, it is difficult to compute the input from the target. As a result of this property, hash functions can be generated using ANNs.

$$y_k = \theta\left(\sum_{j=1}^{m} w_k x_i + b_k\right)$$

Neural-Based Pseudo-Random Number Generator

The neural network is a well-known method that has function approximation capabilities. After training with initial values (weights and bias), the reached output is called neural-based pseudo-random number. A sequence of pseudo-random numbers is generated using an ANN.

Implementation of Neural Cryptology

- In neural cryptology, two neural networks that have the same topology (layer size, transfer function, neuron number in each layer, weight and bias values) can achieve the same output when trained for the same input.

- In other words, two networks which are trained on their mutual input can synchronize with mutual synaptic weights.

- To implement this ability for cryptosystems, two partners (receiver and sender) have to share mutual topological data and chipper text as a secret key.

- In this study, in order to decrypt, the ability to predict unforeseen situations using an artificial neural network is utilized and a neural-based cryptosystem is constructed.

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

- 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.
- Fractal triangle based encryption/decryption technique is performed with the help of KSOFM tuned session key to generate the cipher text.

Parameters in KSOFMSCT

- 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

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: Frame format of *SYN* frame

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

Figure: Finish Synchronization (*FIN_SYN*) frame

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

Figure: Acknowledgement of Synchronization (*ACK_SYN*) frame

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

Figure: Negative Acknowledgement of Synchronization (*NAK_SYN*) frame

Fractal Triangle based Encryption

- 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”

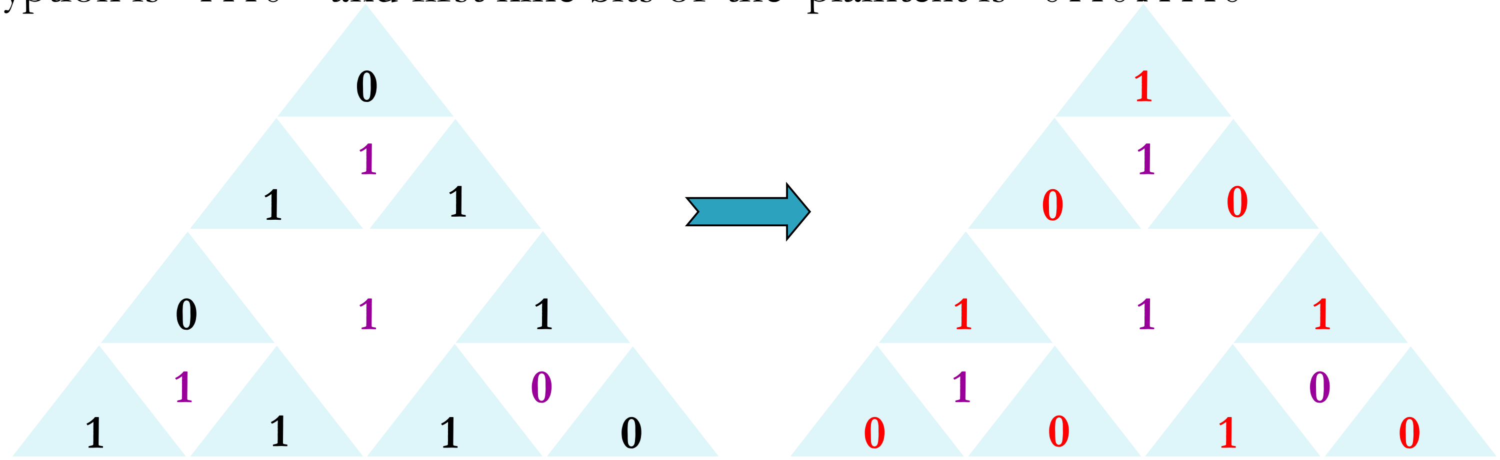


Figure 1: Exclusive-OR operation between central key bit of each triangle and vertex elements of each triangle

Fractal Triangle based Encryption

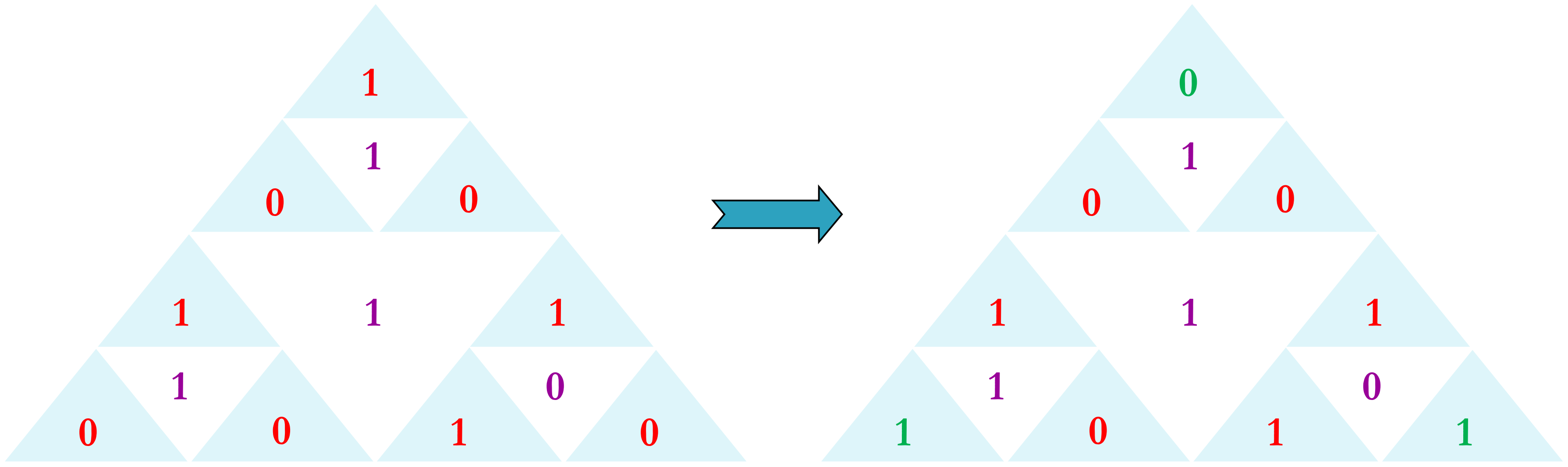


Figure 2: Exclusive-OR operation between triangle's centered key and vertex elements of big triangle

Fractal Triangle based Encryption

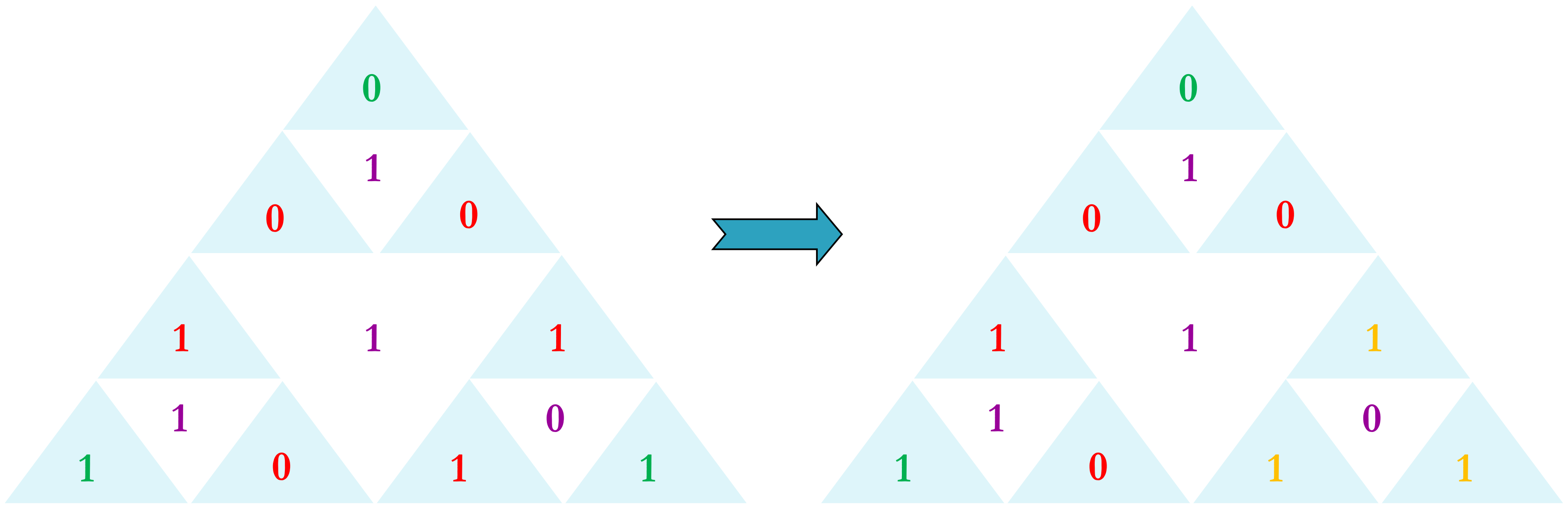


Figure 3: Exclusive-OR operation between upper triangle's vertex elements with right triangle's vertex elements

Fractal Triangle based Encryption

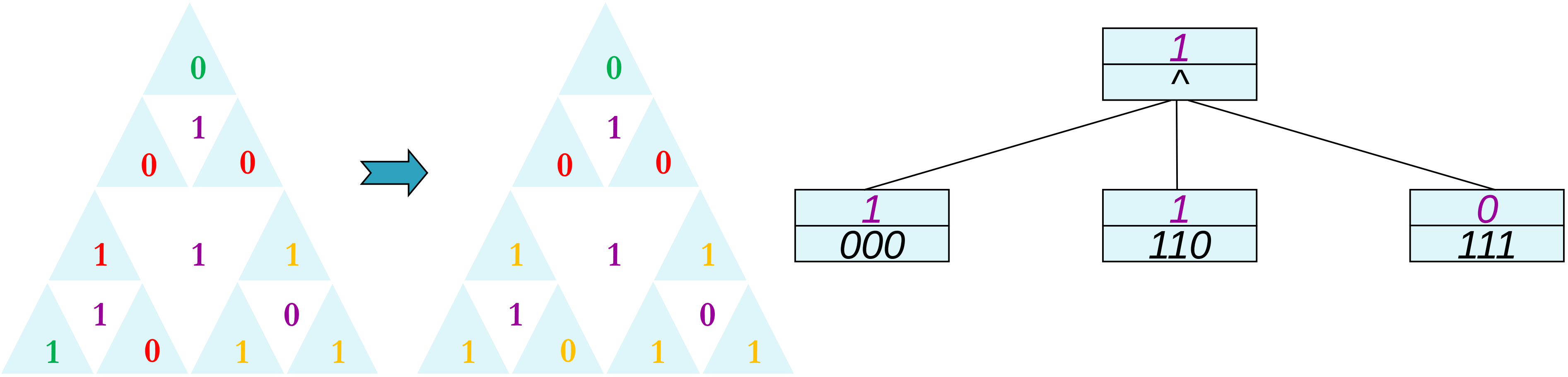


Figure 4: *Exclusive-OR* operation between upper triangle's vertex elements with left triangle's vertex elements

For example from the KSOFM synchronized bits, the following session key is generated

10011110/10001110/01110100/01010111/11100101/01010101/10100011/11011010/
10100011/11010100/10111101/01001101/01101111/10100010/11000011/11101010

Now, consider the plaintext to be encrypted is "**Technique**",

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

Table 1

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

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 3

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 4

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 5

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 6

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 7

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 8

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.

Histogram

The histogram of a document expresses the frequency distribution of the characters of this document in graphical form in a corresponding window (plot type).

The x-axis of the histogram contains all the characters in the character set: while in a window for hexadecimal inputs and outputs, the character set contains the numbers 0 to 255 (see ASCII Table).

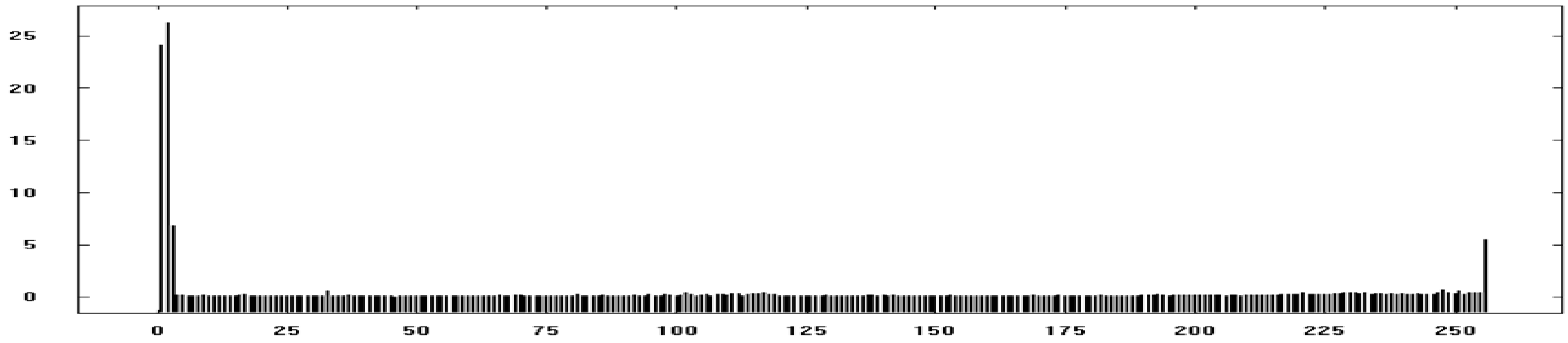


Figure: Input source stream of *.dll* file

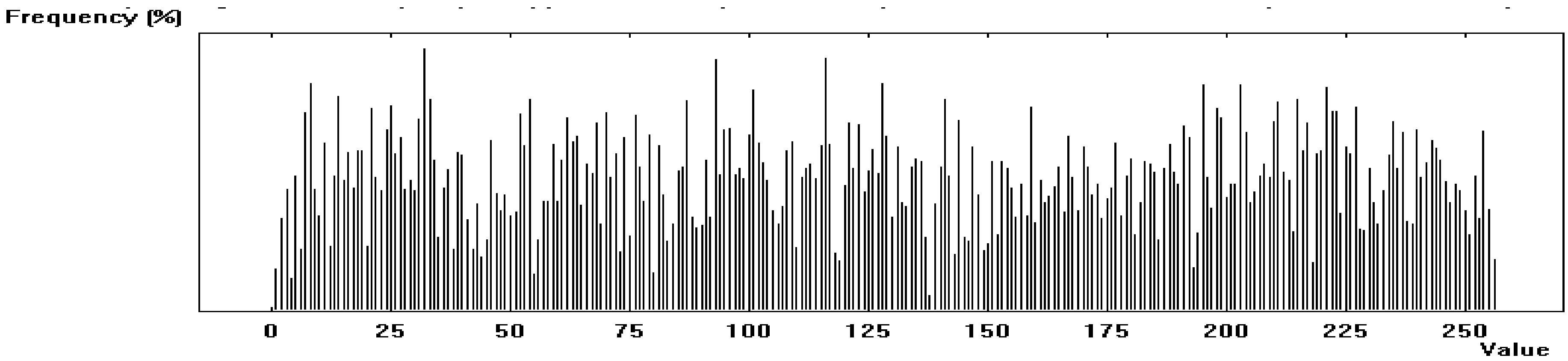


Figure: Encrypted stream using KSOMSCT for *.dll* file

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 n > 64 bytes has n-63 such index numbers in its characteristics.

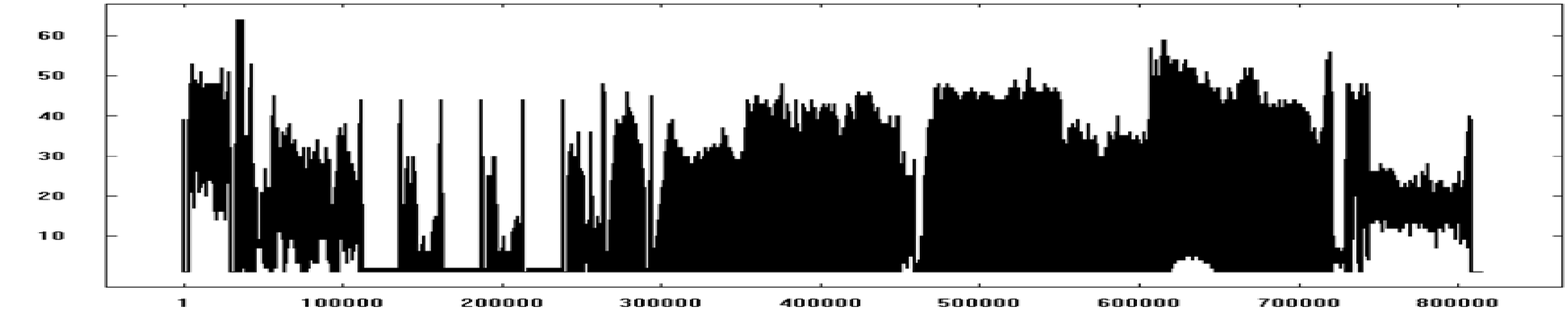


Figure : Floating frequency of the input *.dll* source stream

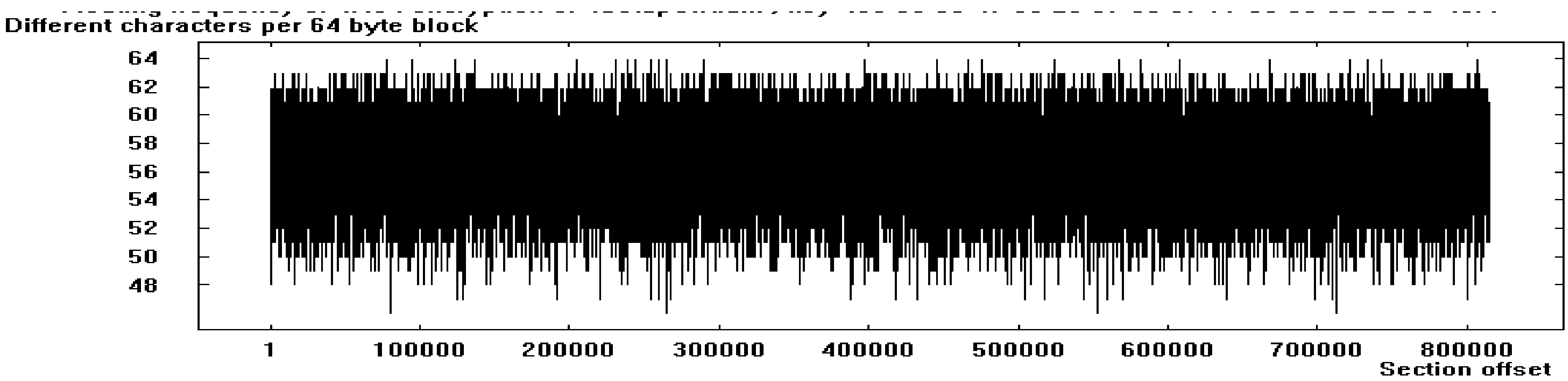


Figure: Floating frequency of the encrypted stream using KSOMSCT for *.dll* file

Autocorrelation

The autocorrelation of a document is an index of the similarity of different sections of the document.

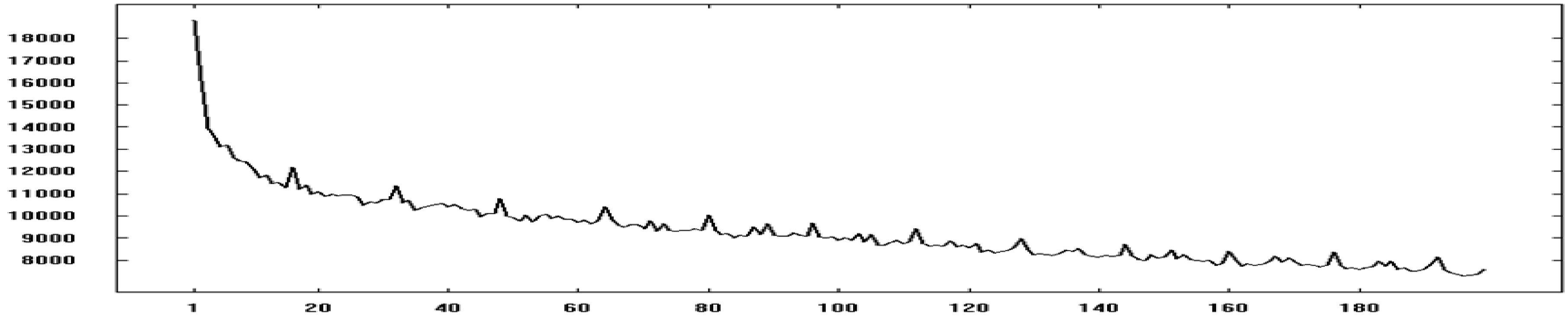


Figure : Autocorrelation of the input *.dll* source stream

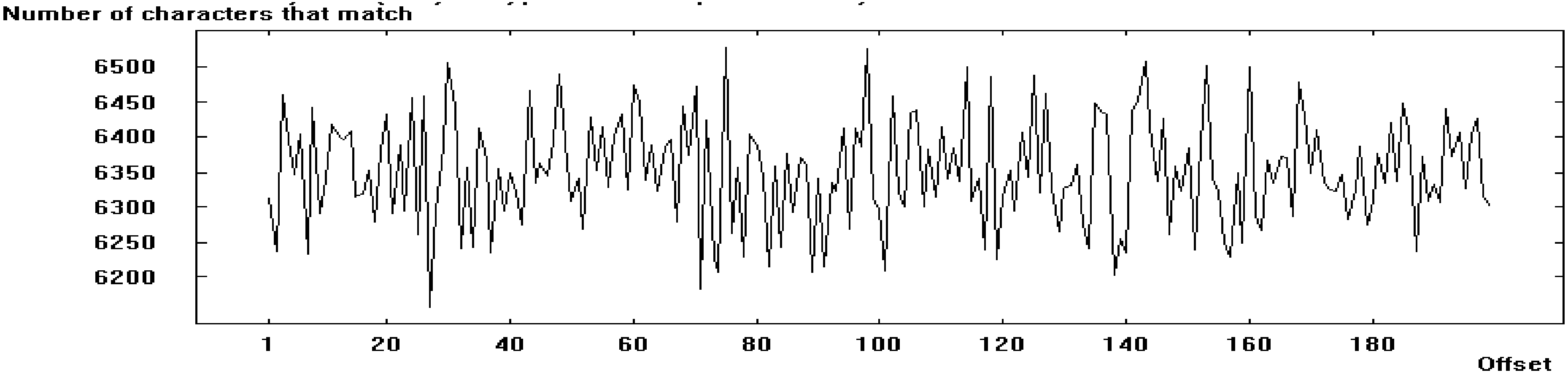


Figure : Autocorrelation of the encrypted stream using KSOMSCT for *.dll* file

Comparisons of Chi-Square value of *.dll* files

Serial no.	Source File name	Source file size (In bytes)	Chi-Square values		
			KSOMSCT	TDES	AES
1	a01.dll	3216	34933	36054	26036
2	a02.dll	6,656	108674	193318	118331
3	a03.dll	12,288	137834	165053	81475
4	a04.dll	24,576	6147653	8310677	4794027
5	a05.dll	58,784	321983	806803	466466
6	a06.dll	85,020	449272	654756	433872
7	a07.dll	169,472	863406	1473410	1601070
8	a08.dll	359,936	731276	423984	398685
9	a09.dll	593,920	1198749	1367968	1277751
10	a10.dll	909,312	1680956	2377544	2275676
11	a11.dll	1,293,824	1633962	1065999	948834
12	a12.dll	1,925,185	45172384	46245126	47627346
13	a13.dll	2,498,560	4887347	4616320	4625829
14	a14.dll	3,485,968	11590534	14567497	13560121
15	a15.dll	3,790,336	8942907	7110339	7051889
16	a16.dll	4,253,816	9215648	8451794	8194777
17	a17.dll	4,575,232	8914895	8632408	8649446
18	a18.dll	4,883,456	9912906	8866085	8450004
19	a19.dll	5,054,464	14109345	14875409	13265423
20	a20.dll	5,456,704	12673493	12432371	12239623
Average			6934661	7133646	6804334

Comparisons of Chi-Square value of *.exe* files

Serial no.	Source File name	Source file size (In bytes)	Chi-Square values		
			KSOMSCT	TDES	AES
1	a01. exe	1,063	14172	31047	15349
2	a02. exe	2,518	89946	167604	58911
3	a03. exe	8,250	86091	3171258	1193952
4	a04. exe	15,937	99387	137421	90439
5	a05. exe	22,874	25985	40605	42948
6	a06. exe	35,106	257394	751034	996561
7	a07. exe	52,032	108746	252246	227972
8	a08. exe	145,387	622467	1619619	879622
9	a09. exe	248,273	618647	1188392	1206461
10	a10. exe	478,321	796454	1646895	1611814
11	a11. exe	738,275	1557482	1953381	1955305
12	a12. exe	1,594,276	221837	3388013	3349821
13	a13. exe	2,273,670	3876748	5386323	5358508
14	a14. exe	2,985,306	3378487	4435189	4391280
15	a15. exe	3,412,639	1765849	312451	304503
16	a16. exe	3,872,984	2018478	2859239	2529935
17	a17. exe	4,038,387	4786	8783	9015
18	a18. exe	5,284,796	4987584	6874552	6590217
19	a19. exe	5,628,037	14894756	22431762	16734368
20	a20. exe	6,735,934	46658494	66742981	34387484
Average			4104189	6169940	4096723

Comparisons of Chi-Square value of *.txt* files

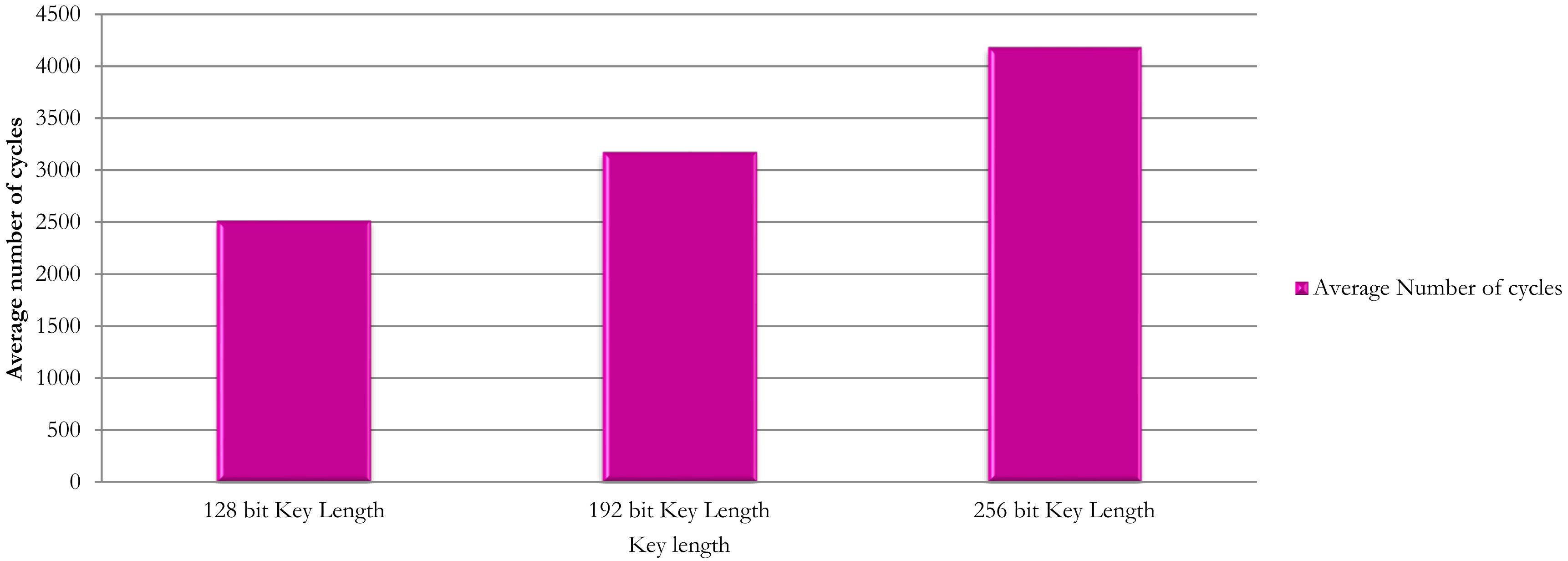
Serial no.	Source File name	Source file size (In bytes)	Chi-Square values		
			KSOMSCT	**TDES**	**AES**
1	a01.txt	1,504	31938	58385	15267
2	a02.txt	7,921	587129	1500874	347663
3	a03.txt	17,036	3809645	7721661	1731310
4	a04.txt	44.624	4923806	4709724	4753971
5	a05.txt	68,823	18569064	29704639	15663977
6	a06.txt	161,935	68865906	76621083	64043270
7	a07.txt	328,017	328096745	388539921	325837900
8	a08.txt	587,290	1364538932	1258362670	1082315460
9	a09.txt	1,049,763	2965423187	3264221211	2585100024
10	a10.txt	1,418,025	8237908765	5896971610	5524089746
11	a11.txt	1,681,329	7941894390	9087072783	8355902146
12	a12.txt	2,059,318	12967095437	11627270156	11387797334
13	a13.txt	2,618,492	25839096738	24260650965	21978420834
14	a14.txt	3,154,937	28634908759	32021906499	29332709650
15	a15.txt	4,073,829	53867340987	47346524666	44660520923
16	a16.txt	4,936,521	52412907645	57698683717	49783638147
17	a17.txt	5,125,847	56164389079	72922461490	64846889153
18	a18.txt	5,593,219	88954120953	81707468147	77543081318
19	a19.txt	5,898,302	126390854880	101228345379	89456481325
20	a20.txt	6,702,831	148280978453	122325249286	109577386113
Average			30722317122	28557702243	25826336277

Comparisons of Chi-Square value of *.doc* files

Serial no.	Source File name	Source file size (In bytes)	Chi-Square values		
			KSOMSCT	TDES	AES
1	a01. doc	21,052	6584038	18918008	14182910
2	a02. doc	33,897	2890458	9503676	4431277
3	a03. doc	45,738	1832039	9361015	2145383
4	a04. doc	75,093	1719053	2848468	1347091
5	a05. doc	106,872	1794092	3933039	1898438
6	a06. doc	327.054	580984	537285	373599
7	a07. doc	582,831	863092	1349490	947148
8	a08. doc	729,916	5509832	5474962	4532789
9	a09. doc	1,170,251	2090482	4598604	3097778
10	a10. doc	1,749,272	24709385	41385774	27850217
11	a11. doc	2,045,805	18630942	23692555	11574426
12	a12. doc	2,372,014	13790434	18656807	11848004
13	a13. doc	2,869,275	5729084	17460853	8762683
14	a14. doc	3,161,353	9987353	9904389	6784251
15	a15. doc	3,570,295	7940973	11123554	6844351
16	a16. doc	3,834,427	6990386	7725687	5230567
17	a17. doc	4,011,986	6759037	9846653	6437662
18	a18. doc	4,562,385	6073904	7376693	5591776
19	a19. doc	4,839,102	4580856	8592223	5374094
20	a20.doc	5,472,298	5209857	8145414	6012872
Average			6713314	11021752	6763362

RESULTS AND ANALYSIS

Average Synchronization Time (in cycle) for Generating variable session key in KSOMSCT



Problems in KSOMSCT

- Overhead of parameters passing and an associated risk in terms of security.
- Required large number of neurons which takes significant amount of memory.
- No well defined terminating criteria to terminate the training of KSOFM

Double Hidden Layer Perceptron Synchronized Cryptographic Technique (DHLPSCT)

- DHLP offers two hidden layers instead of single hidden layer in TPM
- 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
- Three different learning rules are used based on the network size for faster synchronization

Double Hidden Layer Perceptron Synchronized Cryptographic Technique (DHLPSCT)

- 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.

- One of the simple and secure Simulated Annealing (SA) guided enciphering technique is used to generate the cipher text.

Structure of DHLP

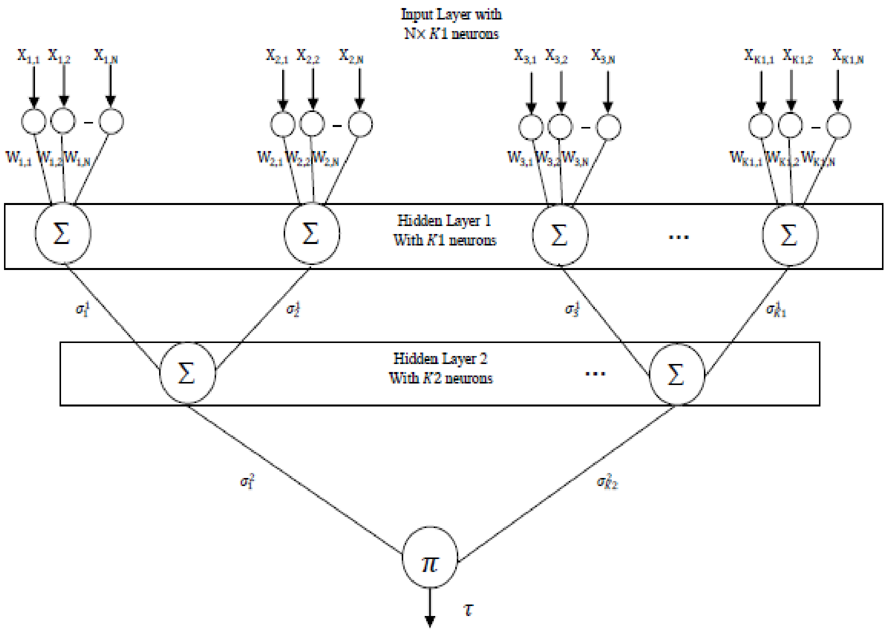


Figure: A DHLP with two hidden layers

Each hidden neuron 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 3.1 and 3.2

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

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

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

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

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

Learning rules of DHLP

Receiver update their weights where $\sigma_k^{Sender/Receiver} = \tau^{Sender/Receiver}$ using learning rules

Anti-Hebbian:

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

Hebbian:

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

Random Walk:

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

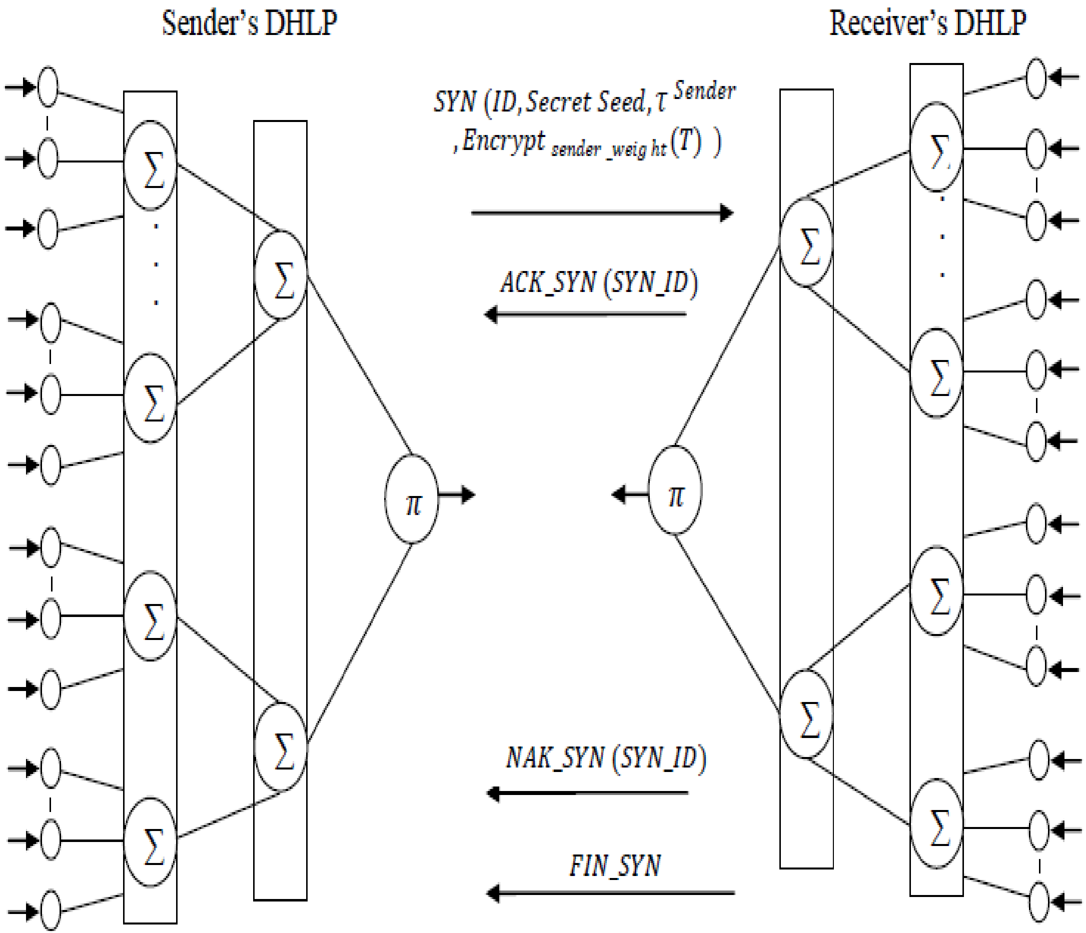


Figure: Exchange of control frames between sender and receiver during DHLP synchronization

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: Synchronization (*SYN*) frame

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

Figure: Acknowledgement of Synchronization (*ACK_SYN*) frame

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

Figure: Negative Acknowledgement of Synchronization (*NAK_SYN*) frame

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

Figure: Finish Synchronization (*FIN_SYN*) frame

SA based Encryption

Generate the initial population randomly

- The keystream (individual) in SA based technique are comprises of character sequence.

- Here, 'a'… p' represents the number 0…15. For representing 128 bit long SA based keystream 32 characters are chosen randomly among 'a'… p' characters,

- Each character represent a four bits binary number. So, one particular character may appear more than once in the sequence.

The following are examples of the chromosomes having 32 characters i.e. 128 bits:

Keystream 1: melapekabrdojenhpgdjlncmaofhjlnc
Keystream 2: ajckehpehgnbmdaofegolplacfbepfhj
Keystream 3: cpdmjalobgejafnbhlicpdamhliejcle
Keystream 4: jlakhfdpnoadflabpmfhnmanlokhgajb

Three factors are considered in the fitness evaluation of the keystream (individual). These

- *Randomness of the generated keystream (individual)*
- *Keystream (individual) period length*
- *Keystream (individual) length*

Set temperature:= 250

Fitness Calculation

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

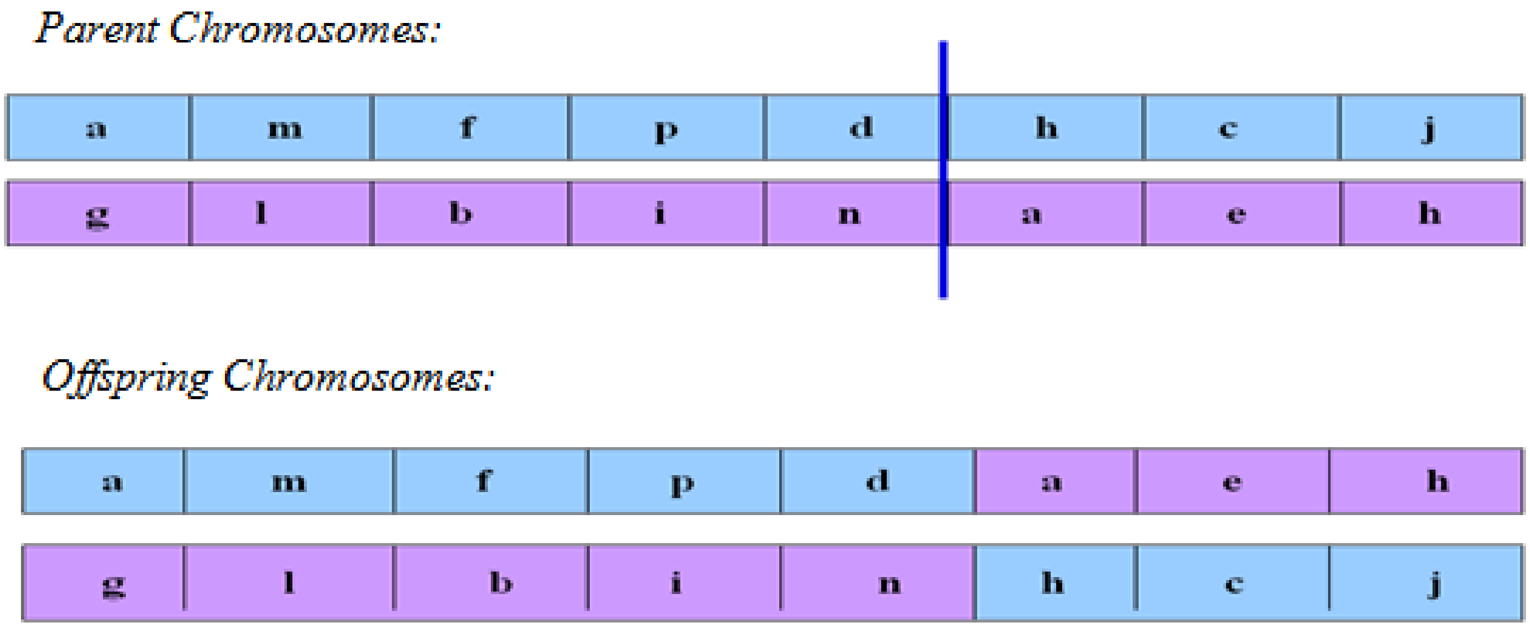
$\frac{1}{2^i} \times n_r$ of the runs in the sequence are of length i, where n_r is the number of runs in the sequence. Thus, the following equation represents the period length.

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

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

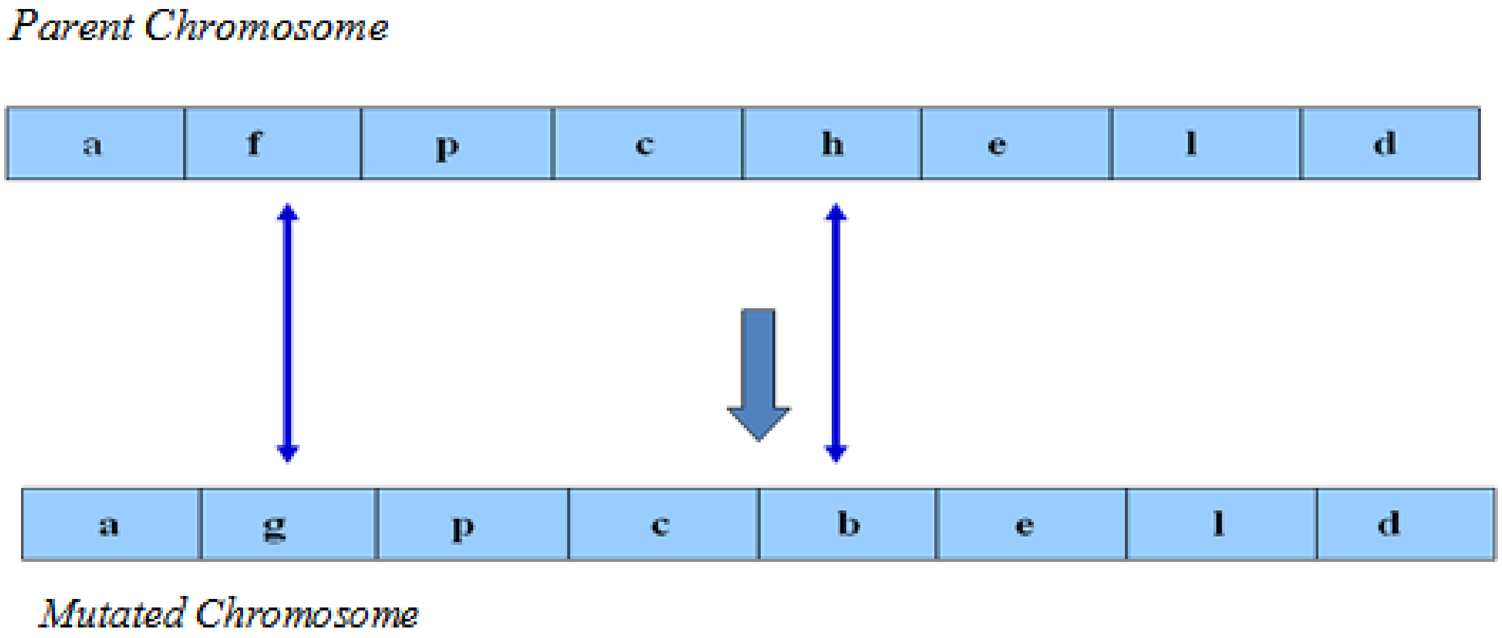
Single Point Crossover

In this technique single point crossover is performed with **probability 0.6 to 0.9.**



Mutation Operation

In this technique mutation is performed with **probability 0.001 to 0.01**.



SA based Encryption

- Evaluate the fitness of the new generated individual of pop1.
- Calculate the averages of fitness values for pop and pop1, av and av1 respectively.
- 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 $pop = pop1$.
- Set $Temp = Temp \times 0.95$
- Return the best chromosome of the final generation

Key Expansion Operation

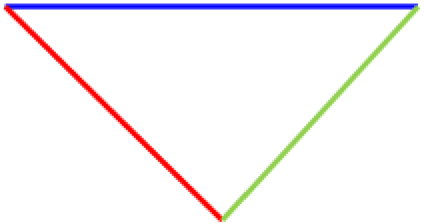
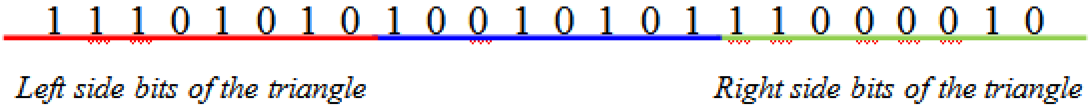
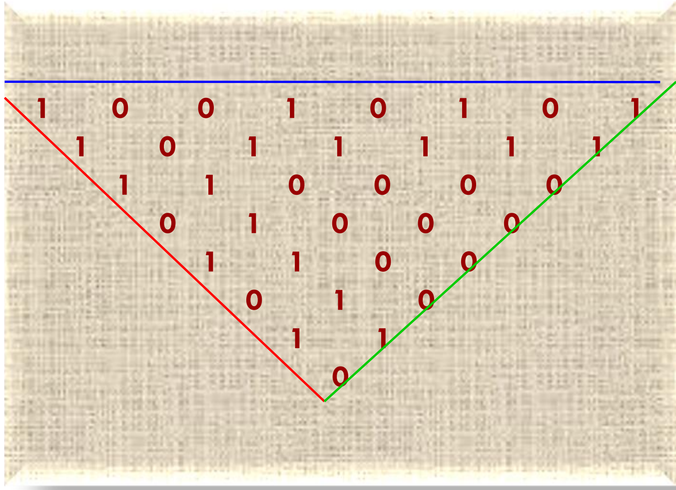


Figure 3.11: Triangle of different color sides, blue side represents the original key, red and green side represents the left and right side extended key





Example of DHLPSCT

Let the binary form of bits SA based keystream is

11111101/10101110/00011111/11011010/11010010/10000010/10101101/01100110/01001111/11101001/00001110/11110101/01010010

Consider the plaintext to be encrypted is "**SA Encryption**"

01000001/01010011/00100000/01000101/01101110/01100011/01110010/01111001/01110000/01110100/01101001/01101111/01101110

Example of DHLPSCT

Exclusive-OR encrypted text is

10010011/01010001/00110000/01001000/00101011/10001010/11010010/00010000/10110000/10111010/10000100/11100010/00100000

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)

Perform following operation as per equations on each block until the source block itself is generated.

$$s_0^j = s_0^{j-1}$$

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

Example of DHLPSCT

The formation of cycles for segments S_1 (1011110011111101) 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}→1 0110111001100100^{12}→1101101000100011^{13}→1001001111000010^{14}→ 1110001010000011^{15}→ 1011110011111101^{16}

The formation of cycles for segments S_2 (00111111100111111011110011100001). An arbitrary intermediate segment (00110000001001000000101011110001010) after iteration-22 considered as an encrypted segment for the segment S_2.

00111111100111111011110011100001→00101010111010101101011101000001^{1}→
00110011010011001001101001111110^{2}→00100010011101110001001110101011^{3}→
00111100010110100001110100110010^{4}→00101000011011000000101100010001 1^{5}→
00110000001001000000110111100001 0^{6}→00100000011100000000100101000001 1^{7}→
00111111101000000000111001111110 1^{8}→00101010110000000000101110101011 0^{9}→
00110011011111111110010110011011^{10}→00100010010101010100011011101101^{11}→
00111100011001100111101101001001^{12}→00101000010001000101001001110001^{13}→
00110000001111000011000111010000 1^{14}→00100000010100000100001011000001^{15}→
00111111100111111000000110111111 0^{16}→00101010111010101111110110101011^{17}→
00110011010011001010100100110010^{18}→00100010011101110011000111011100^{19}→
00111100010110100010000101101000^{20}→00101000011011000011111001001111^{21}→
00110000001001000000101011110001010^{22}→00100000011100000011001011110011^{23}→
00111111101000000001000110101110 1^{24}→00101010110000000001111011001011 0^{25}→
00110011011111111101011011100100^{26}→00100010010101010110010010111000^{27}→
00111100011001100100011100101111^{28}→00101000010001000111101000110101^{29}→
00110000001111000010100111101100 1^{30}→00100000010100000110001010010001^{31}→
00111111100111111011110011100001^{32}

The formation of cycles for segments S_3 (11011111). An arbitrary intermediate segment (11010010) after iteration-4 considered as an encrypted segment for the segment S_3.

$11011111 \rightarrow 10010101^{1} \rightarrow 11100110^{2} \rightarrow \mathbf{10111011^{3}} \rightarrow 11010010^{4} \rightarrow 10011100^{5} \rightarrow 11101000^{6} \rightarrow$
$10110000^{7} \rightarrow 11011111^{8}$

The formation of cycles for segments S_4 (0001111100111111). An arbitrary intermediate segment I_{47} (0001000010110000) after iteration-7 considered as an encrypted segment for the segment S_4.

$0001111100111111 \rightarrow 0001010111010101^{1} \rightarrow 0001100101100110^{2} \rightarrow 0001000110111011^{3} \rightarrow$
$0001111011010010^{4} \rightarrow 0001010010011100^{5} \rightarrow 0001100011101000^{6} \rightarrow \mathbf{0001000010110000^{7}} \rightarrow$
$0001111100100000^{8} \rightarrow 0001010111000000^{9} \rightarrow 0001100101111111^{10} \rightarrow 0001000110101010^{11} \rightarrow 0$
$001111011001100^{12} \rightarrow 0001010010001000^{13} \rightarrow 0001100011110000^{14} \rightarrow$
$0001000010100000^{15} \rightarrow 0001111100111111^{16}$

The formation of cycles for segments S_5 (1001110101100111). An arbitrary intermediate segment (1011101010000100) after iteration-6 considered as an encrypted segment for the segment S_5.

$1001110101100111 \rightarrow 1110100110111010^{1} \rightarrow 1011000100101100^{2} \rightarrow 1101111000110111^{3} \rightarrow 1001010000100101^{4} \rightarrow 1110011111000110^{5} \rightarrow \mathbf{1011101010000100^{6}} \rightarrow 1010011000001111^{7} \rightarrow 1001110111111010^{8} \rightarrow 1110100101010011^{9} \rightarrow 1011000110011101^{10} \rightarrow 1101111011101001^{11} \rightarrow 1001010010110001^{12} \rightarrow 1110011100100001^{13} \rightarrow 1011101000111110^{14} \rightarrow 1101001111010100^{15} \rightarrow 1001110101100111^{16}$

The formation of cycles for segments S_6 (10011010). An arbitrary intermediate segment (11100010) after iteration-5 considered as an encrypted segment for the segment S_6.

$10011010 \rightarrow 11101100^{1} \rightarrow 10110111^{2} \rightarrow 11011010^{3} \rightarrow 10010011^{4} \rightarrow \mathbf{11100010^{5}} \rightarrow 10111100^{6} \rightarrow 11010111^{7} \rightarrow 10011010^{8}$

The formation of cycles for segments S_7 (00111100). An arbitrary intermediate segment (00100000) after iteration-3 considered as an encrypted segment for the segment S_7.

$00111100 \rightarrow 00101000^{1} \rightarrow 00110000^{2} \rightarrow \mathbf{00100000^{3}} \rightarrow 00111111^{4} \rightarrow 00101010^{5} \rightarrow 00110011^{6} \rightarrow 00100010^{7} \rightarrow 00111100^{8}$

Example of DHLPSCT

On merging the above seven encrypted segments following SA based encrypted text is generated.
10010011/01010001/00110000/01001000/00101011/10001010/11010010/00010000/
10110000/10111010/10000100/11100010/00100000

Consider the Double Hidden Layer Perceptron (DHLP) synchronized bits session key is 00110010/11101001/10111000/11000101/00011001/01000111/00010000/01010100/
11001100/00011010/01101111/00100101/01000111/11001110/10101100/00101011

Following is the DHLP session key encrypted final cipher text
10100001/10111000/10001000/10001101/00110010/11001101/11000010/01000100/
01111100/10100000/11101011/11000111/01100111.

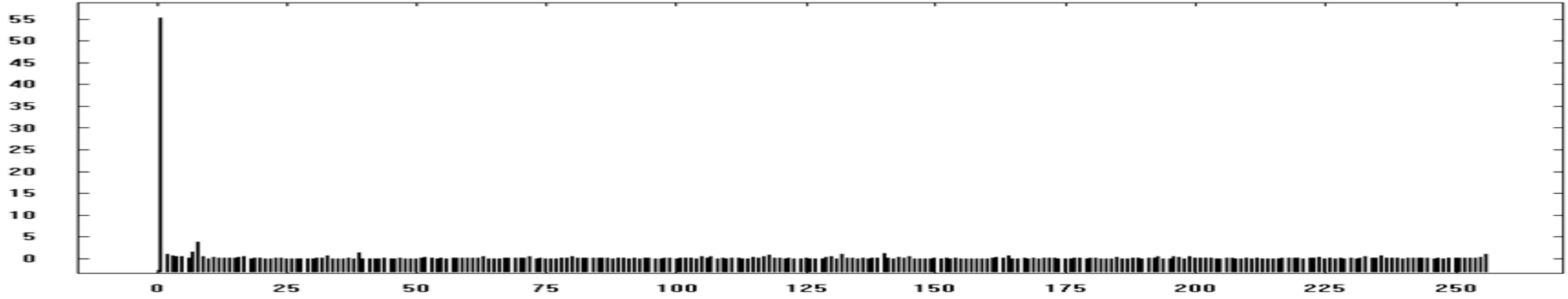


Figure : Graphical representation of frequency distribution spectrum of characters for the input *.com* source stream

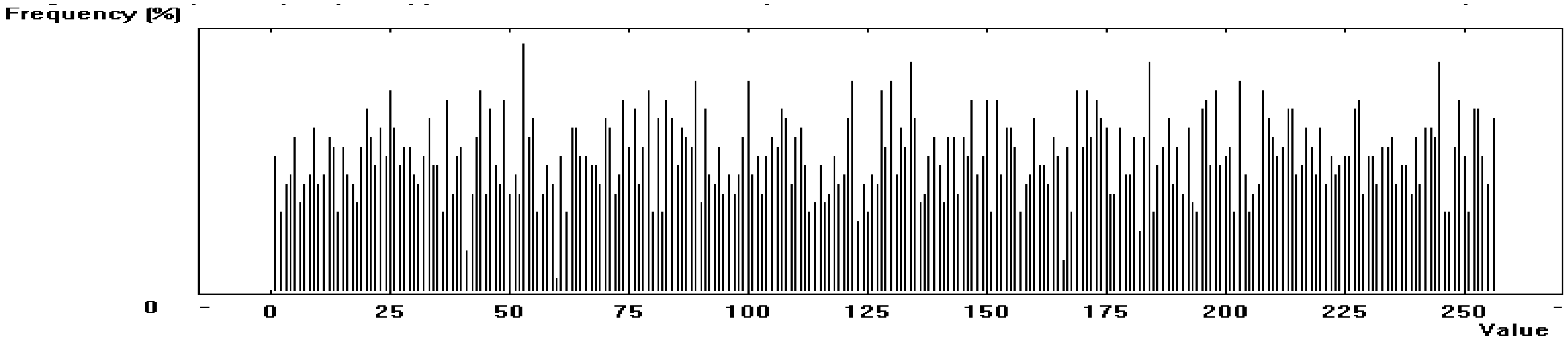


Figure : Graphical representation of frequency distribution spectrum of characters for the encrypted stream using DHLPSCT for *.com* file

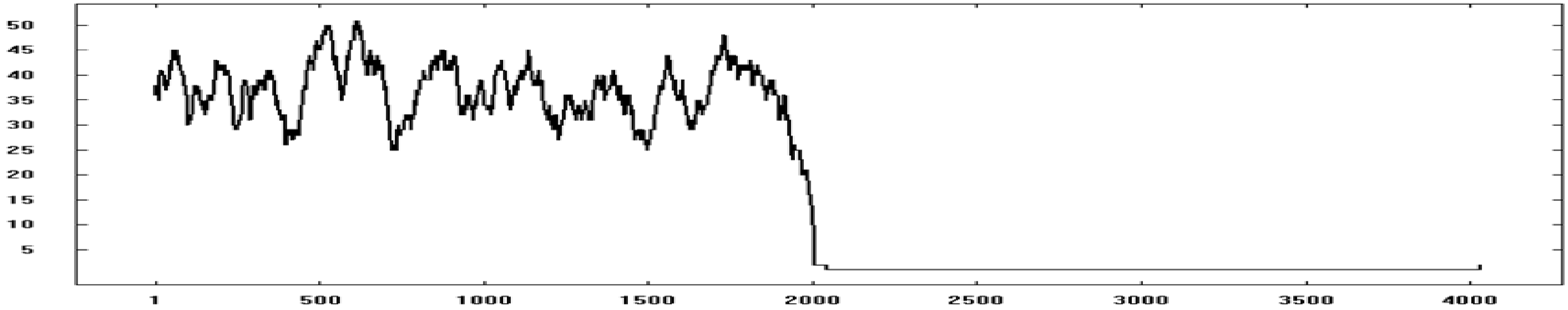


Figure : Floating frequency of the input *.com* source stream

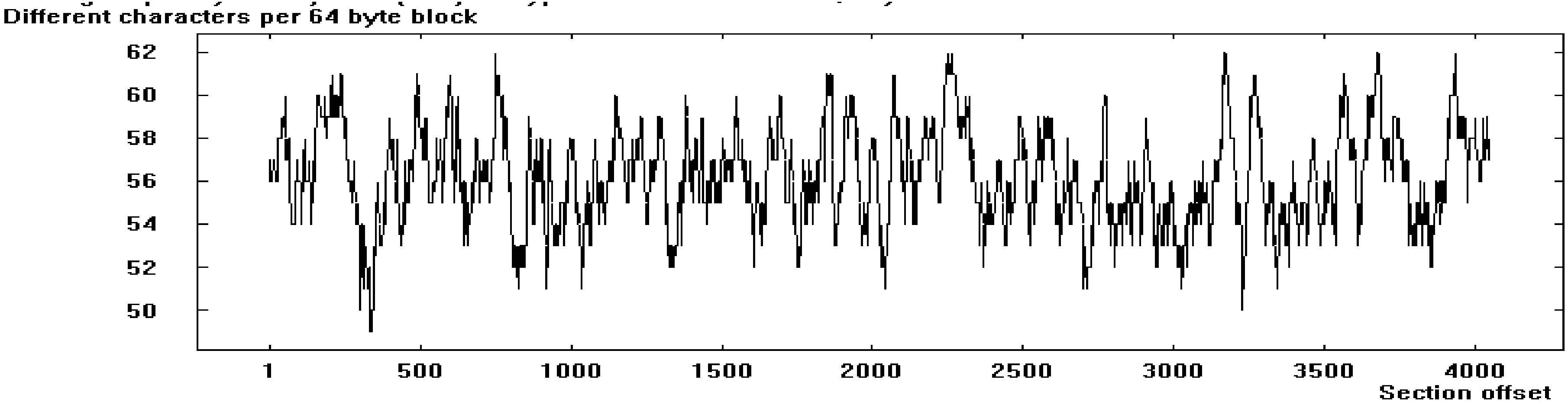


Figure : Floating frequency of the encrypted stream using DHLPSCT for *.com* file

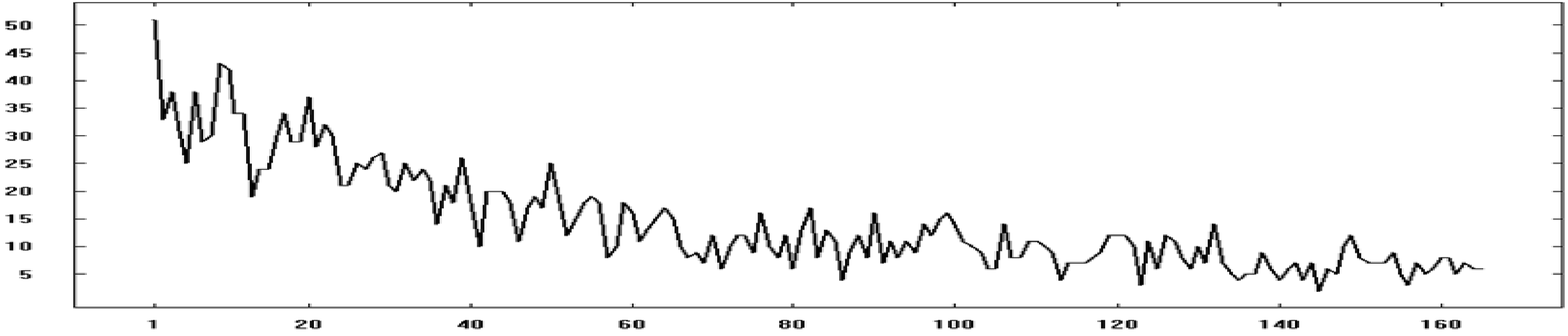


Figure : Autocorrelation of the input *.com* source stream

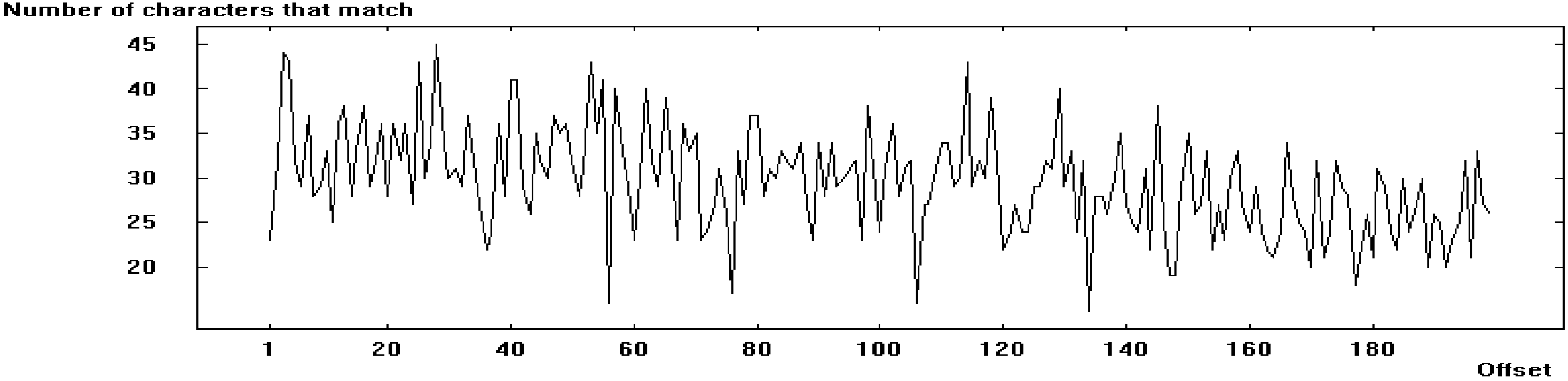


Figure : Autocorrelation of the encrypted stream using DHLPSCT for *.com* file

Comparisons of Chi-Square value of *.dll* files

Serial no.	Source File name	Source file size (In bytes)	Chi-Square values			
			DHLPSCT	KSOMSCT	TDES	AES
1	a01.dll	3216	34094	34933	36054	26036
2	a02.dll	6,656	126340	108674	193318	118331
3	a03.dll	12,288	145543	137834	165053	81475
4	a04.dll	24,576	7129562	6147653	8310677	4794027
5	a05.dll	58,784	381295	321983	806803	466466
6	a06.dll	85,020	468943	449272	654756	433872
7	a07.dll	169,472	971906	863406	1473410	1601070
8	a08.dll	359,936	735237	731276	423984	398685
9	a09.dll	593,920	1259042	1198749	1367968	1277751
10	a10.dll	909,312	1918420	1680956	2377544	2275676
11	a11.dll	1,293,824	1832907	1633962	1065999	948834
12	a12.dll	1,925,185	41264893	45172384	46245126	47627346
13	a13.dll	2,498,560	4712984	4887347	4616320	4625829
14	a14.dll	3,485,968	11939433	11590534	14567497	13560121
15	a15.dll	3,790,336	8783748	8942907	7110339	7051889
16	a16.dll	4,253,816	10321117	9215648	8451794	8194777
17	a17.dll	4,575,232	9393217	8914895	8632408	8649446
18	a18.dll	4,883,456	10872319	9912906	8866085	8450004
19	a19.dll	5,054,464	14556342	14109345	14875409	13265423
20	a20.dll	5,456,704	12498389	12673493	12432371	12239623
Average			6975581	6934661	7133646	6804334

Comparisons of Chi-Square value of *.exe* files

Serial no.	Source File name	Source file size (In bytes)	Chi-Square values			
			DHLPSCT	KSOMSCT	TDES	AES
1	a01. exe	1,063	12980	14172	31047	15349
2	a02. exe	2,518	76321	89946	167604	58911
3	a03. exe	8,250	84120	86091	3171258	1193952
4	a04. exe	15,937	103895	99387	137421	90439
5	a05. exe	22,874	26498	25985	40605	42948
6	a06. exe	35,106	269874	257394	751034	996561
7	a07. exe	52,032	110845	108746	252246	227972
8	a08. exe	145,387	630955	622467	1619619	879622
9	a09. exe	248,273	629864	618647	1188392	1206461
10	a10. exe	478,321	801097	796454	1646895	1611814
11	a11. exe	738,275	1589549	1557482	1953381	1955305
12	a12. exe	1,594,276	228948	221837	3388013	3349821
13	a13. exe	2,273,670	3975635	3876748	5386323	5358508
14	a14. exe	2,985,306	3517843	3378487	4435189	4391280
15	a15. exe	3,412,639	1820946	1765849	312451	304503
16	a16. exe	3,872,984	2285773	2018478	2859239	2529935
17	a17. exe	4,038,387	5129	4786	8783	9015
18	a18. exe	5,284,796	5276749	4987584	6874552	6590217
19	a19. exe	5,628,037	15749327	14894756	22431762	16734368
20	a20. exe	6,735,934	49786481	46658494	66742981	34387484
Average			4349141	4104189	6169940	4096723

Comparisons of Chi-Square value of *.txt* files

Serial no.	Source File name	Source file size (In bytes)	Chi-Square values			
			DHLPSCT	KSOMSCT	TDES	AES
1	a01.txt	1,504	37198	31938	58385	15267
2	a02.txt	7,921	653812	587129	1500874	347663
3	a03.txt	17,036	4425909	3809645	7721661	1731310
4	a04.txt	44.624	5956321	4923806	4709724	4753971
5	a05.txt	68,823	22129876	18569064	29704639	15663977
6	a06.txt	161,935	76438907	68865906	76621083	64043270
7	a07.txt	328,017	353890745	328096745	388539921	325837900
8	a08.txt	587,290	1549867335	1364538932	1258362670	1082315460
9	a09.txt	1,049,763	3485690453	2965423187	3264221211	2585100024
10	a10.txt	1,418,025	7549087564	8237908765	5896971610	5524089746
11	a11.txt	1,681,329	12198087654	7941894390	9087072783	8355902146
12	a12.txt	2,059,318	18767453098	12967095437	11627270156	11387797334
13	a13.txt	2,618,492	29543098734	25839096738	24260650965	21978420834
14	a14.txt	3,154,937	34529187230	28634908759	32021906499	29332709650
15	a15.txt	4,073,829	49756230987	53867340987	47346524666	44660520923
16	a16.txt	4,936,521	56867215490	52412907645	57698683717	49783638147
17	a17.txt	5,125,847	116340982395	56164389079	72922461490	64846889153
18	a18.txt	5,593,219	96490863457	88954120953	81707468147	77543081318
19	a19.txt	5,898,302	175563409174	126390854880	101228345379	89456481325
20	a20.txt	6,702,831	134895489087	148280978453	122325249286	109577386113
Average			36900009771	30722317122	28557702243	25826336277

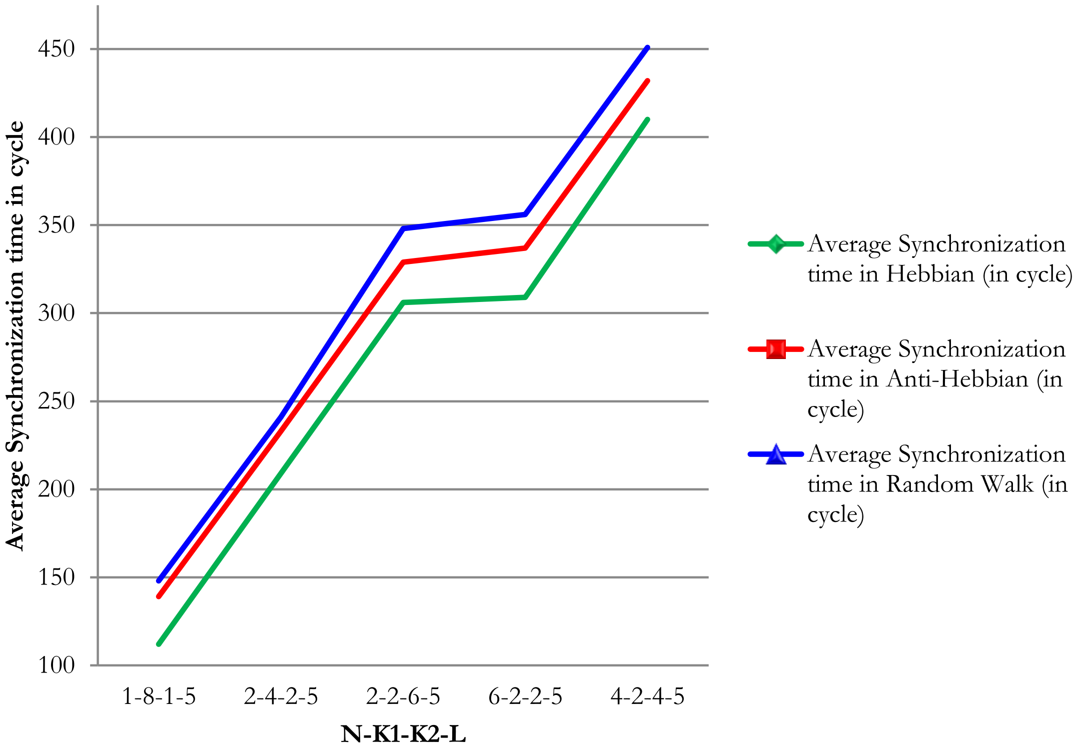
Comparisons of Chi-Square value of *.doc* files

Serial no.	Source File name	Source file size (In bytes)	Chi-Square values			
			DHLPSCT	KSOMSCT	TDES	AES
1	a01. doc	21,052	6109459	6584038	18918008	14182910
2	a02. doc	33,897	2540299	2890458	9503676	4431277
3	a03. doc	45,738	1904827	1832039	9361015	2145383
4	a04. doc	75,093	1799038	1719053	2848468	1347091
5	a05. doc	106,872	1870653	1794092	3933039	1898438
6	a06. doc	327.054	590956	580984	537285	373599
7	a07. doc	582,831	886429	863092	1349490	947148
8	a08. doc	729,916	5639042	5509832	5474962	4532789
9	a09. doc	1,170,251	2190563	2090482	4598604	3097778
10	a10. doc	1,749,272	25137093	24709385	41385774	27850217
11	a11. doc	2,045,805	19134097	18630942	23692555	11574426
12	a12. doc	2,372,014	14178095	13790434	18656807	11848004
13	a13. doc	2,869,275	6023896	5729084	17460853	8762683
14	a14. doc	3,161,353	10198431	9987353	9904389	6784251
15	a15. doc	3,570,295	8129054	7940973	11123554	6844351
16	a16. doc	3,834,427	7230986	6990386	7725687	5230567
17	a17. doc	4,011,986	6939093	6759037	9846653	6437662
18	a18. doc	4,562,385	6287235	6073904	7376693	5591776
19	a19. doc	4,839,102	4750934	4580856	8592223	5374094
20	a20.doc	5,472,298	5630939	5209857	8145414	6012872
Average			6858556	6713314	11021752	6763362

Results and analysis

Average Synchronization Time (in cycle) for Generating 128 bit Session Key using fixed Weight range (L=5) with variable Neurons in DHLPSCT

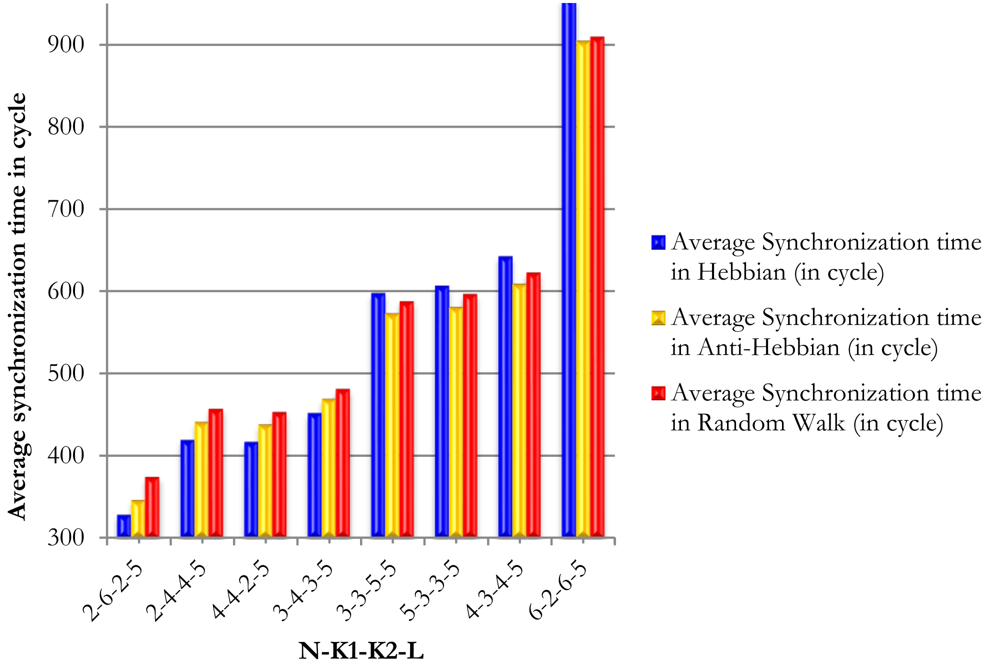
DHLP Size	N-K1-K2-L	Average Synchronization time in cycle		
		Hebbian	Anti-Hebbian	Random Walk
8	1-8-1-5	112,73	139,18	148,62
16	2-4-2-5	209,94	233,36	241,49
24	2-2-6-5	306,97	329,19	348,07
24	6-2-2-5	309,49	337,45	356,32
32	4-2-4-5	410,13	432,69	451,28



Results and analysis

Average Synchronization Time (in cycle) for Generating 192 bit Session Key using fixed Weight range (L=5) with variable Neurons in DHLPSCT

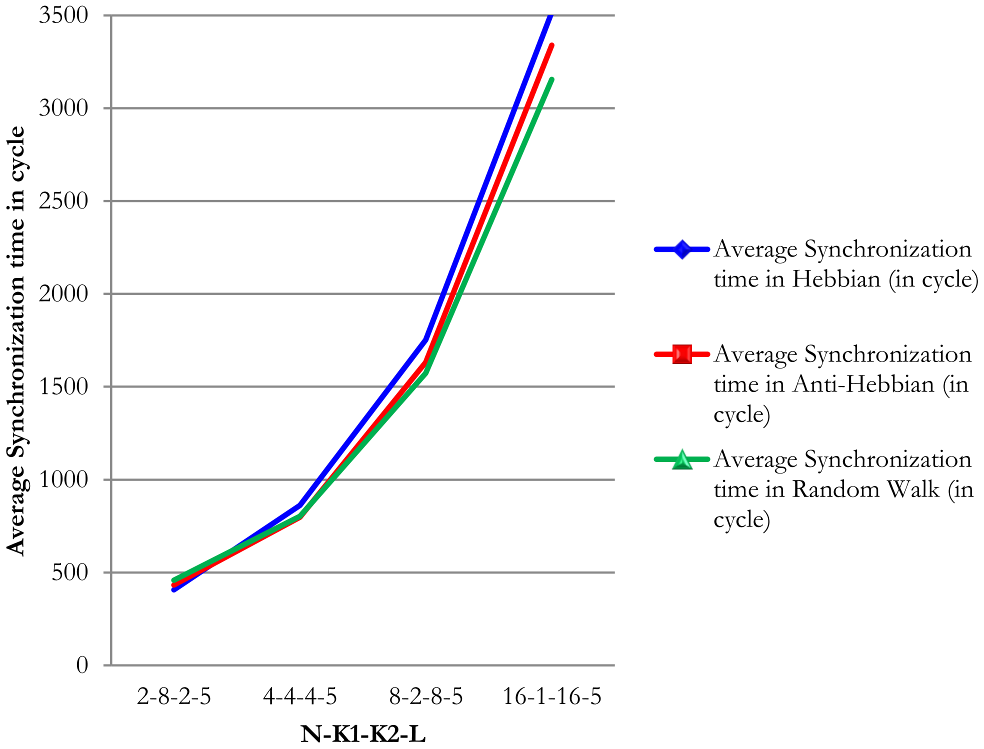
DHLP Size	N-K1-K2-L	Average synchronization time in cycle		
		Hebbian	Anti-Hebbian	Random Walk
24	2-6-2-5	328,73	346,23	374,89
32	2-4-4-5	419,26	441,64	457,08
32	4-4-2-5	417,39	438,48	453,11
36	3-4-3-5	452,73	469,91	481,03
45	3-3-5-5	598,27	573,62	588,18
45	5-3-3-5	607,42	581,83	597,39
48	4-3-4-5	643,61	609,10	623,27
72	6-2-6-5	956,71	904,28	909,37



Results and analysis

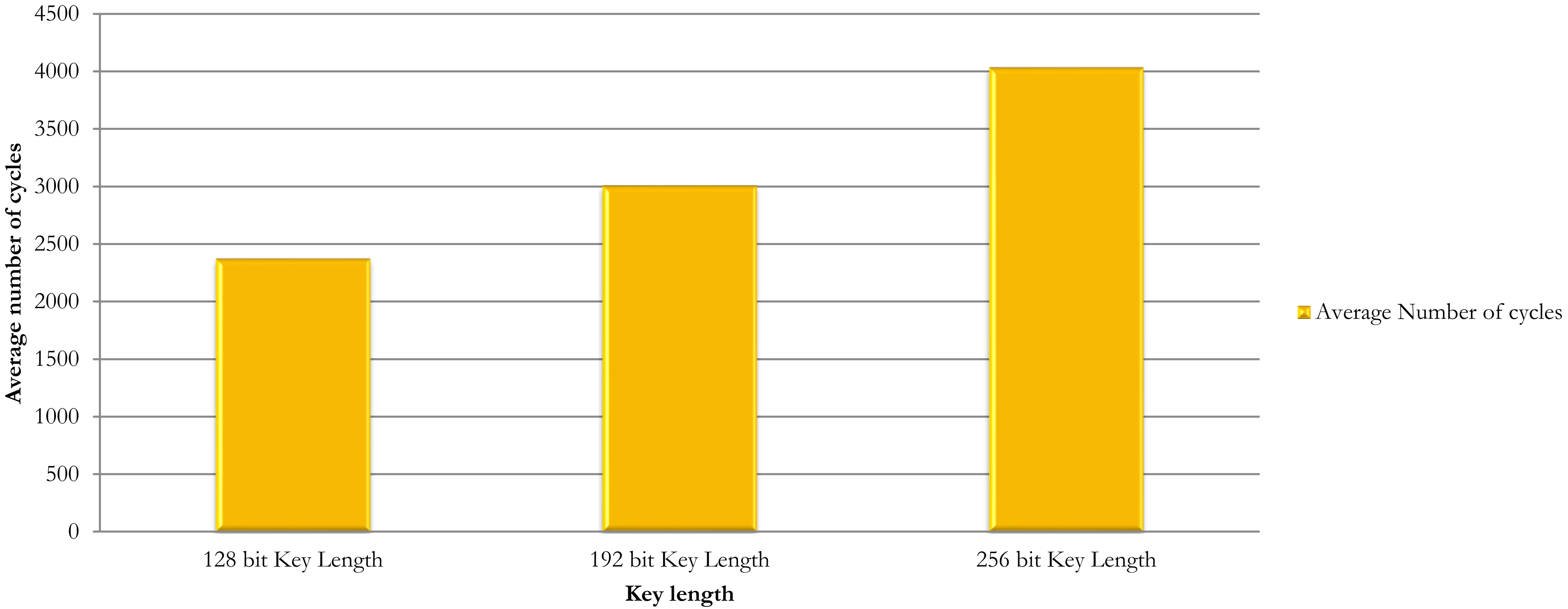
Average Synchronization Time (in cycle) for Generating 256 bit Session Key using fixed Weight range (L=5) with variable Neurons in DHLPSCT

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



Results and analysis

Average Synchronization Time (in cycle) for Generating variable session key in DHLPSCT



Problems in DHLPSCT

- Problem in generate identical random seed value for generating common input vector at the both ends.

- For ensuring the security this parameters should not be transmitted via public channel.

- Does not offer any authentication technique.

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

- Chaos is use to generate identical random seed value for generating common input vector at the both ends.
- On the completion of the tuning phase identical session keys is generated at the both end with the help of synchronized CDHLP.
- Also to ensure that only entities authorized have access to information, authentication service has been introduced in CDHLP technique.
- Simple and secure Genetic Algorithm (GA) guided enciphering technique is proposed.

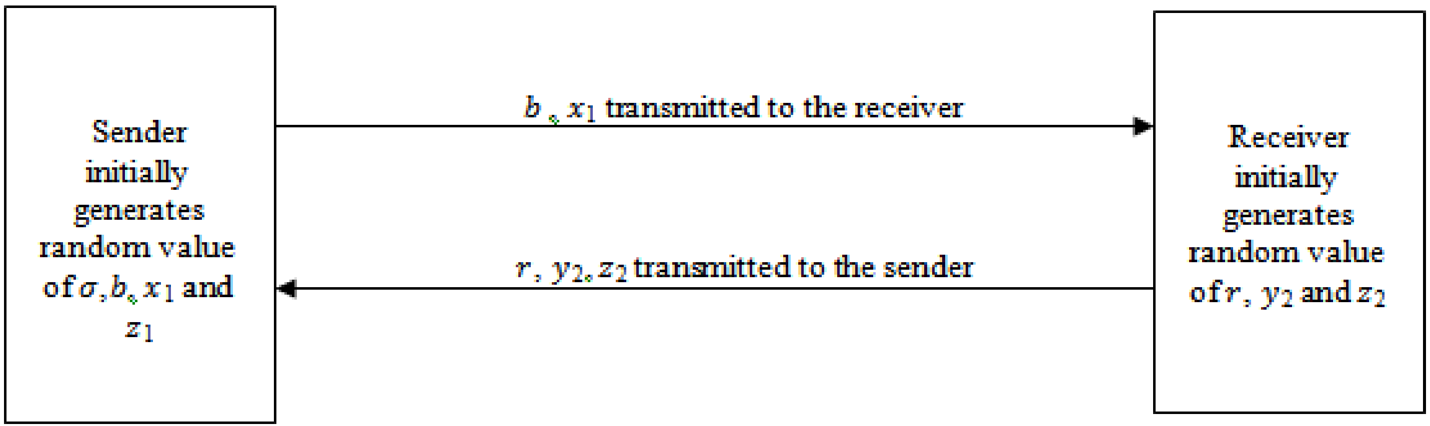


Figure: Exchange of values between sender and receiver at the initial state

The initiator (sender) ($\dot{x}_1$, $\dot{z}_1$), can be defined by following equations.

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

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

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

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

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

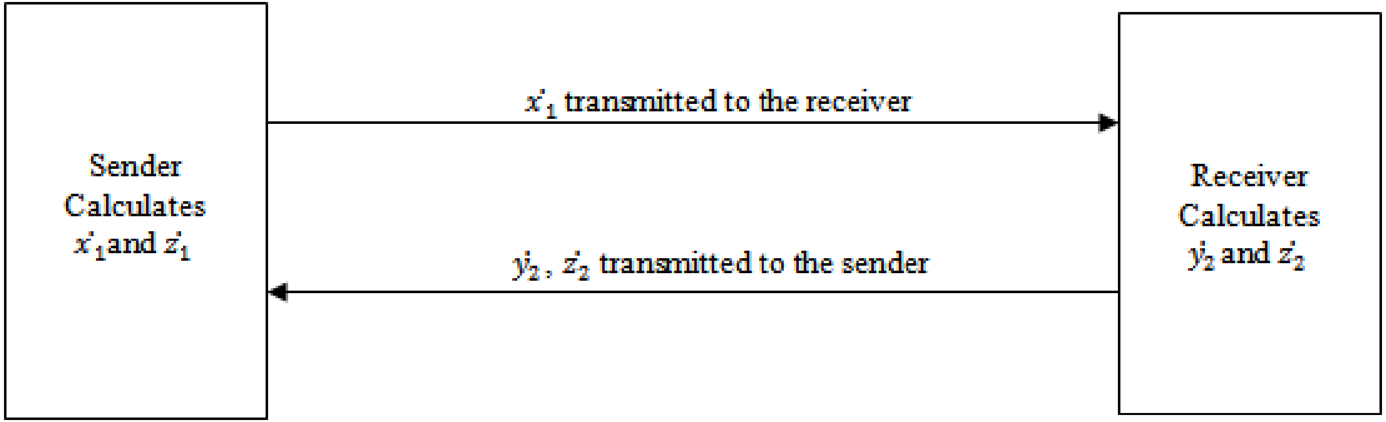


Figure: Exchange of updated values of the parameters x'_1, y'_2 and z'_2

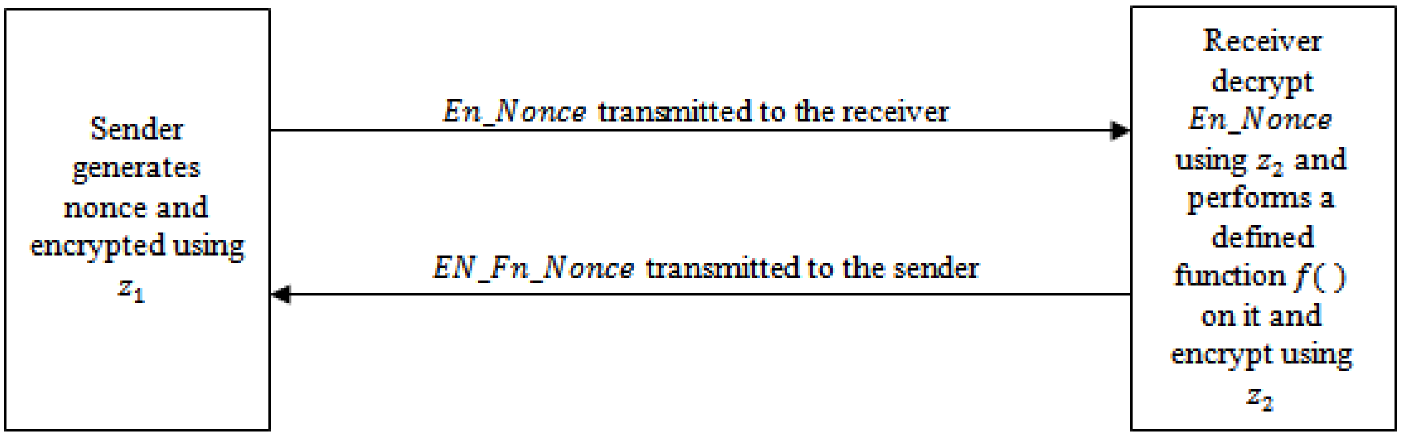


Figure: Exchange of En_Nonce and EN_Fn_Nonce

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

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

$$Fn\_Nonce = f(De\_Nonce)$$

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

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

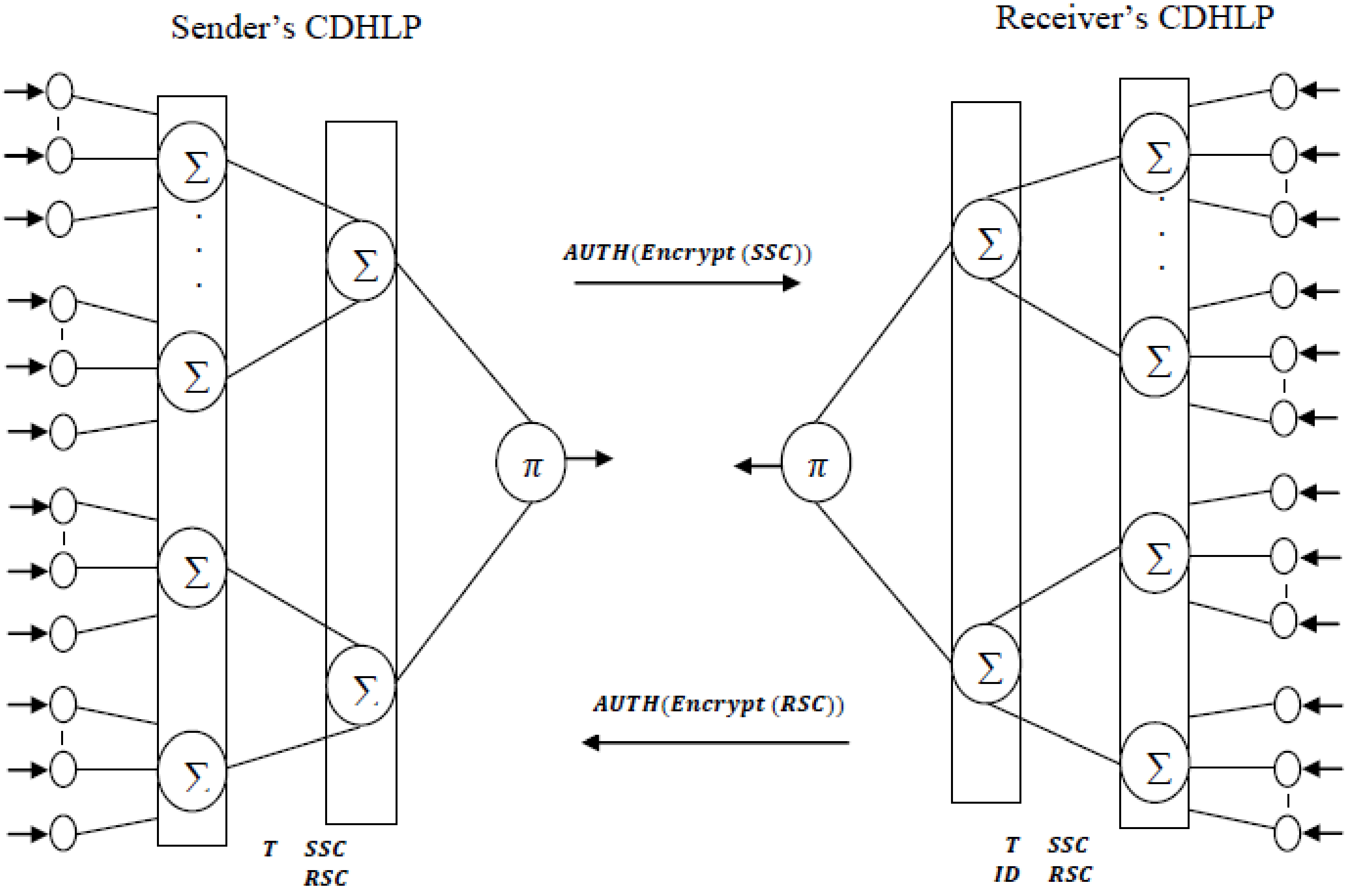


Figure: Exchange of authentication frame during session key certification phase

GA based Encryption

The following are examples of the chromosomes :

Chromosome 1: SRahij

Chromosome 2: ΛSRmchpSRncoe

Chromosome 3: SRahjflpdmobenka

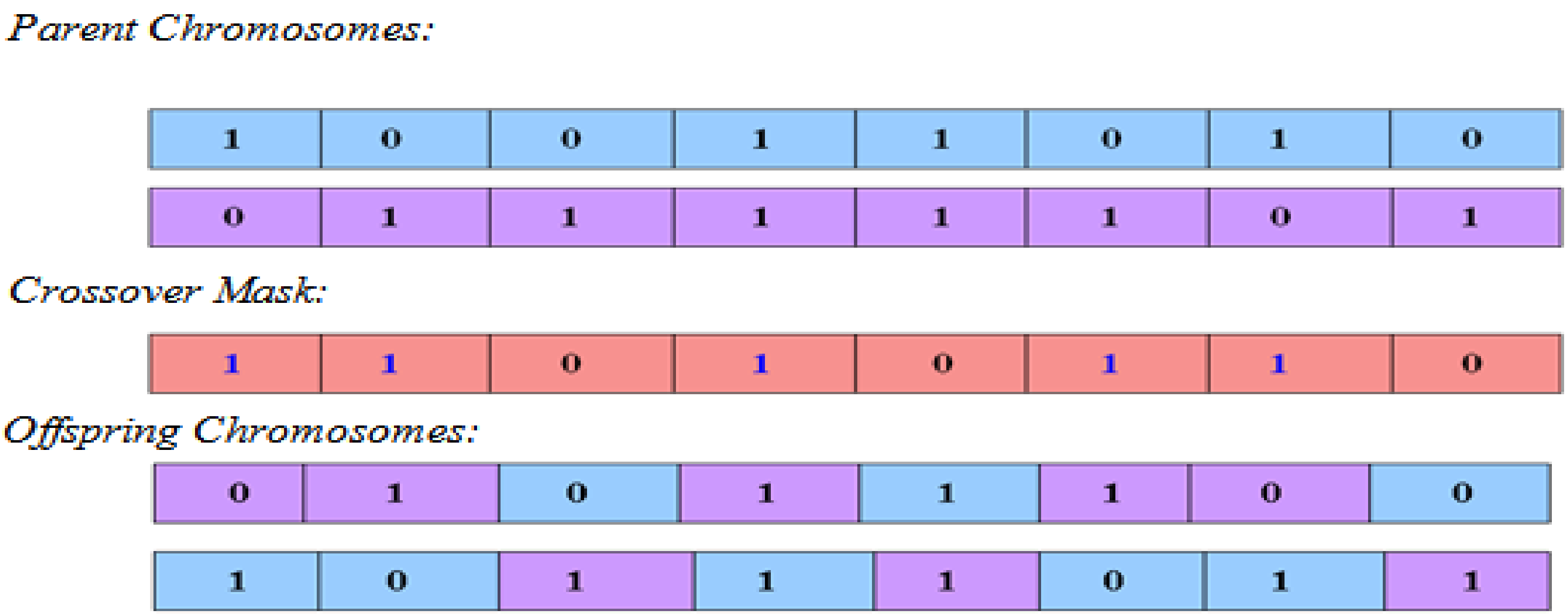
Chromosome 4: |&SRajΛSRfh&|SRglncΛSRbacfSRpoSRln

Three factors are considered in the fitness evaluation of the keystream (individual). These are:

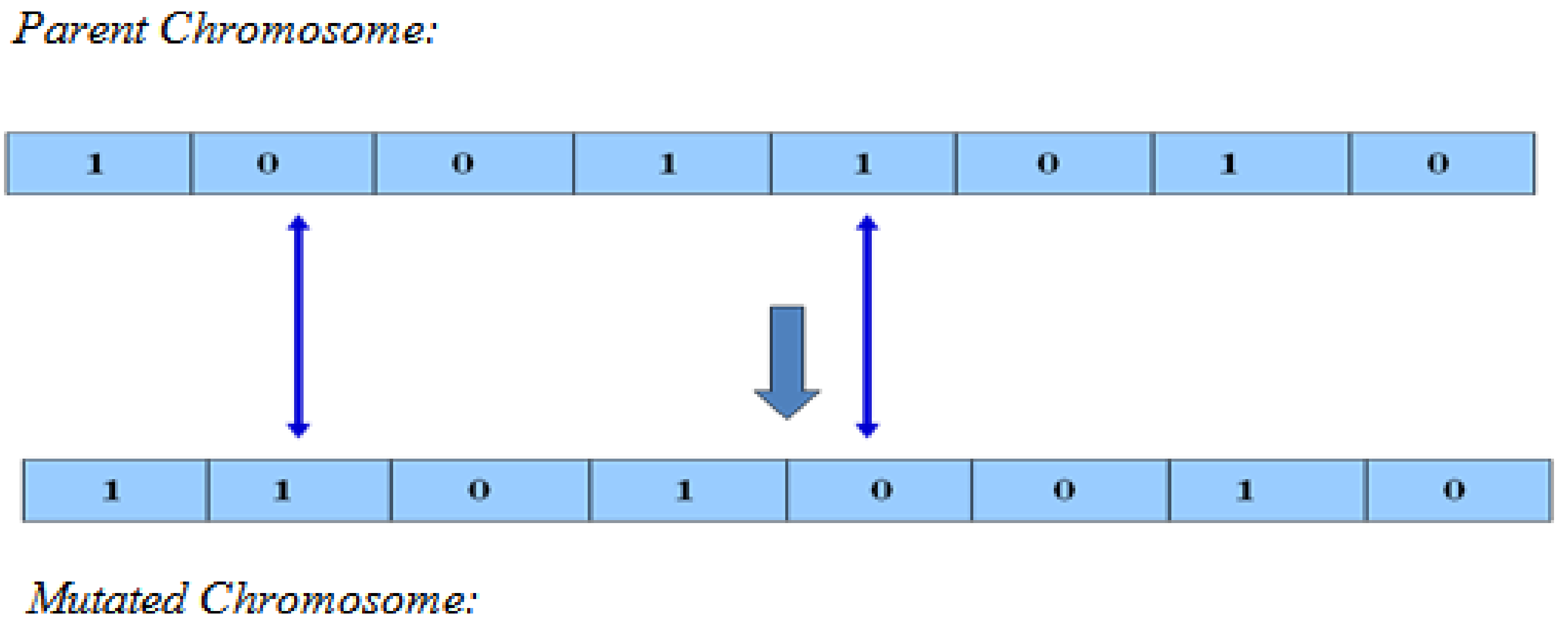
- *Randomness of the generated keystream (individual)*
- *Keystream (individual) period length*
- *Keystream (individual) length*

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

Uniform Crossover operation



Mutation Operation



Crossover and Mutation Probability

$$\text{if } (fitness \geq \max\_fitness) \text{ then}$$

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

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

$$\text{else}$$

$$Crossover\_Prob = Crossover\_Prob_1$$

$$Mutation\_Prob = Mutation\_Prob_1$$

$Crossover\_Prob_1 = 1.0$, $Crossover\_Prob_2 = 0.7$

$Mutation\_Prob_1 = 0.2$, $Mutation\_Prob_2 = 0.01$.

The maximum chromosome length is 300 characters

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.

Example of Encryption

Let the binary form of bits GA based keystream is
10011011/01011110/11001101/10010111/01010100/11010001/10101010/10110011/0100
1010/01110001/01010101/10011100/11111001/01101110/11011111/00110101

Consider the plaintext to be encrypted is "**Network Security**
01001110/01100101/01110100/01110111/01101111/01110010/01101011/00100000/0101
0011/01100101/01100011/01110101/01110010/01101001/01110100/01111001

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

Example of Encryption

S_1 = 11010101001110111011100111100000 (32 bits)
S_2 = 00111011101000111100000110010011 (32 bits)
S_3 = 00011001000101000011011011101001 (32 bits)
S_4 = 10001011000001111010101101001100 (32 bits)

1282368353 → Corresponding Decimal Value

641184177^1 → Position of 1282368353 in the Series of Natural Odd Numbers (1 for Odd)

320592089^1 → Position of 641184177 in the Series of Natural Odd Numbers (1 for Odd)

160296045^1 → Position of 320592089 in the Series of Natural Odd Numbers (1 for Odd)

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

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

So, **T_1 =00000100011000100010001101010101001**

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

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

So, **T_2 =10110110011111000011101000010001**

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 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.

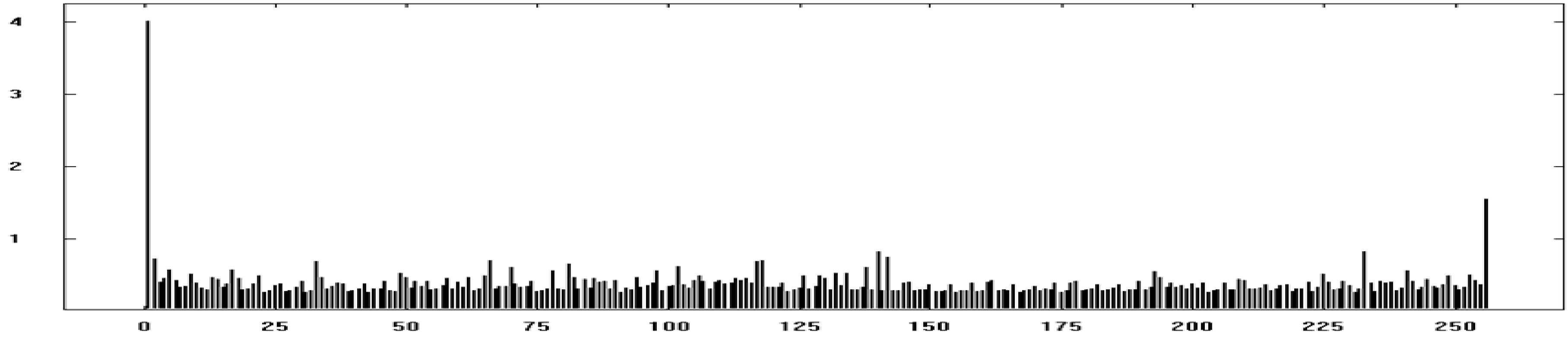


Figure : Graphical representation of frequency distribution spectrum of characters for the input .*exe* source stream

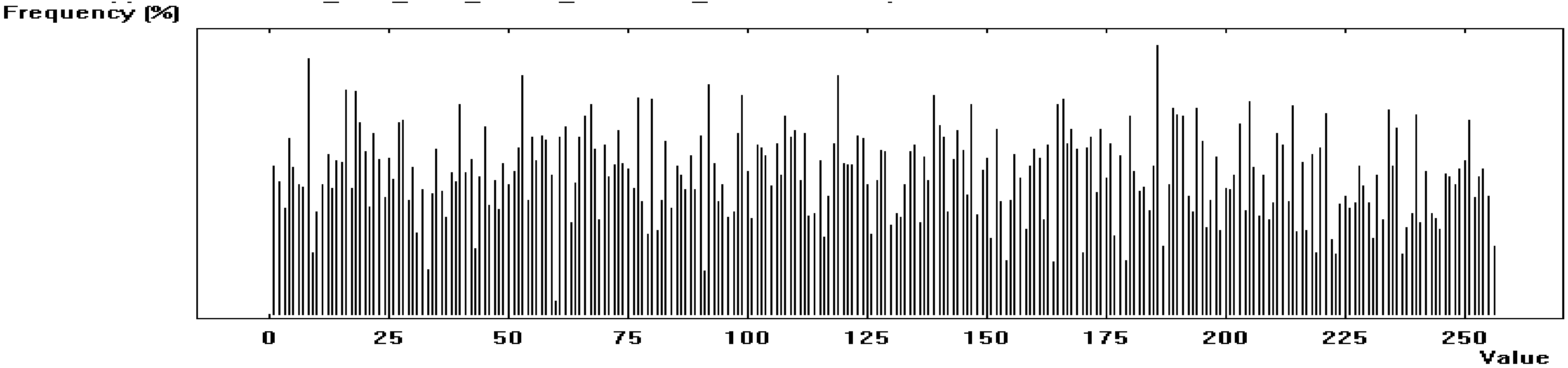


Figure : Graphical representation of frequency distribution spectrum of characters for the encrypted stream using CDHLPSCT for .*exe* file

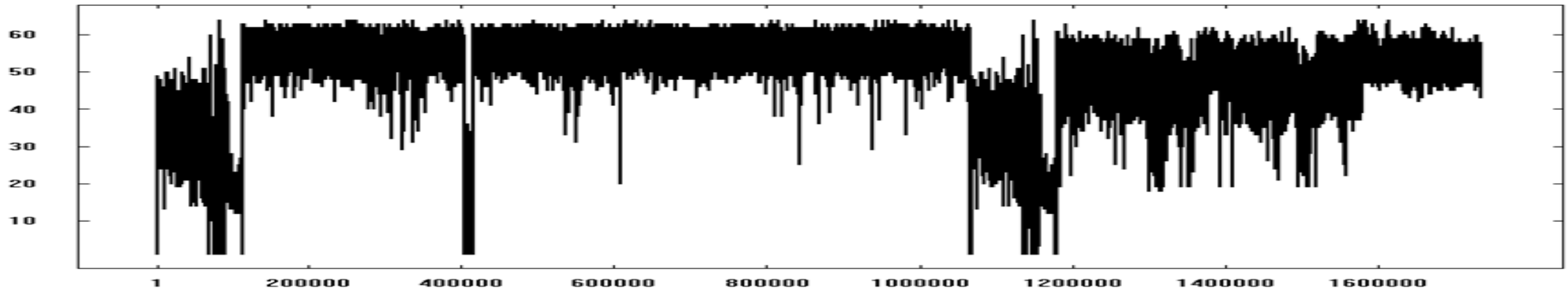


Figure : Floating frequency of the input *.exe* source stream

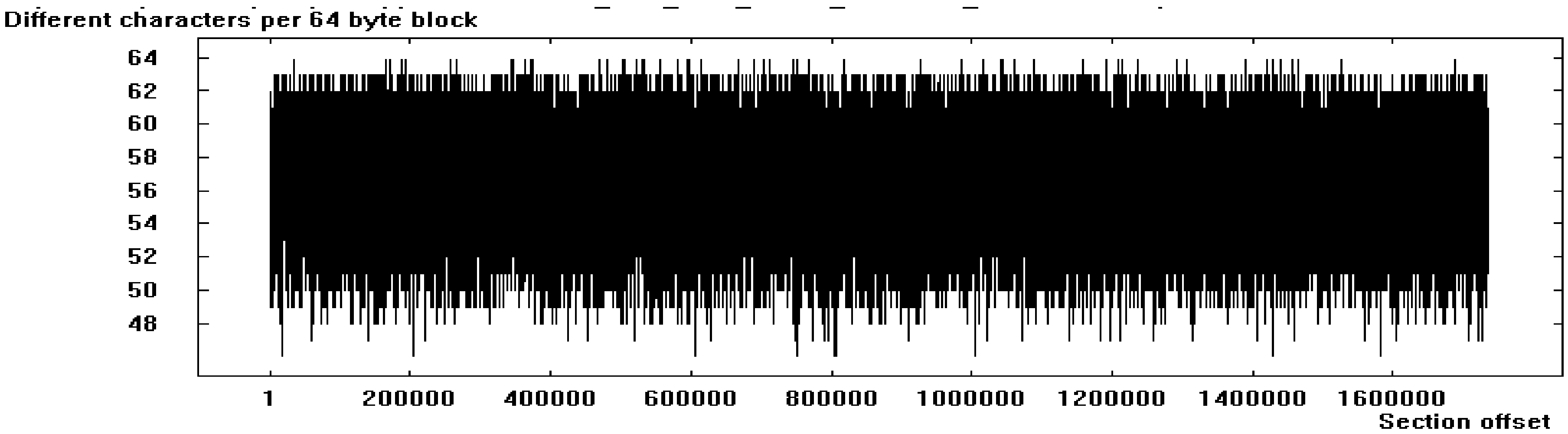


Figure : Floating frequency of the encrypted stream using CDHLPSCT for *.exe* file

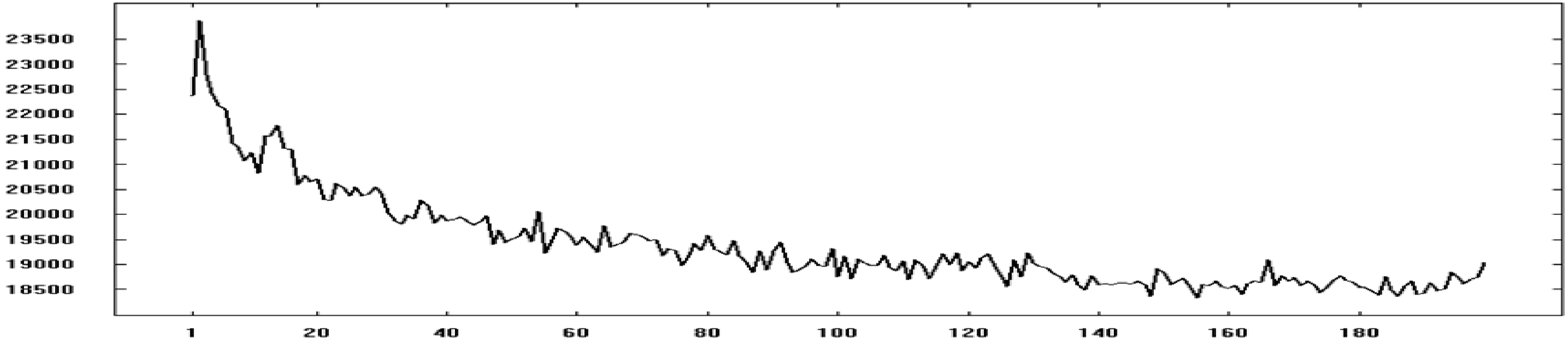


Figure : Autocorrelation of the input *.exe* source stream

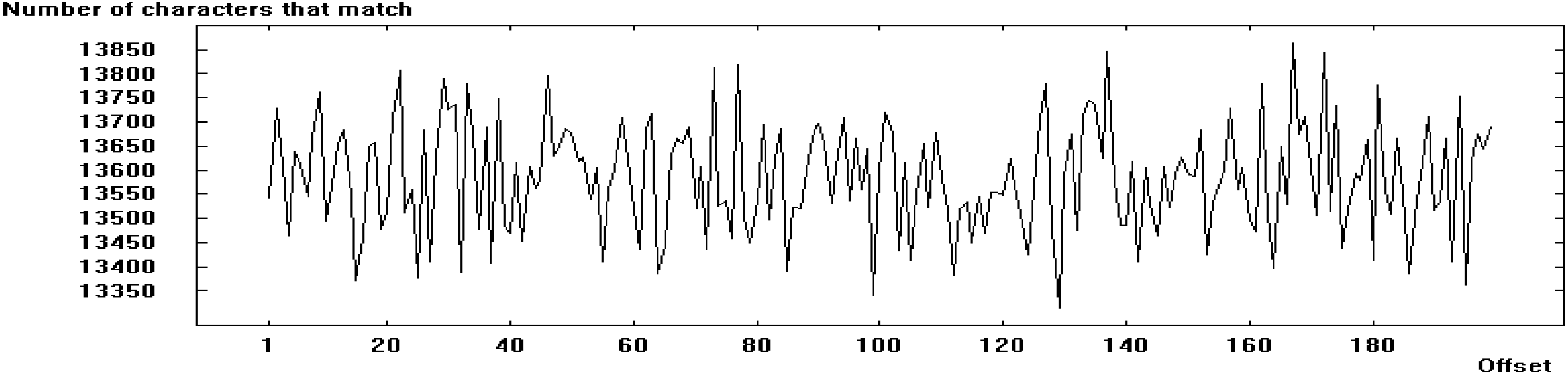


Figure : Autocorrelation of the encrypted stream using CDHLPSCT for *.exe* file

Comparisons of Chi-Square value of *.dll* files

Serial no.	Source File name	Source file size (In bytes)	Chi-Square values				
			CDHLPSCT	DHLPSCT	KSOMSCT	TDES	AES
1	a01.dll	3216	33739	34094	34933	36054	26036
2	a02.dll	6,656	128393	126340	108674	193318	118331
3	a03.dll	12,288	147306	145543	137834	165053	81475
4	a04.dll	24,576	7249089	7129562	6147653	8310677	4794027
5	a05.dll	58,784	395936	381295	321983	806803	466466
6	a06.dll	85,020	481904	468943	449272	654756	433872
7	a07.dll	169,472	996952	971906	863406	1473410	1601070
8	a08.dll	359,936	722904	735237	731276	423984	398685
9	a09.dll	593,920	1277829	1259042	1198749	1367968	1277751
10	a10.dll	909,312	2040637	1918420	1680956	2377544	2275676
11	a11.dll	1,293,824	1858428	1832907	1633962	1065999	948834
12	a12.dll	1,925,185	38922982	41264893	45172384	46245126	47627346
13	a13.dll	2,498,560	4780274	4712984	4887347	4616320	4625829
14	a14.dll	3,485,968	12031847	11939433	11590534	14567497	13560121
15	a15.dll	3,790,336	8807946	8783748	8942907	7110339	7051889
16	a16.dll	4,253,816	10952471	10321117	9215648	8451794	8194777
17	a17.dll	4,575,232	9464528	9393217	8914895	8632408	8649446
18	a18.dll	4,883,456	10963053	10872319	9912906	8866085	8450004
19	a19.dll	5,054,464	15920532	14556342	14109345	14875409	13265423
20	a20.dll	5,456,704	13784296	12498389	12673493	12432371	12239623
Average			7046365	6975581	6934661	7133646	6804334

Comparisons of Chi-Square value of *.exe* files

Serial no.	Source File name	Source file size (In bytes)	Chi-Square values				
			CDHLPSCT	DHLPSCT	KSOMSCT	TDES	AES
1	a01. exe	1,063	13806	12980	14172	31047	15349
2	a02. exe	2,518	79938	76321	89946	167604	58911
3	a03. exe	8,250	84902	84120	86091	3171258	1193952
4	a04. exe	15,937	108356	103895	99387	137421	90439
5	a05. exe	22,874	26894	26498	25985	40605	42948
6	a06. exe	35,106	272095	269874	257394	751034	996561
7	a07. exe	52,032	111875	110845	108746	252246	227972
8	a08. exe	145,387	637894	630955	622467	1619619	879622
9	a09. exe	248,273	636949	629864	618647	1188392	1206461
10	a10. exe	478,321	821674	801097	796454	1646895	1611814
11	a11. exe	738,275	1654098	1589549	1557482	1953381	1955305
12	a12. exe	1,594,276	2319485	228948	221837	3388013	3349821
13	a13. exe	2,273,670	4009856	3975635	3876748	5386323	5358508
14	a14. exe	2,985,306	3567849	3517843	3378487	4435189	4391280
15	a15. exe	3,412,639	1897485	1820946	1765849	312451	304503
16	a16. exe	3,872,984	2487650	2285773	2018478	2859239	2529935
17	a17. exe	4,038,387	5521	5129	4786	8783	9015
18	a18. exe	5,284,796	5343398	5276749	4987584	6874552	6590217
19	a19. exe	5,628,037	16654901	15749327	14894756	22431762	16734368
20	a20. exe	6,735,934	50748754	49786481	46658494	66742981	34387484
Average			4574169	4349141	4104189	6169940	4096723

Comparisons of Chi-Square value of *.txt* files

Serial no.	Source File name	Source file size (In bytes)	Chi-Square values				
			CDHLPSCT	DHLPSCT	KSOMSCT	TDES	AES
1	a01.txt	1,504	37485	37198	31938	58385	15267
2	a02.txt	7,921	785634	653812	587129	1500874	347663
3	a03.txt	17,036	4823409	4425909	3809645	7721661	1731310
4	a04.txt	44.624	7239064	5956321	4923806	4709724	4753971
5	a05.txt	68,823	27187564	22129876	18569064	29704639	15663977
6	a06.txt	161,935	92790569	76438907	68865906	76621083	64043270
7	a07.txt	328,017	423376129	353890745	328096745	388539921	325837900
8	a08.txt	587,290	1738797676	1549867335	1364538932	1258362670	1082315460
9	a09.txt	1,049,763	4849846875	3485690453	2965423187	3264221211	2585100024
10	a10.txt	1,418,025	8195645098	7549087564	8237908765	5896971610	5524089746
11	a11.txt	1,681,329	14145390986	12198087654	7941894390	9087072783	8355902146
12	a12.txt	2,059,318	18967452398	18767453098	12967095437	11627270156	11387797334
13	a13.txt	2,618,492	31912985534	29543098734	25839096738	24260650965	21978420834
14	a14.txt	3,154,937	40756340987	34529187230	28634908759	32021906499	29332709650
15	a15.txt	4,073,829	75327909654	49756230987	53867340987	47346524666	44660520923
16	a16.txt	4,936,521	81470983456	56867215490	52412907645	57698683717	49783638147
17	a17.txt	5,125,847	141907654387	116340982395	56164389079	72922461490	64846889153
18	a18.txt	5,593,219	125984609879	96490863457	88954120953	81707468147	77543081318
19	a19.txt	5,898,302	128653909645	175563409174	126390854880	101228345379	89456481325
20	a20.txt	6,702,831	148409329116	134895489087	148280978453	122325249286	109577386113
Average			41143854777	36900009771	30722317122	28557702243	25826336277

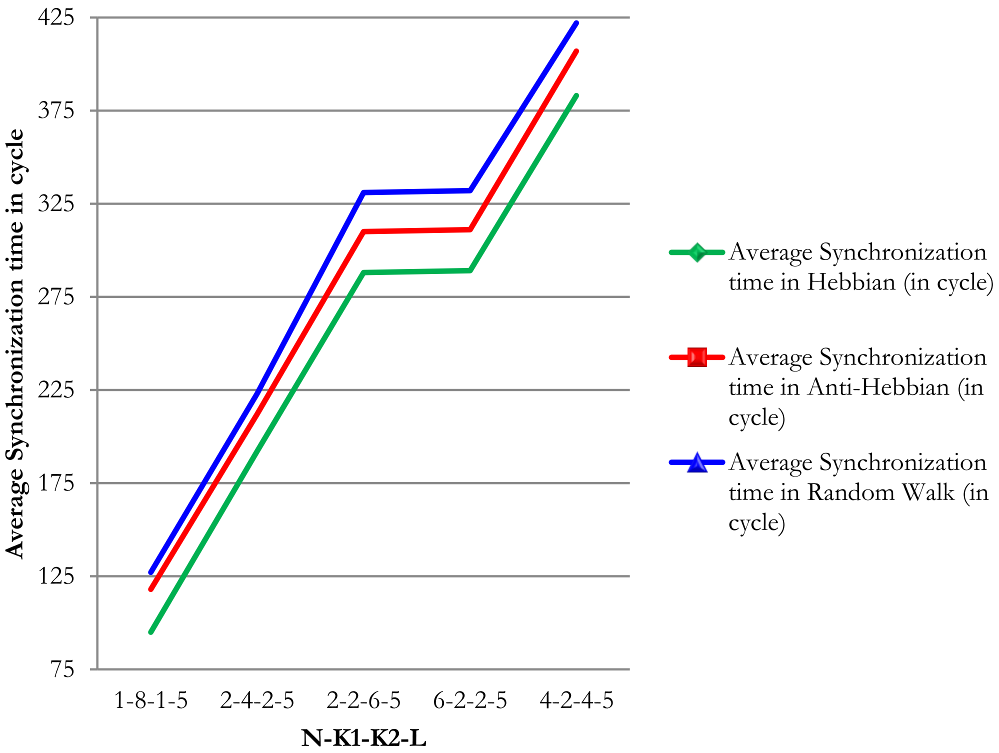
Comparisons of Chi-Square value of *.doc* files

Serial no.	Source File name	Source file size (In bytes)	Chi-Square values				
			CDHLPSCT	DHLPSCT	KSOMSCT	TDES	AES
1	a01. doc	21,052	6210846	6109459	6584038	18918008	14182910
2	a02. doc	33,897	1298305	2540299	2890458	9503676	4431277
3	a03. doc	45,738	1467406	1904827	1832039	9361015	2145383
4	a04. doc	75,093	1760847	1799038	1719053	2848468	1347091
5	a05. doc	106,872	1909562	1870653	1794092	3933039	1898438
6	a06. doc	327.054	523096	590956	580984	537285	373599
7	a07. doc	582,831	920982	886429	863092	1349490	947148
8	a08. doc	729,916	5829064	5639042	5509832	5474962	4532789
9	a09. doc	1,170,251	2426703	2190563	2090482	4598604	3097778
10	a10. doc	1,749,272	25795683	25137093	24709385	41385774	27850217
11	a11. doc	2,045,805	19504987	19134097	18630942	23692555	11574426
12	a12. doc	2,372,014	14290539	14178095	13790434	18656807	11848004
13	a13. doc	2,869,275	6109565	6023896	5729084	17460853	8762683
14	a14. doc	3,161,353	10267429	10198431	9987353	9904389	6784251
15	a15. doc	3,570,295	8534074	8129054	7940973	11123554	6844351
16	a16. doc	3,834,427	7556031	7230986	6990386	7725687	5230567
17	a17. doc	4,011,986	7031864	6939093	6759037	9846653	6437662
18	a18. doc	4,562,385	6539063	6287235	6073904	7376693	5591776
19	a19. doc	4,839,102	4804169	4750934	4580856	8592223	5374094
20	a20.doc	5,472,298	6041093	5630939	5209857	8145414	6012872
Average			6941065	6858556	6713314	11021752	6763362

Results and analysis

Average Synchronization Time (in cycle) for Generating 128 bit Session Key using fixed Weight range (L=5) with variable Neurons in CDHLPSCT

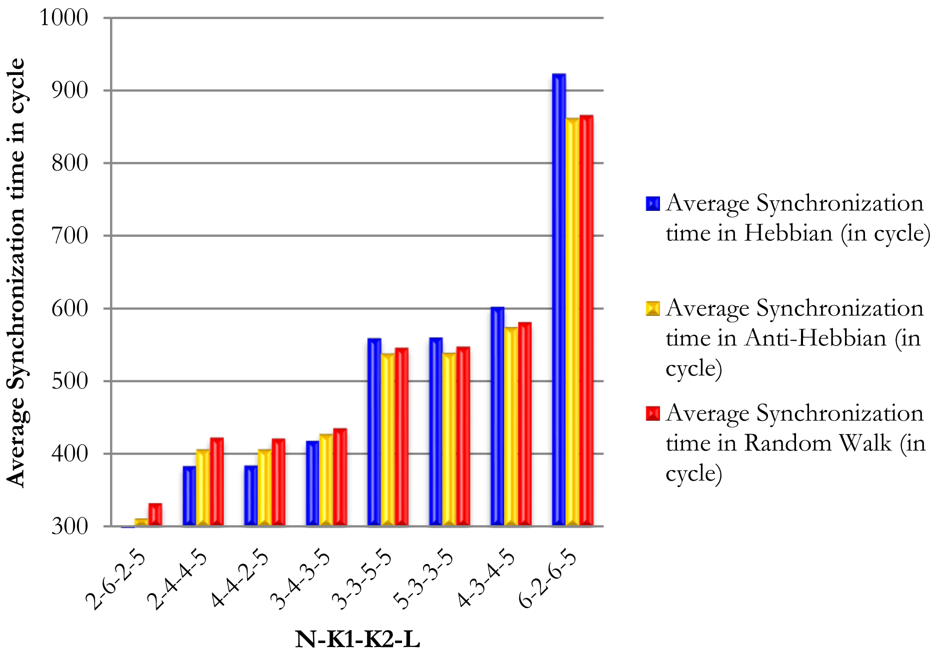
CDHLP Size	N-K1-K2-L	Average Synchronization time in cycle		
		Hebbian	Anti-Hebbian	Random Walk
8	1-8-1-5	95,42	118,73	127,12
16	2-4-2-5	192,06	212,58	223,86
24	2-2-6-5	288,13	310,79	331,14
24	6-2-2-5	289,72	311,18	332,94
32	4-2-4-5	383,96	407,05	422,89



Results and analysis

Average Synchronization Time (in cycle) for Generating 192 bit Session Key using fixed Weight range (L=5) with variable Neurons in CDHLPSCT

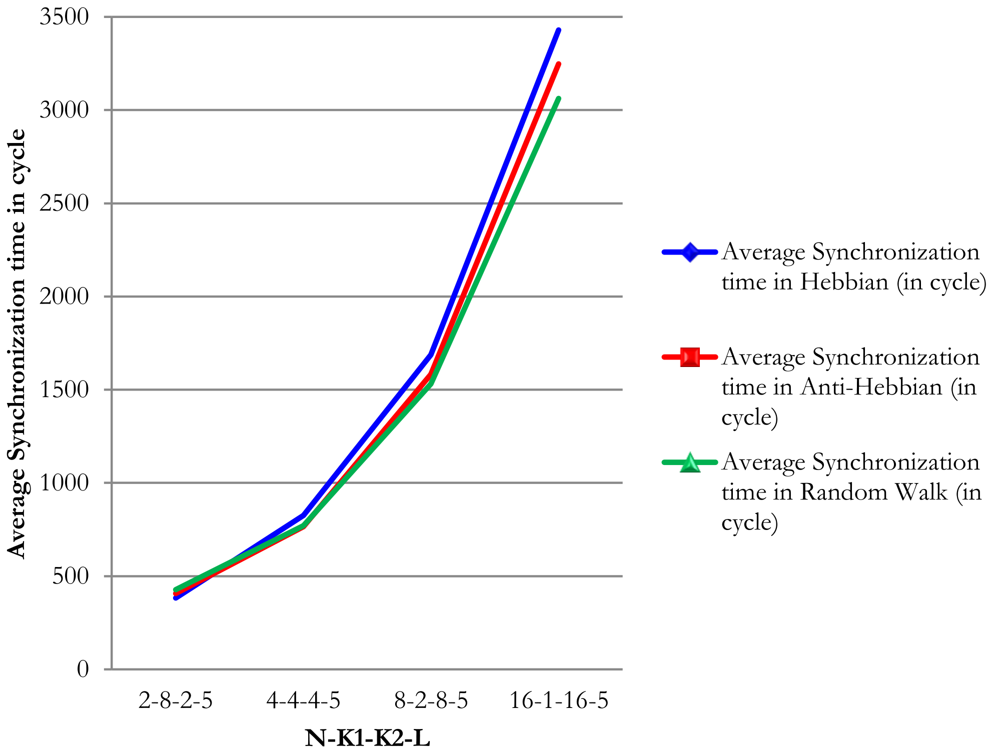
CDHLP Size	N-K1-K2-L	Average Synchronization time in cycle		
		Hebbian	Anti-Hebbian	Random Walk
24	2-6-2-5	289,35	311,62	332,14
32	2-4-4-5	383,78	406,89	422,17
32	4-4-2-5	384,10	406,15	421,08
36	3-4-3-5	418,69	427,83	435,86
45	3-3-5-5	559,18	538,59	546,13
45	5-3-3-5	560,23	539,26	547.64
48	4-3-4-5	602,76	574,35	581,08
72	6-2-6-5	923,85	862,19	866,11



Results and analysis

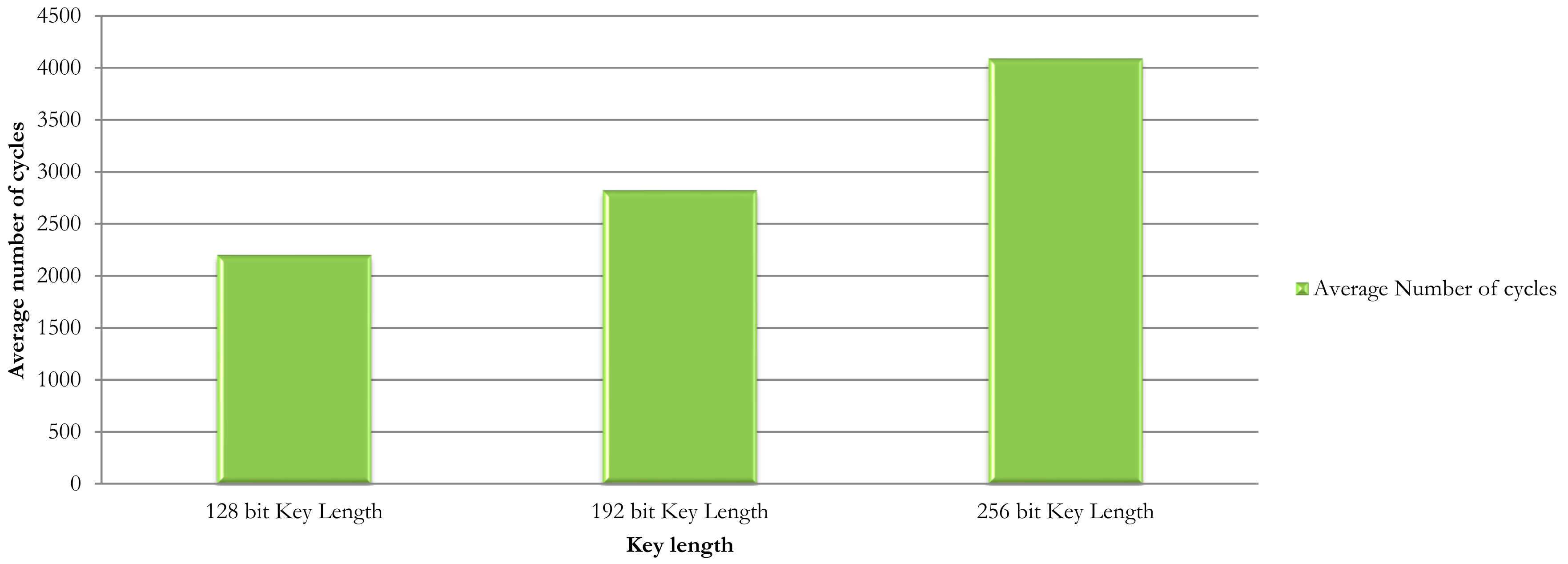
Average Synchronization Time (in cycle) for Generating 256 bit Session Key using fixed Weight range (L=5) with variable Neurons in CDHLPSCT

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



RESULTS AND ANALYSIS

Average Synchronization Time (in cycle) for Generating variable session key in CDHLPSCT



Problems in CDHLPSCT

- For increasing the length of the session key number of neurons in input layer need to be increase.
- It increase the synaptic links (weight) between input layer and hidden layer.
- It introduces the large diversity among each weight values generated randomly which may slower down the synchronization process.
- Significant amount of time is required for authentication steps.

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

- For increasing the length of the session key one additional layer is introduce instead of increasing number of neurons in input layer.
- It saves the synaptic links (weight) between input layer and hidden layer.
- It avoids the large diversity among each weight values generated randomly which may slower down the synchronization process.
- For saving the significant amount of time for authentication purpose authentication steps are performed perform parallel with the synchronization steps.

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

- On the completion of the tuning phase identical session keys is generated at the both end with the help of synchronized CTHLP.

- Simple and secure Ant Colony Intelligence (ACI) guided enciphering technique has been proposed.

Structure of CTHLPSCT

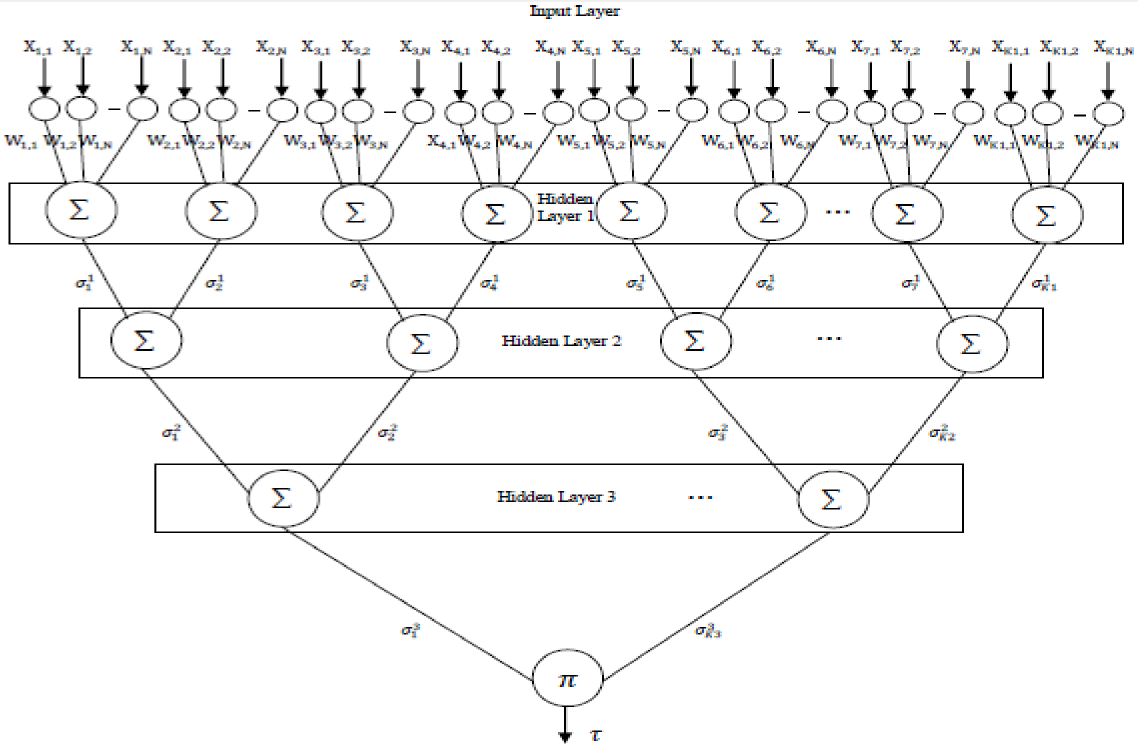


Figure: A CTHLP with three hidden layers

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

$\sigma^2_i = sgn\left(\sum_{j=1}^{N} \sigma_i^1\right)$

$\sigma^3_i = sgn\left(\sum_{j=1}^{N} \sigma_i^2\right)$

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

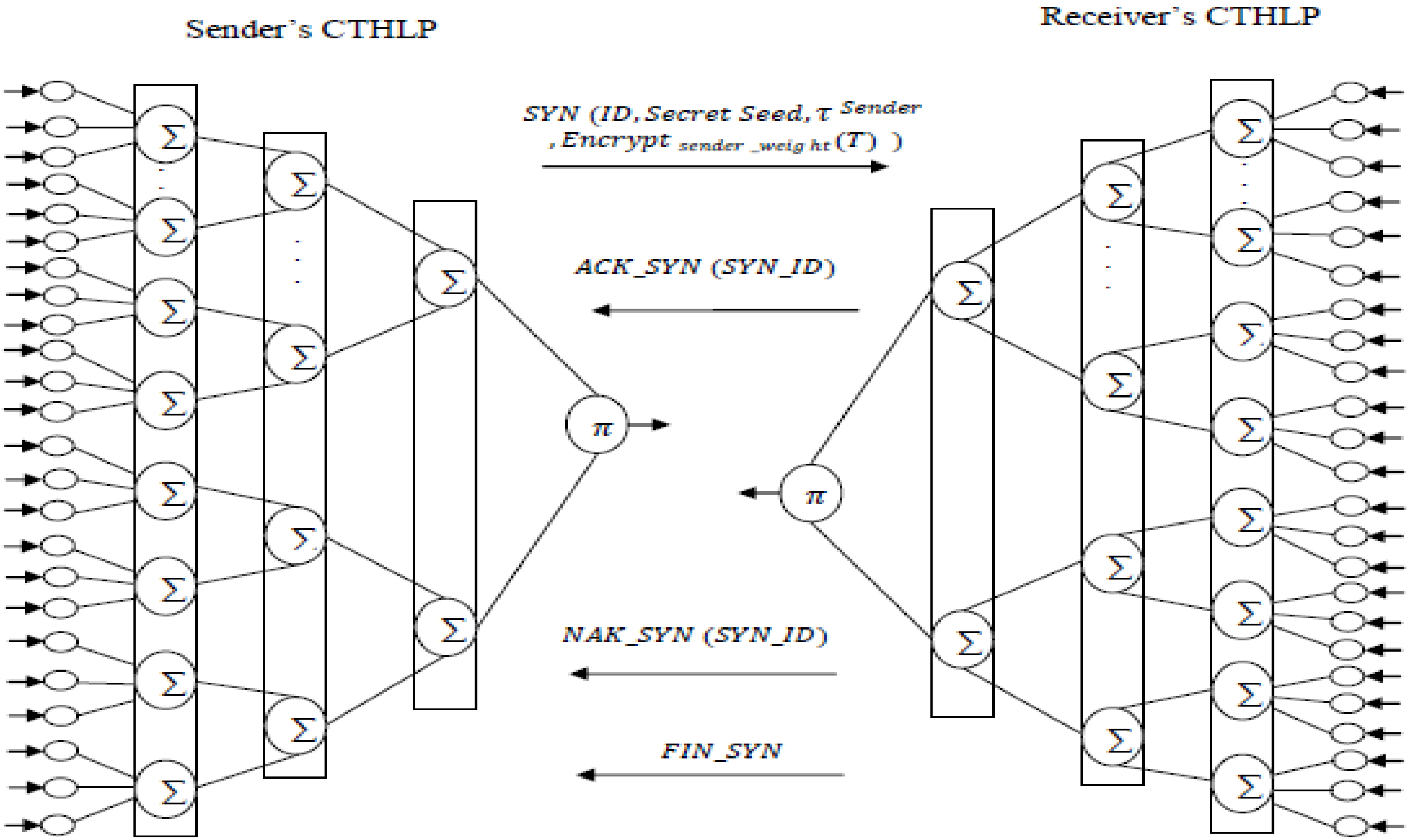


Figure: Exchange of control frames between sender and receiver during CTHLP synchronization

Example of ACI based Keystream Generation

- Consider the plaintext to be encrypted is "**antcolonyintelligence**" threshold value is assumed to be 0.65.

Keystream at Iteration 1	Energy	Keystream at 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	0.60	Highest energy	0.66

Example of Encryption

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

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

Example of Encryption

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)

Perform cycle formation techniques on $C = c_0^j c_1^j c_2^j c_3^j c_4^j \ldots c_{n-1}^j$ *of block of size n. In the following cases* $\oplus$ *is used to represents the Exclusive-OR operation. Perform the operations given in equations for generating the intermediate block*

$I_j = c_0^{j+1} c_1^{j+1} c_2^{j+1} c_3^{j+1} c_4^{j+1} \ldots c_{n-1}^{j+1}$ *from C in the following way:*

$$c_{n-1}^{j+1} = c_{n-1}^j$$

$$c_{n-2}^{j+1} = c_{n-2}^j \oplus c_{n-1}^{j+1}$$

$$c_1^{j+1} = c_1^j \oplus c_2^{j+1}$$

$$c_0^{j+1} = c_0^j \oplus c_1^{j+1}$$

The process continues for a finite number of iterations, which depends on the value of n, the source block C is regenerated.

The formation of cycles for segments S_1 (0001010000001011). 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). 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 (0001110000001101An 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). 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). 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). 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). An arbitrary intermediate segment (1111001100000010) after iteration-9 considered as an encrypted segment for the segment S_8.

0001001100000110→1111000100000010^{1}→0101000011111110^{2}→1100111110101010^{3}→
0100010101100110^{4}→1100001100100010^{5}→0100000100011110^{6}→0011111100001010^{7}→
0001010100000110^{8}→**1111001100000010**^{9}→1010111011111110^{10}→0110010110101010^{11}→
0010001101100110^{12}→1110000100100010^{13}→0101111100011110^{14}→
0011010100001010^{15}→0001001100000110^{16}

The formation of cycles for segments S_9 (00010100). An arbitrary intermediate segment (11001100) after iteration-5 considered as an encrypted segment for the segment S_9.

00010100→00001100^{1}→00000100^{2}→11111100^{3}→01010100^{4}→**11001100^{5}**→01000100^{6}→00111100^{7}→00010100^{8}

The formation of cycles for segments S_{10} (0001010000010001). An arbitrary intermediate segment (0000010000000101) after iteration-2 considered as an encrypted segment for the segment S_{10}.

0001010000010001→0000110000001111^{1}→**0000010000000101^{2}**→1111110000000011^{3}→0101010000000001^{4}→0011001111111111^{5}→0001000101010101^{6}→0000111001100111^{7}→0000010100010001^{8}→0000001100001111^{9}→0000000100000101^{10}→1111111100000011^{11}→0101010100000001^{12}→1100110011111111^{13}→0100010001010101^{14}→0011110000110011^{15}→0001010000010001^{16}

The formation of cycles for segments S_{11} (00011011 An arbitrary intermediate segment (10011001) after iteration-5 considered as an encrypted segment for the segment S_{11}.

00011011→00001001[1]→00000111[2]→11111101[3]→10101011[4]→**10011001[5]**→01110111[6]→00101101[7]→00011011[8]

The formation of cycles for segments S_{12} (00011000). An arbitrary intermediate segment (10011000) after iteration-4 considered as an encrypted segment for the segment S_{12}.

00011000→00001000[1]→11111000[2]→10101000[3]→**10011000[4]**→10001000[5]→01111000[6]→00101000[7]→00011000[8]

Example of Encryption

ACI based encrypted text is
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

Example of Encryption

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.

11000110/01001001/00010000/01111111/01001000/01000101/01010101/11000101/0110
0111/01110110/00111110/01011101/11000000/11110010/10000110/00010101/01011000
/00010001/01011010/01001110/11110110

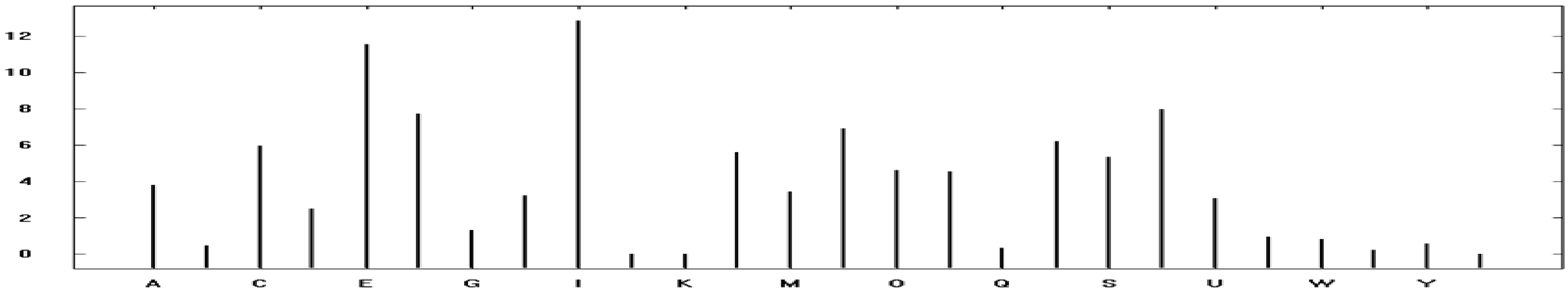


Figure : Graphical representation of frequency distribution spectrum of characters for the input *.cpp* source stream

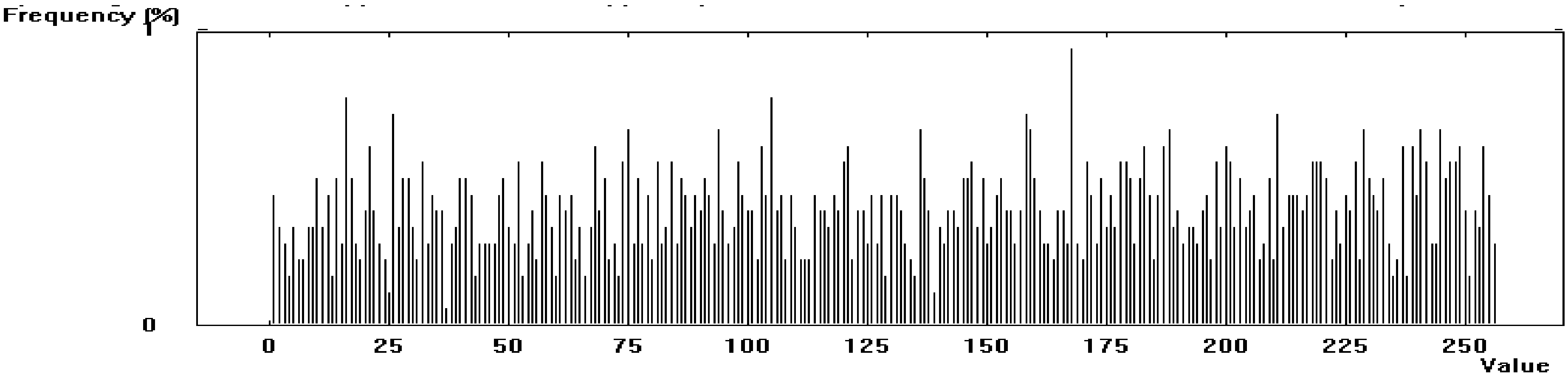


Figure : Graphical representation of frequency distribution spectrum of characters for the encrypted stream using CTHLPSCT for *.cpp* file

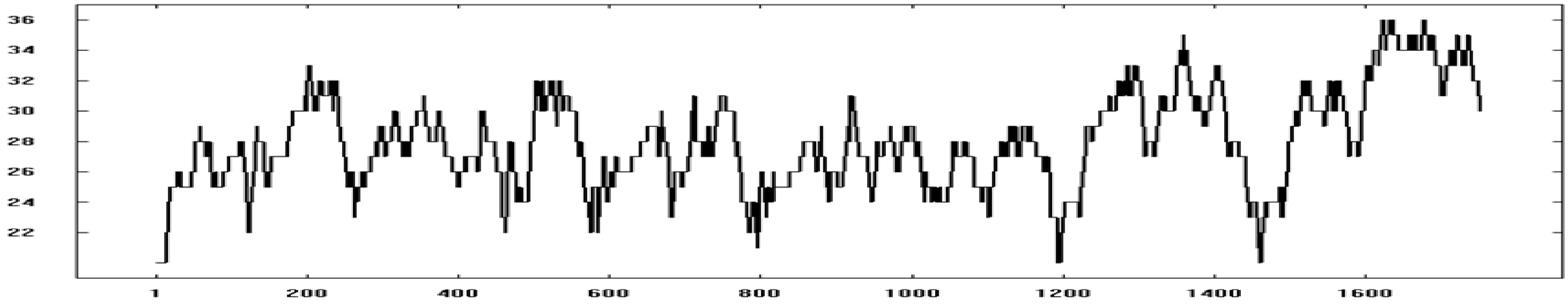


Figure : Floating frequency of the input *.cpp* source stream

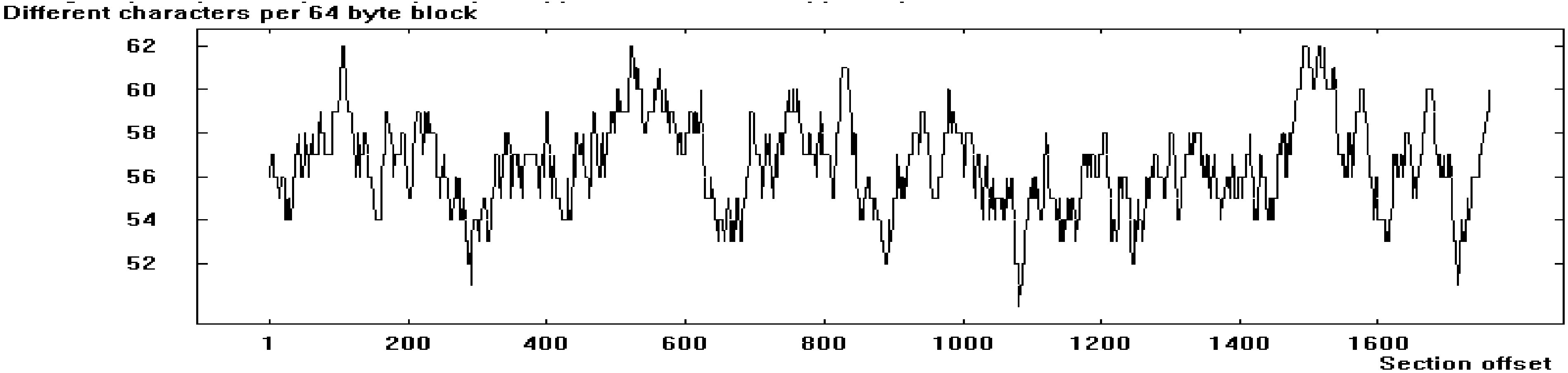


Figure : Floating frequency of the encrypted stream using CTHLPSCT for *.cpp* file

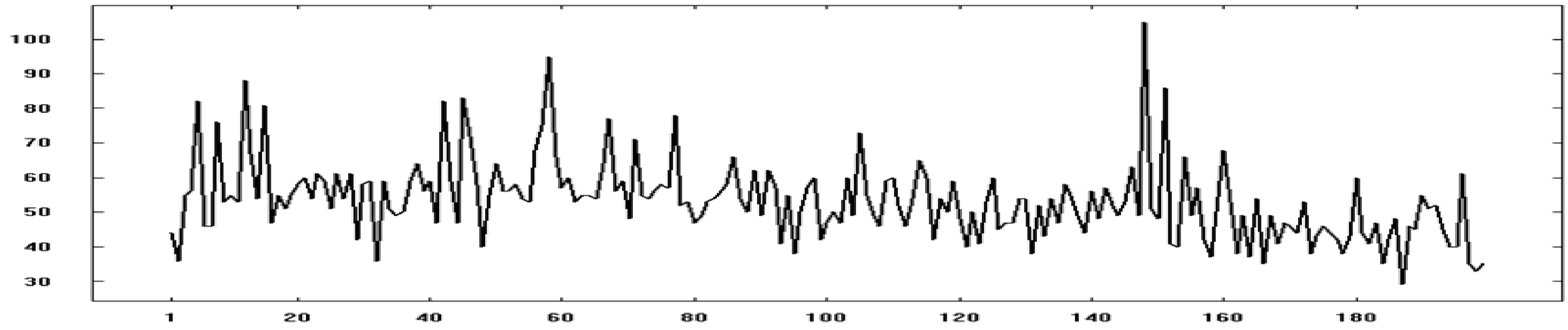


Figure : Autocorrelation of the input .*cpp* source stream

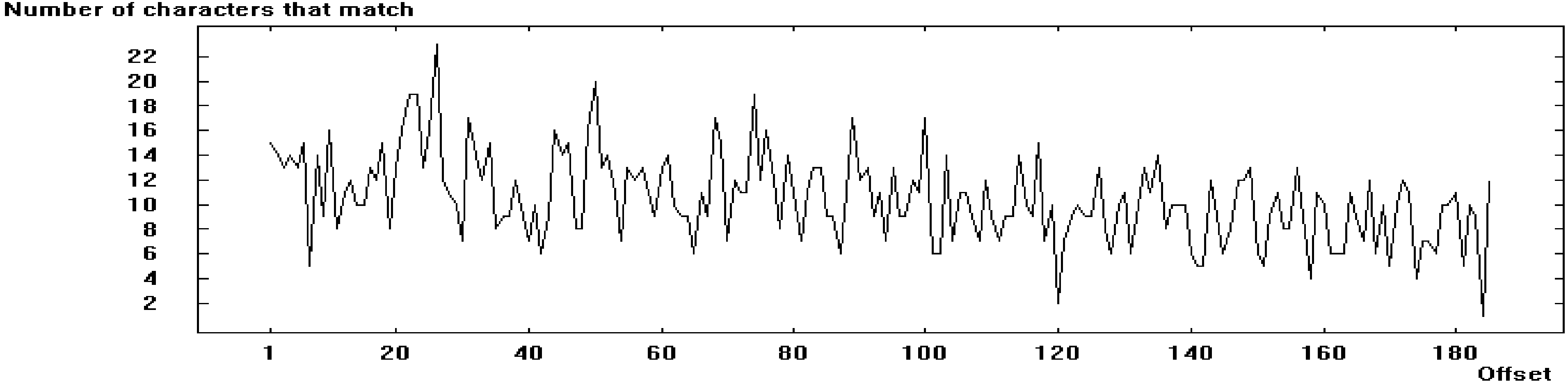


Figure : Autocorrelation of the encrypted stream using CTHLPSCT for .*cpp* file

Comparisons of Chi-Square value of *.dll* files

Serial no.	Source File name	Source file size (In bytes)	Chi-Square values					
			CTHLPSCT	CDHLPSCT	DHLPSCT	KSOMSCT	TDES	AES
1	a01.dll	3216	34083	33739	34094	34933	36054	26036
2	a02.dll	6,656	185297	128393	126340	108674	193318	118331
3	a03.dll	12,288	194032	147306	145543	137834	165053	81475
4	a04.dll	24,576	8260321	7249089	7129562	6147653	8310677	4794027
5	a05.dll	58,784	809425	395936	381295	321983	806803	466466
6	a06.dll	85,020	659304	481904	468943	449272	654756	433872
7	a07.dll	169,472	1462812	996952	971906	863406	1473410	1601070
8	a08.dll	359,936	899053	722904	735237	731276	423984	398685
9	a09.dll	593,920	1316539	1277829	1259042	1198749	1367968	1277751
10	a10.dll	909,312	2298417	2040637	1918420	1680956	2377544	2275676
11	a11.dll	1,293,824	2063494	1858428	1832907	1633962	1065999	948834
12	a12.dll	1,925,185	43290342	38922982	41264893	45172384	46245126	47627346
13	a13.dll	2,498,560	5098285	4780274	4712984	4887347	4616320	4625829
14	a14.dll	3,485,968	13884306	12031847	11939433	11590534	14567497	13560121
15	a15.dll	3,790,336	8830287	8807946	8783748	8942907	7110339	7051889
16	a16.dll	4,253,816	8472804	10952471	10321117	9215648	8451794	8194777
17	a17.dll	4,575,232	8651904	9464528	9393217	8914895	8632408	8649446
18	a18.dll	4,883,456	8910462	10963053	10872319	9912906	8866085	8450004
19	a19.dll	5,054,464	14740371	15920532	14556342	14109345	14875409	13265423
20	a20.dll	5,456,704	11806373	13784296	12498389	12673493	12432371	12239623
Average			7091691	7046365	6975581	6934661	7133646	6804334

Comparisons of Chi-Square value of *.exe* files

Serial no.	Source File name	Source file size (In bytes)	Chi-Square values					
			CTHLPSCT	CDHLPSCT	DHLPSCT	KSOMSCT	TDES	AES
1	a01. exe	1,063	11985	13806	12980	14172	31047	15349
2	a02. exe	2,518	67093	79938	76321	89946	167604	58911
3	a03. exe	8,250	84521	84902	84120	86091	3171258	1193952
4	a04. exe	15,937	115239	108356	103895	99387	137421	90439
5	a05. exe	22,874	27905	26894	26498	25985	40605	42948
6	a06. exe	35,106	273973	272095	269874	257394	751034	996561
7	a07. exe	52,032	114783	111875	110845	108746	252246	227972
8	a08. exe	145,387	641097	637894	630955	622467	1619619	879622
9	a09. exe	248,273	641117	636949	629864	618647	1188392	1206461
10	a10. exe	478,321	825287	821674	801097	796454	1646895	1611814
11	a11. exe	738,275	1715636	1654098	1589549	1557482	1953381	1955305
12	a12. exe	1,594,276	2330865	2319485	228948	221837	3388013	3349821
13	a13. exe	2,273,670	4029864	4009856	3975635	3876748	5386323	5358508
14	a14. exe	2,985,306	3751834	3567849	3517843	3378487	4435189	4391280
15	a15. exe	3,412,639	1933896	1897485	1820946	1765849	312451	304503
16	a16. exe	3,872,984	2519086	2487650	2285773	2018478	2859239	2529935
17	a17. exe	4,038,387	5986	5521	5129	4786	8783	9015
18	a18. exe	5,284,796	536027	5343398	5276749	4987584	6874552	6590217
19	a19. exe	5,628,037	18837648	16654901	15749327	14894756	22431762	16734368
20	a20. exe	6,735,934	51852098	50748754	49786481	46658494	66742981	34387484
Average			4757447	4574169	4349141	4104189	6169940	4096723

Comparisons of Chi-Square value of *.txt* files

Serial no.	Source File name	Source file size (In bytes)	Chi-Square values					
			CTHLPSCT	CDHLPSCT	DHLPSCT	KSOMSCT	TDES	AES
1	a01.txt	1,504	386473	37485	37198	31938	58385	15267
2	a02.txt	7,921	12258309	785634	653812	587129	1500874	347663
3	a03.txt	17,036	86129865	4823409	4425909	3809645	7721661	1731310
4	a04.txt	44.624	46919023	7239064	5956321	4923806	4709724	4753971
5	a05.txt	68,823	389278954	27187564	22129876	18569064	29704639	15663977
6	a06.txt	161,935	457129845	92790569	76438907	68865906	76621083	64043270
7	a07.txt	328,017	3561209673	423376129	353890745	328096745	388539921	325837900
8	a08.txt	587,290	15990965340	1738797676	1549867335	1364538932	1258362670	1082315460
9	a09.txt	1,049,763	55312986743	4849846875	3485690453	2965423187	3264221211	2585100024
10	a10.txt	1,418,025	51287369561	8195645098	7549087564	8237908765	5896971610	5524089746
11	a11.txt	1,681,329	165532816732	14145390986	12198087654	7941894390	9087072783	8355902146
12	a12.txt	2,059,318	181632907453	18967452398	18767453098	12967095437	11627270156	11387797334
13	a13.txt	2,618,492	305095623874	31912985534	29543098734	25839096738	24260650965	21978420834
14	a14.txt	3,154,937	487589453287	40756340987	34529187230	28634908759	32021906499	29332709650
15	a15.txt	4,073,829	619432716093	75327909654	49756230987	53867340987	47346524666	44660520923
16	a16.txt	4,936,521	926431276356	81470983456	56867215490	52412907645	57698683717	49783638147
17	a17.txt	5,125,847	1255439061805	141907654387	116340982395	56164389079	72922461490	64846889153
18	a18.txt	5,593,219	1496745328775	125984609879	96490863457	88954120953	81707468147	77543081318
19	a19.txt	5,898,302	1695790756468	128653909645	175563409174	126390854880	101228345379	89456481325
20	a20.txt	6,702,831	1880940484091	148409329116	134895489087	148280978453	122325249286	109577386113
Average			457088752936	41143854777	36900009771	30722317122	28557702243	25826336277

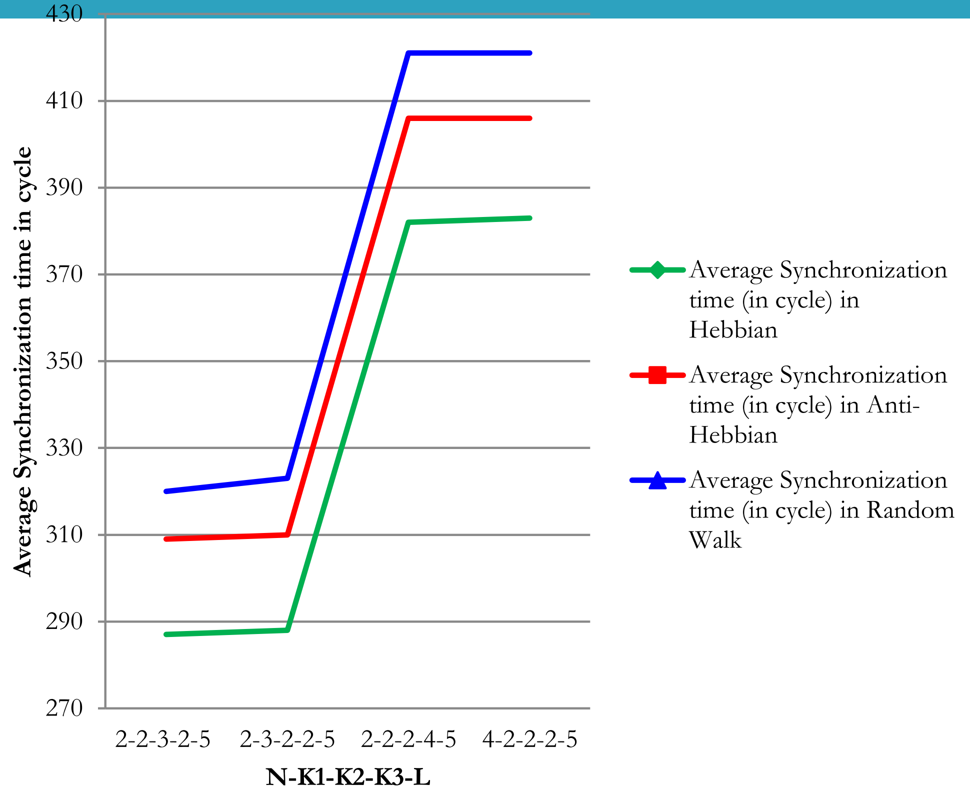
Comparisons of Chi-Square value of *.doc* files

Serial no.	Source File name	Source file size (In bytes)	Chi-Square values					
			CTHLPSCT	CDHLPSCT	DHLPSCT	KSOMSCT	TDES	AES
1	a01. doc	21,052	14893	6210846	6109459	6584038	18918008	14182910
2	a02. doc	33,897	43903	1298305	2540299	2890458	9503676	4431277
3	a03. doc	45,738	115845	1467406	1904827	1832039	9361015	2145383
4	a04. doc	75,093	237094	1760847	1799038	1719053	2848468	1347091
5	a05. doc	106,872	507195	1909562	1870653	1794092	3933039	1898438
6	a06. doc	327.054	297127	523096	590956	580984	537285	373599
7	a07. doc	582,831	946538	920982	886429	863092	1349490	947148
8	a08. doc	729,916	5919048	5829064	5639042	5509832	5474962	4532789
9	a09. doc	1,170,251	2560942	2426703	2190563	2090482	4598604	3097778
10	a10. doc	1,749,272	26192730	25795683	25137093	24709385	41385774	27850217
11	a11. doc	2,045,805	21295672	19504987	19134097	18630942	23692555	11574426
12	a12. doc	2,372,014	14909583	14290539	14178095	13790434	18656807	11848004
13	a13. doc	2,869,275	8569396	6109565	6023896	5729084	17460853	8762683
14	a14. doc	3,161,353	10263935	10267429	10198431	9987353	9904389	6784251
15	a15. doc	3,570,295	9245042	8534074	8129054	7940973	11123554	6844351
16	a16. doc	3,834,427	8598424	7556031	7230986	6990386	7725687	5230567
17	a17. doc	4,011,986	8437261	7031864	6939093	6759037	9846653	6437662
18	a18. doc	4,562,385	7838924	6539063	6287235	6073904	7376693	5591776
19	a19. doc	4,839,102	6210498	4804169	4750934	4580856	8592223	5374094
20	a20.doc	5,472,298	7329048	6041093	5630939	5209857	8145414	6012872
Average			6976655	6941065	6858556	6713314	11021752	6763362

RESULTS AND ANALYSIS

Average Synchronization Time (in cycle) for Generating 128 bit Session Key using fixed Weight range (L=5) with variable Neurons in CTHLPSCT

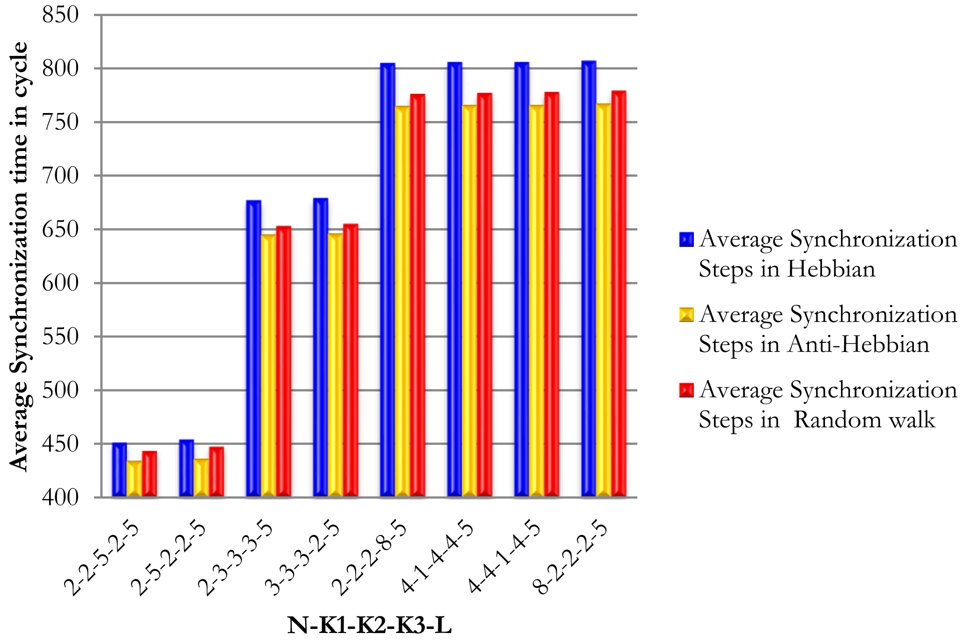
CTHLP Size	N-K1-K2-K3-L	Average Synchronization time in cycle		
		Hebbian	Anti-Hebbian	Random Walk
24	2-2-3-2-5	287,81	309,23	320,52
24	2-3-2-2-5	288,77	310,35	323,15
32	2-2-2-4-5	382,93	406,87	421,36
32	4-2-2-2-5	383,18	406,91	421,89



RESULTS AND ANALYSIS

Average Synchronization Time (in cycle) for Generating 192 bit Session Key using fixed Weight range (L=5) with variable Neurons in CTHLPSCT

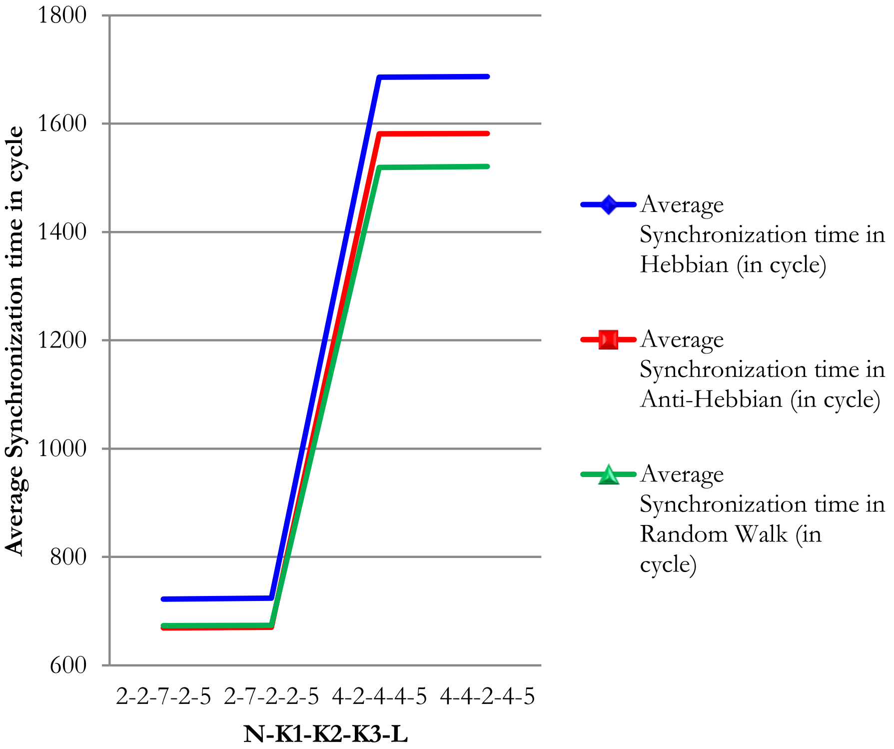
CTHLP Size	N-K1-K2-K3-L	Average Synchronization time in cycle		
		Hebbian	Anti-Hebbian	Random Walk
40	2-2-5-2-5	451,17	434,83	443,02
40	2-5-2-2-5	454,37	436,11	447,19
54	2-3-3-3-5	677,76	645,89	653,92
54	3-3-3-2-5	679,23	646,05	655,11
64	2-2-2-8-5	805.71	765,53	776,84
64	4-1-4-4-5	806.16	766,10	777,61
64	4-4-1-4-5	806.98	766,87	778,41
64	8-2-2-2-5	807.24	767,63	779,12



RESULTS AND ANALYSIS

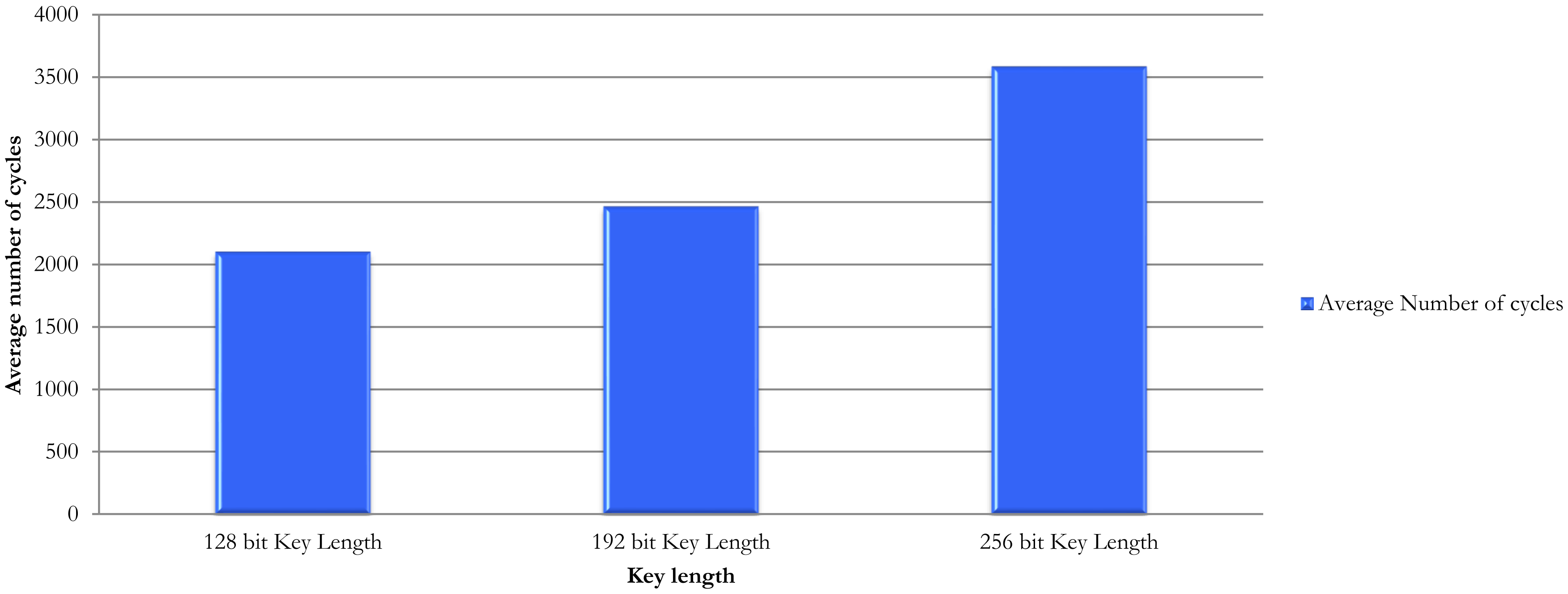
Average Synchronization Time (in cycle) for Generating 256 bit Session Key using fixed Weight range (L=5) with variable Neurons in CTHLPSCT

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



Results and analysis

Average Synchronization Time (in cycle) for Generating variable session key in CTHLPSCT



Problems in CTHLPSCT

- Previous technique does not offer synchronization of group of parties.

- The CTHLPSCT does not implements the key swap over technique in scales logarithmically with the number of parties participating in the key swap over protocol.

- So, if there are n parties then total number of synchronizations needed is $O(n^2)$.

Chaos based Grouped Triple Hidden Layer Perceptron Synchronized Cryptographic Technique (CGTHLPSCT)

- Performs key swap over by synchronization among cluster of CTHLP.
- 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.
- Simple and secure Particle Swarm Intelligence (PSI) guided enciphering technique has been proposed.

Chaos based Grouped Triple Hidden Layer Perceptron Synchronized Cryptographic Technique (CGTHLPSCT)

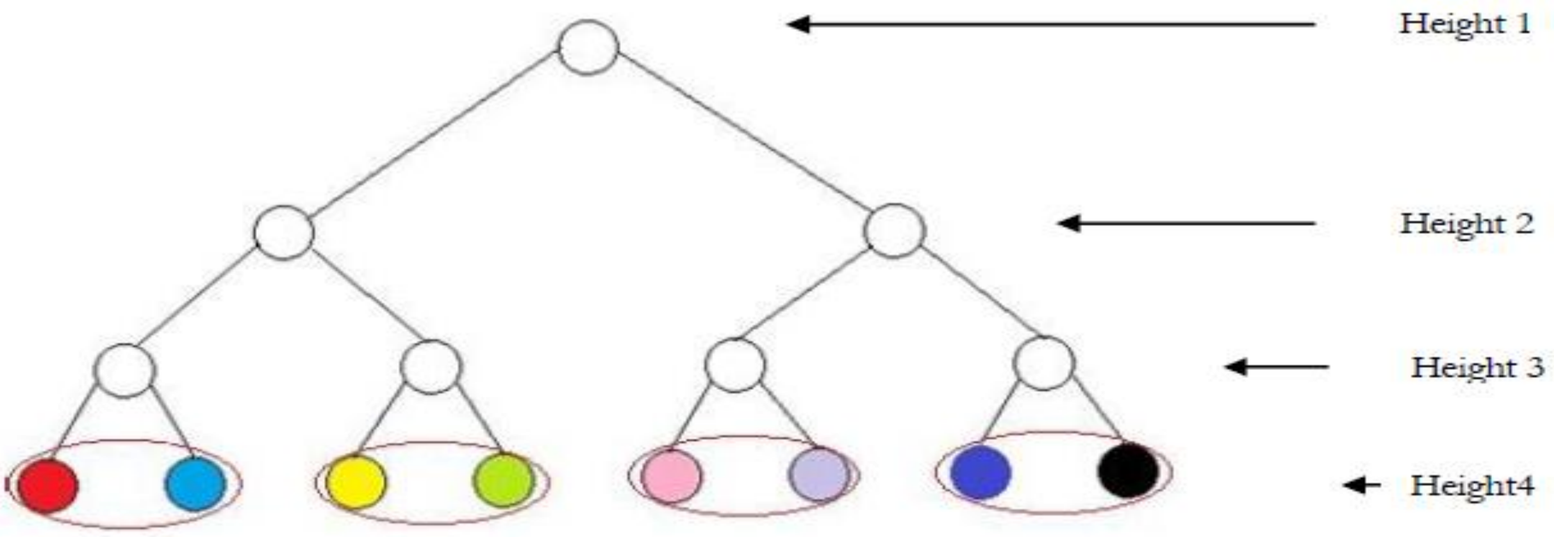


Figure 1: Initial state of group synchronization

Chaos based Grouped Triple Hidden Layer Perceptron Synchronized Cryptographic Technique (CGTHLPSCT)

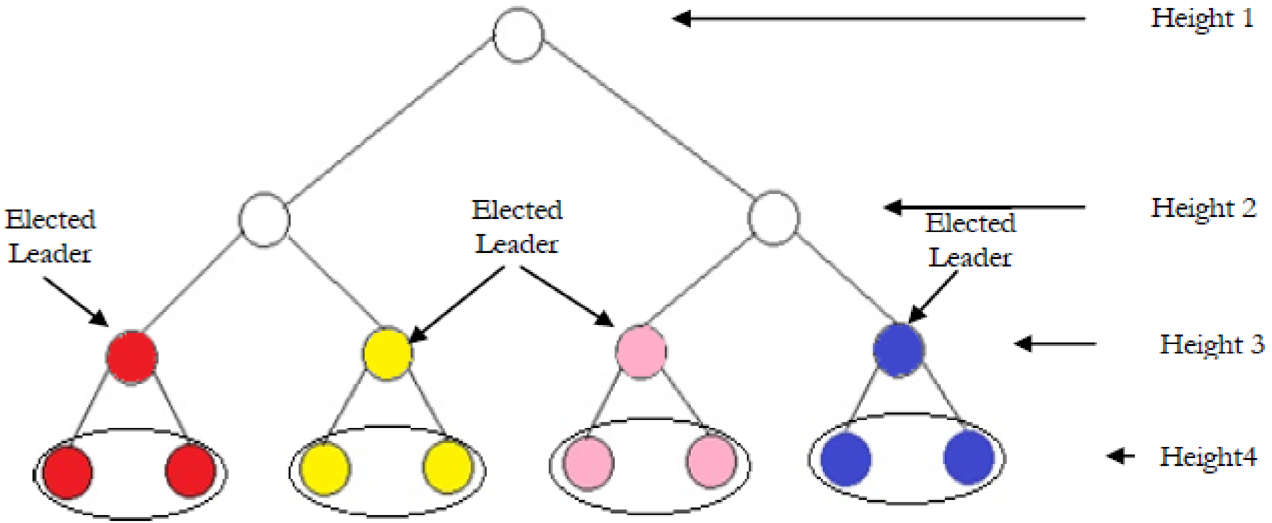


Figure 2: First round of group synchronization

Chaos based Grouped Triple Hidden Layer Perceptron Synchronized Cryptographic Technique (CGTHLPSCT)

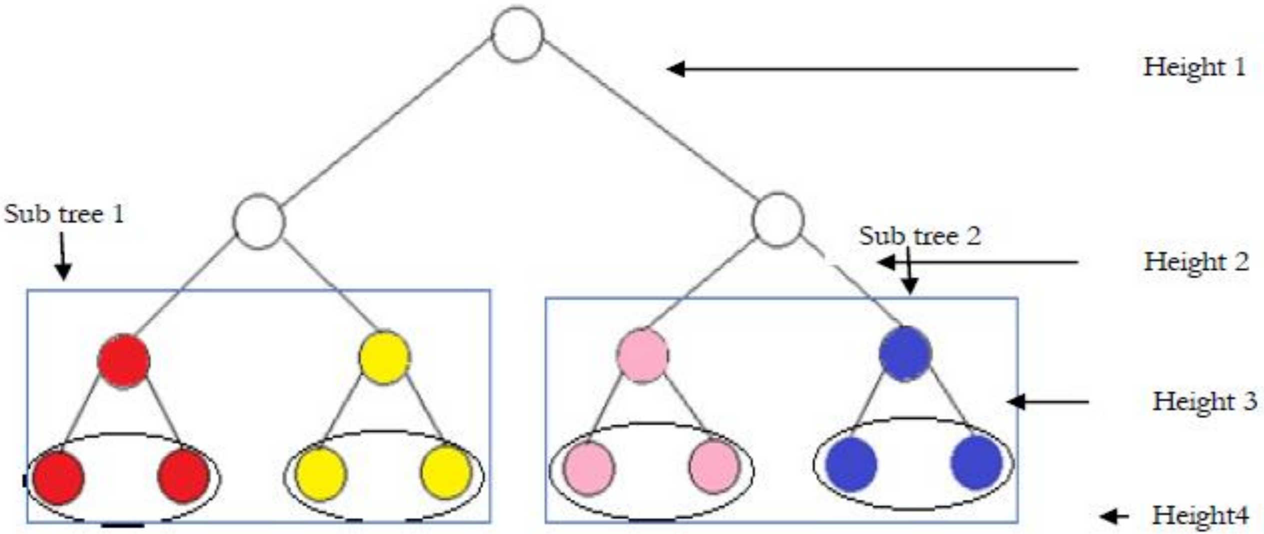


Figure 3: Second round of group synchronization

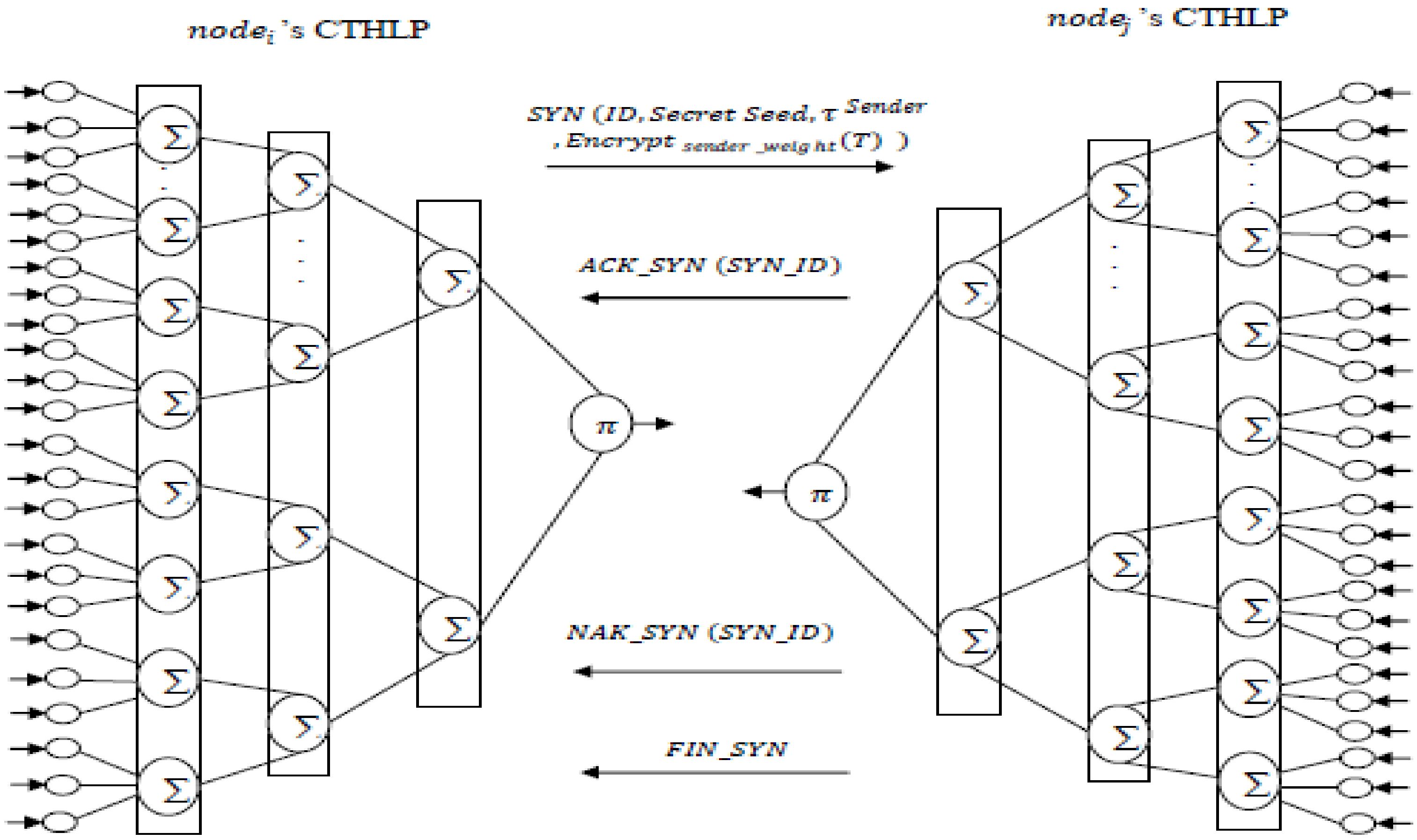


Figure : Exchange of control frames between $node_i$ and $node_j$ during CGTHLP synchronization

Example of PSI based Keystream Generation

Consider the text to be encrypted is "**softcomputing**". The minimum probability value is assumed to be **0.75.**

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	0.60	tof-3	6	gmtof	0.75
rbzlsy	1	0.16	pcu-3	4	pcu	0.44	ma-1	5	pcuma	0.45
csegdx	3	0.50	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**				**0.60**				**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

So binary representation of ASCII value of the PSI based keystream is
01101000/01100011/01110110/01100111/01101101/01110100/01101111/01100110/
01101110/01101001/01111100/01101101/01110011

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

$S_1 = 0111001101101111$ (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)

Perform cycle formation techniques on $C = c_0^j\, c_1^j\, c_2^j\, c_3^j c_4^j\, \ldots\, c_{n-1}^j$ *of block of size n. In the following cases* $\oplus$ *is used to represents the Exclusive-OR operation. Perform the operations given in equations for generating the first intermediate block*

$I_1 = c_0^{j+1} c_1^{j+1}\, c_2^{j+1}\, c_3^{j+1}\, c_4^{j+1}\, \ldots\, c_{n-1}^{j+1}$ *from C in the following way:*

$$c_{n-1}^{j+1} = c_{n-1}^{j}$$

$$c_{n-2}^{j+1} = c_{n-2}^{j} \oplus c_{n-1}^{j+1}$$

$$c_{1}^{j+1} = c_{1}^{j} \oplus c_{2}^{j+1}$$

$$c_{0}^{j+1} = c_{0}^{j}$$

This process continues for a finite number of iterations, which depends on the value of n, the source block C is regenerated.

The formation of cycles for segments (0111001101101111). An arbitrary intermediate segment (0101001111100011) after iteration-10 considered as an encrypted segment for the segment S_1.

0111001101101111→0101000100100101^{1}→0011000011100011^{2}→0110111110100001^{3}→
0101101010011111^{4}→0011011001110101^{5}→0110110111010011^{6}→0010010010110001^{7}→
0001110001101111^{8}→0111010000100101^{9}→**0101001111100011^{10}**→0100111010100001^{11}→010
0010110011111^{12}→0100001101110101^{13}→0011111011010011^{14}→
0001010110110001^{15}→0111001101101111^{16}

The formation of cycles for segments S_2 (0110011001110100 An arbitrary intermediate segment (0001001001110100) after iteration-8 considered as an encrypted segment for the segment S_2.

0110011001110100→0010001000101100^{1}→0110000111100100^{2}→0101111101011100^{3}→
0011010100110100^{4}→0110110011101100^{5}→0101101110100100^{6}→0011011010011100^{7}→
0001001001110100^{8}→0000111000101100^{9}→0000010111100100^{10}→0000001101011100^{11}→00
00001001110100^{12}→0000000011101100^{13}→0111111110100100^{14}→
0010101010011100^{15}→0110011001110100^{16}

The formation of cycles for segments S_3 (0110001101101111). 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}→0000000111100011^{10}→0111111010100001^{11}→
0101010110011111^{12}→0011001101110101^{13}→0110111011010011^{14}→
0010010110110001^{15}→0110001101101111^{16}

The formation of cycles for segments S_4 (0110110101110000). 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}→0010011100100000^{7}→
0001110101110000^{8}→0111010011010000^{9}→0101001110110000^{10}→0100111010010000^{11}→001
1101001110000^{12}→0110100111010000^{13}→0101100010110000^{14}→
0011011110010000^{15}→0110110101110000^{16}

The formation of cycles for segments S_5 (0111010101110100). 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). 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}→00110111^{7}→01101001^{8}

The formation of cycles for segments S_7 (011011100110011 1). An arbitrary intermediate segment (001011001111011) after iteration-6 considered as an encrypted segment for the segment S_7.

011011100110011 1→0010010111011101^{1}→0110001101001011^{2}→0010000100111001^{3}→0001111100010111^{4}→0111010100001101^{5}→**0010110011111011^{6}**→0001101110101001^{7}→0000100101100111^{8}→0111100011011101^{9}→0010100001001011^{10}→0001100000111001^{11}→0 0001000000010111^{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.

01010011/11100011/00010010/01110100/01010101/10011111/01001010/01110000/0110 0110/01011100/00011101/00101100/11111011

Example of Encryption

For example CGTHLP based following group session key is generated
10010101/01010010/11111000/01010101/01011101/01111110/00110101/01001111/1000 1010/00011100/10001010/10010010/11011100

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.
10101110/11010010/10011100/01000110/01100101/10010101/00011010/00101100/0001 0000/01101010/11000010/10000011/00010000

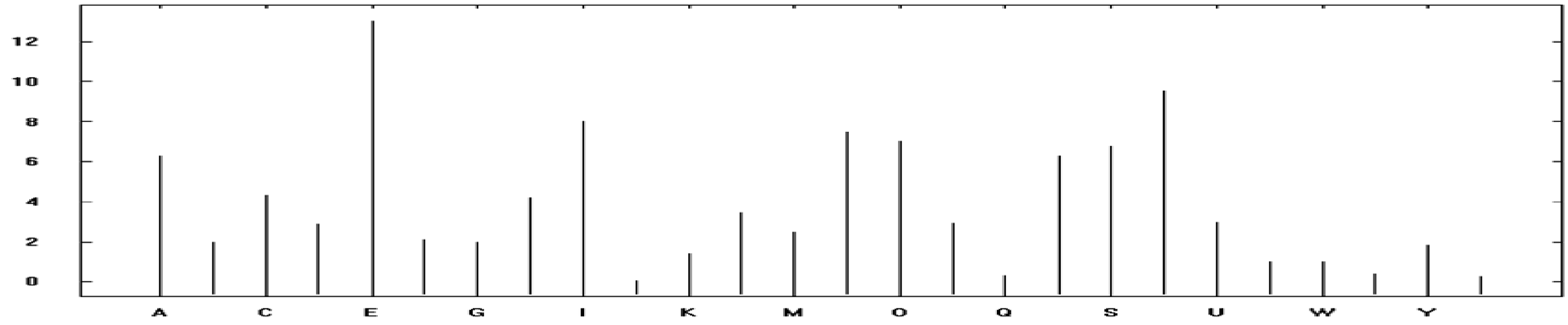


Figure: Graphical representation of frequency distribution spectrum of characters for the input .*txt* source stream

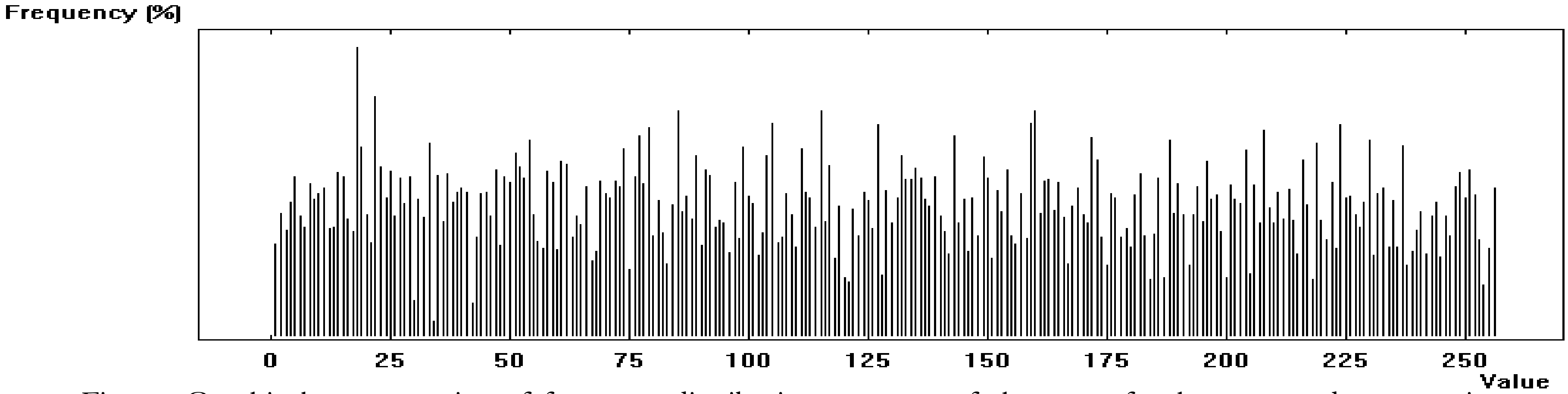


Figure : Graphical representation of frequency distribution spectrum of characters for the encrypted stream using CGTHLPSCT for .*txt* file

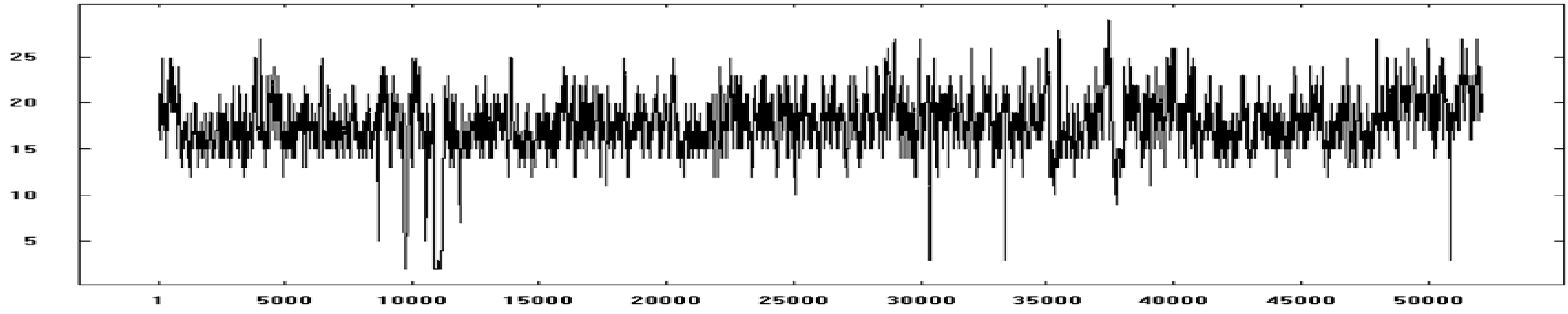


Figure : Floating frequency of the input *.txt* source stream

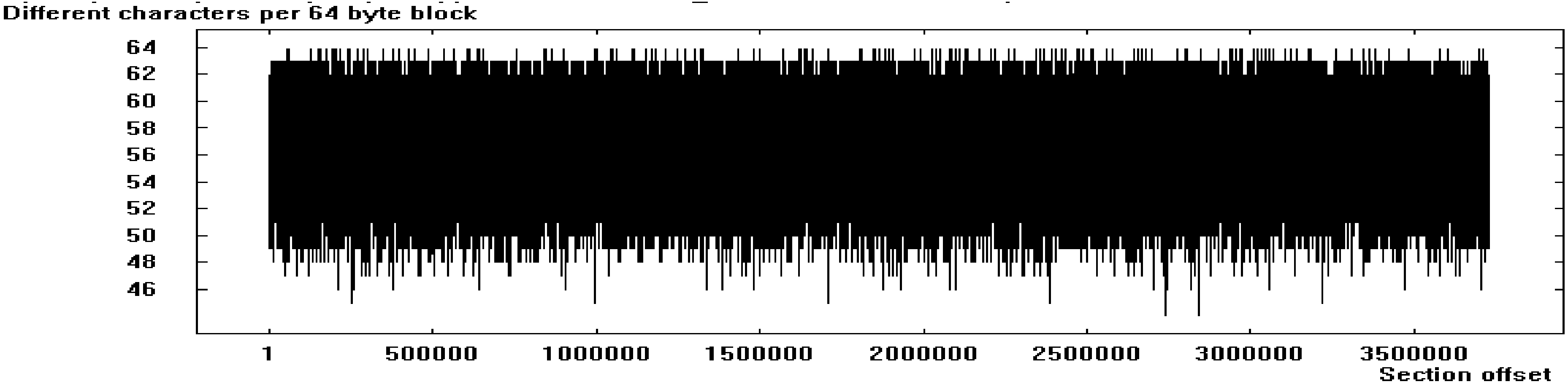


Figure : Floating frequency of the encrypted stream using CGTHLPSCT for *.txt* file

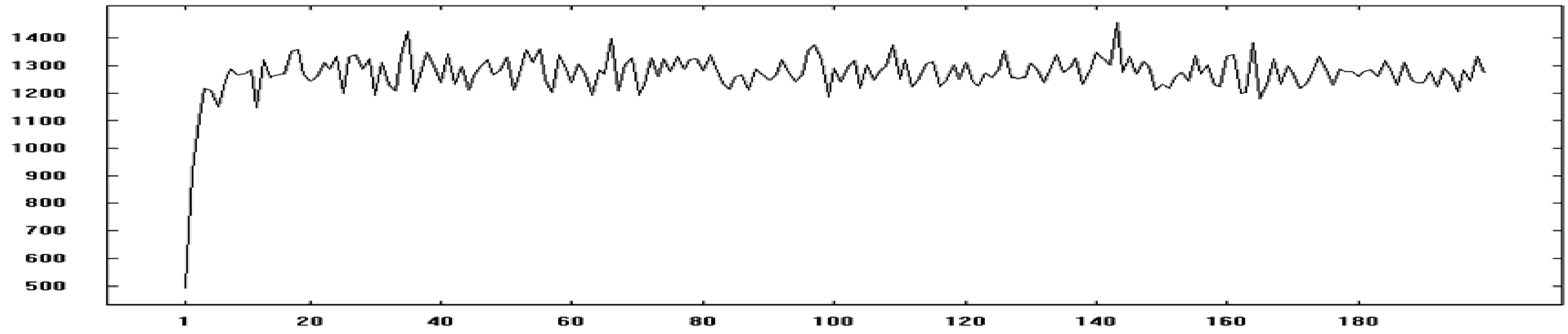


Figure : Autocorrelation of the input *.txt* source stream

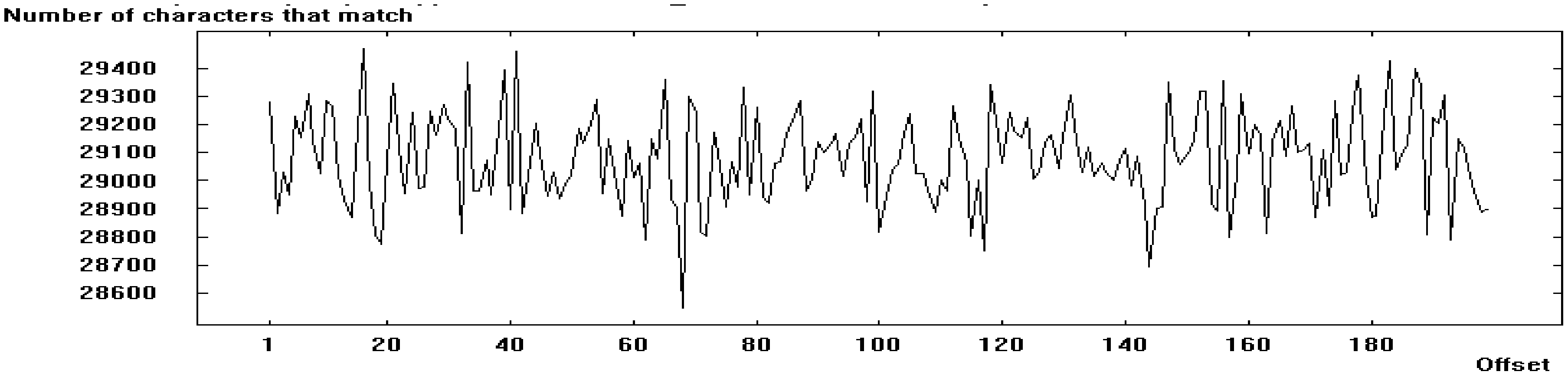


Figure : Autocorrelation of the encrypted stream using CGTHLPSCT for *.txt* file

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

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

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

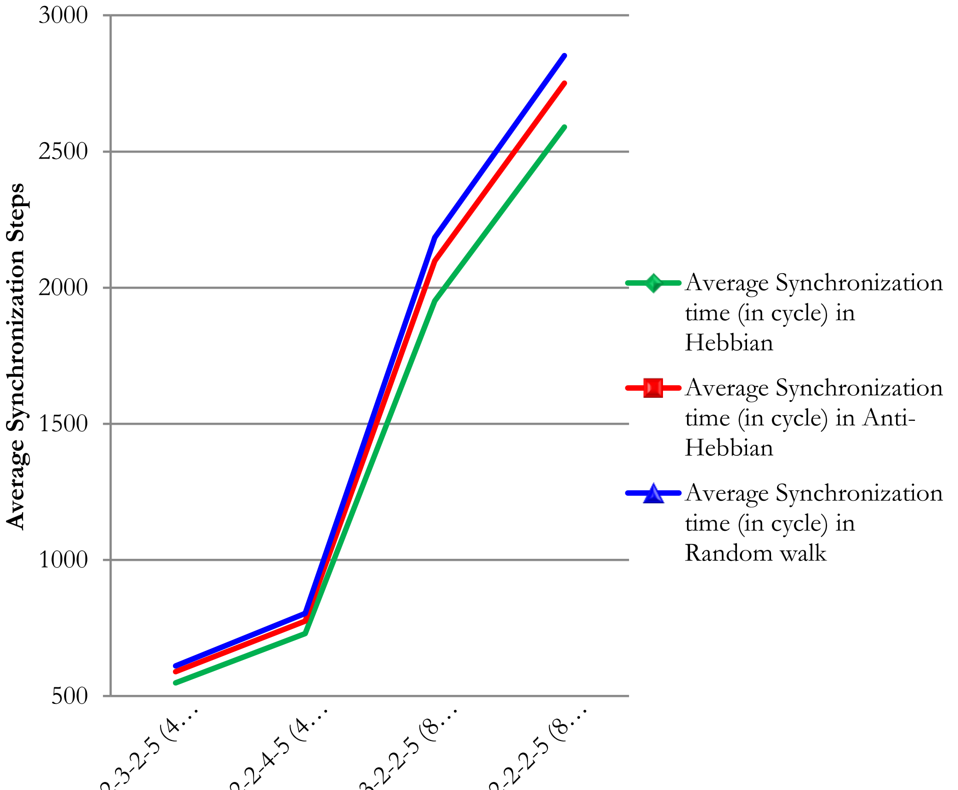
Comparisons of Chi-Square value of *.doc* files

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

RESULTS AND ANALYSIS

Average Synchronization Time (in cycle) for Generating 128 bit Session Key using fixed Weight range (L=5) with variable Neurons in CGTHLPSCT

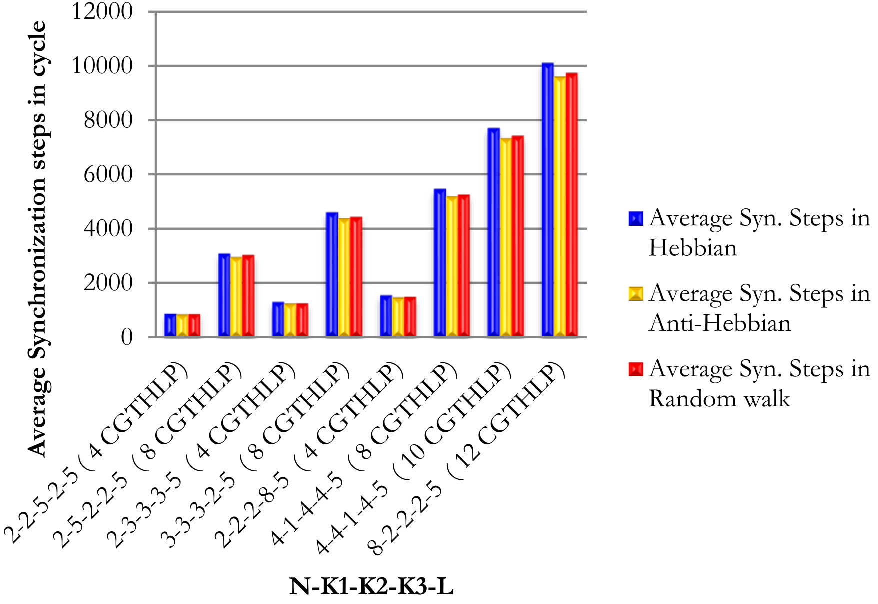
CGT HLP Size	N-K1-K2-K3-L	No. of CGTHLP Participated at Group Session Key Generation	Average Synchronization Steps		
			Hebbian	Anti-Hebbian	Random Walk
24	2-2-3-2-5	4	549,28	590,16	611,70
32	2-2-2-4-5	4	730,81	776,50	804,15
24	2-3-2-2-5	8	1952,31	2098,20	2184,74
32	4-2-2-2-5	8	2590,59	2751,03	2852,30



Results and analysis

Average Synchronization Time (in cycle) for Generating 192 bit Session Key using fixed Weight range (L=5) with variable Neurons in CGTHLPSCT

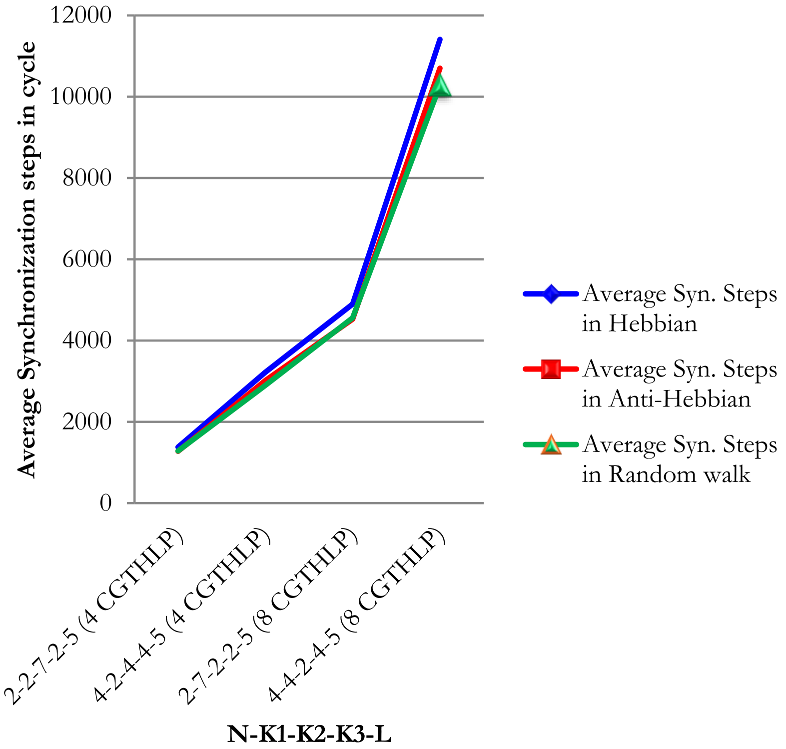
CTHLP Size	N-K1-K2-K3-L	No. of CTHLP Participated at Group Session Key Generation	Average Synchronization steps in cycle		
			Hebbian	Anti-Hebbian	Random Walk
40	2-2-5-2-5	4	861,05	829,86	845,49
40	2-5-2-2-5	8	3071,89	2948,44	3023,35
54	2-3-3-3-5	4	1293,49	1232,67	1247,99
54	3-3-3-2-5	8	4592,12	4367,80	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



Results and analysis

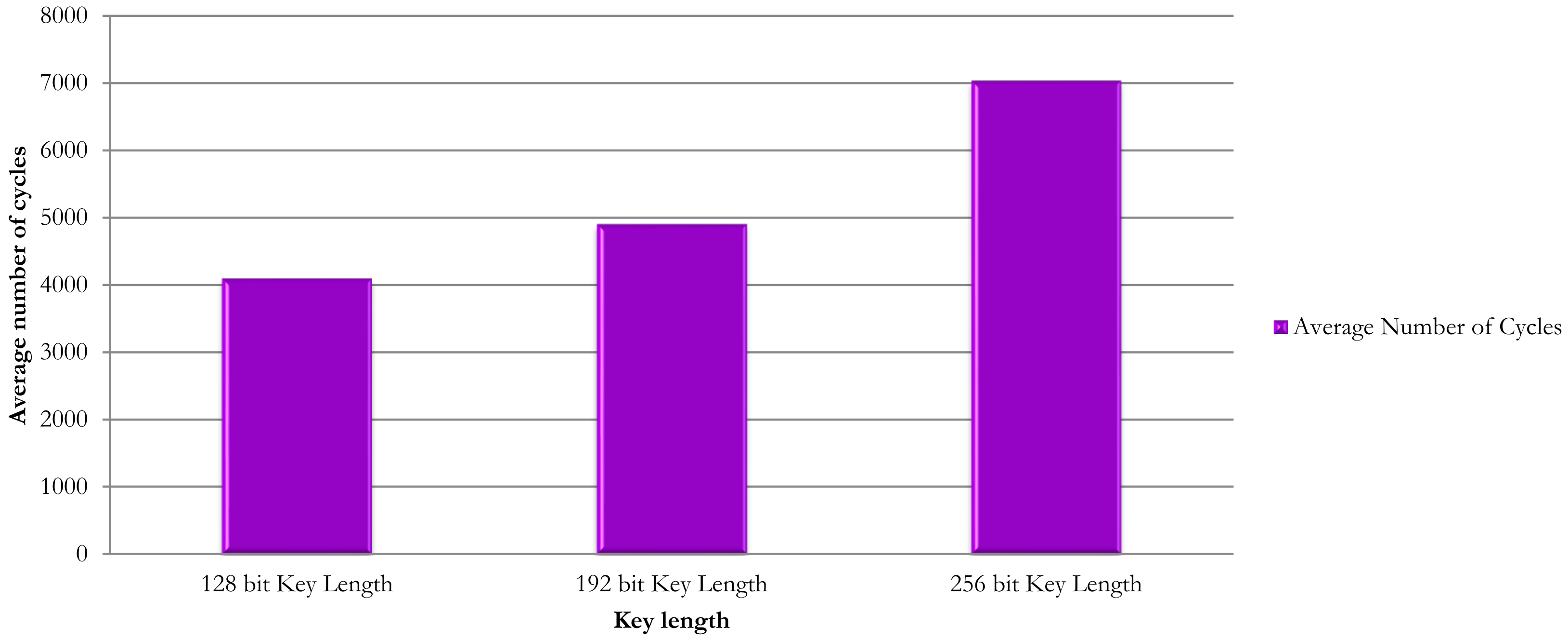
Average Synchronization Time (in cycle) for Generating 256 bit Session Key using fixed Weight range (L=5) with variable Neurons in CGTHLPSCT

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



RESULTS AND ANALYSIS

Average Synchronization Time (in cycle) for Generating variable session key in CGTHLPSCT



Strength of Neural Cryptography

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

Strength of Neural Cryptography

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	$256^{64\times1} = 2^{512}$	2^{511}μs= 2.13×2^{140} years	2.13×2^{134} years
2	$256^{64\times2} = 2^{1024}$	2^{1023}μs= 2.85×2^{294} years	2.85×2^{288} years
3	$256^{64\times3} = 2^{1536}$	2^{1535}μs= 3.82×2^{448} years	3.82×2^{442} years
4	$256^{64\times4} = 2^{2048}$	2^{2047}μs= 5.12×2^{602} years	5.12×2^{596} years
5	$256^{64\times5} = 2^{2560}$	2^{2559}μs= 6.87×2^{756} years	6.87×2^{750} years
6	$256^{64\times6} = 2^{3072}$	2^{3071}μs= 9.21×2^{910} years	9.21×2^{904} years
7	$256^{64\times7} = 2^{3584}$	2^{3583}μs= 1.23×2^{1065} years	1.23×2^{1059} years
8	$256^{64\times8} = 2^{4096}$	2^{4095}μs= 1.66×2^{1219} years	1.66×2^{1213} years

NIST Statistical Test

Frequency (Monobits) Test

The focus of the test is the proportion of zeroes and ones for the entire sequence. 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.

Frequency (Monobits) Test

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing
TPM	0.972766	0.973333	Success
KSOMSCT		0.976329	Success
DHLPSCT		0.979437	Success
CDHLPSCT		0.983333	Success
CTHLPSCT		0.984871	Success
CGTHLPSCT		0.986667	Success

Test For Frequency Within A Block

The focus of the test is 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 M/2.

Test For Frequency Within A Block

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing
TPM	0.972766	0.963333	Unsuccess
KSOMSCT		0.972818	Success
DHLPSCT		0.977942	Success
CDHLPSCT		0.980000	Success
CTHLPSCT		0.984792	Success
CGTHLPSCT		0.990000	Success

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.

Runs Test

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing
TPM	0.972766	0.974275	Success
KSOMSCT		0.975746	Success
DHLPSCT		0.977263	Success
CDHLPSCT		0.986997	Success
CTHLPSCT		0.990000	Success
CGTHLPSCT		0.993333	Success

Test For The Longest Run Of Ones In A Block

The focus of the test is 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.

Test For The Longest Run Of Ones In A Block

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing
TPM	0.972766	0.986667	Success
KSOMSCT		0.97051	Unsuccess
DHLPSCT		0.988026	Success
CDHLPSCT		0.990000	Success
CTHLPSCT		0.993174	Success
CGTHLPSCT		0.996667	Success

Random 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.

Random Binary Matrix Rank Test

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing
TPM	0.972766	0.970173	Unsuccess
KSOMSCT		0.990000	Success
DHLPSCT		0.992619	Success
CDHLPSCT		0.993333	Success
CTHLPSCT		0.995493	Success
CGTHLPSCT		0.996667	Success

Discrete Fourier Transform (Spectral) 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.

Discrete Fourier Transform (Spectral) Test

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing
TPM	0.972766	0.951467	Unsuccess
KSOMSCT		0.968329	Unsuccess
DHLPSCT		1.000000	Success
CDHLPSCT		1.000000	Success
CTHLPSCT		1.000000	Success
CGTHLPSCT		1.000000	Success

Non-Overlapping (Aperiodic) Template Matching Test

The focus of this test is the number of occurrences of pre-defined target substrings. 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.

Non-Overlapping (Aperiodic) Template Matching Test

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing
TPM	0.972766	0.987164	Success
KSOMSCT		0.988891	Success
DHLPSCT		0.992275	Success
CDHLPSCT		0.994872	Success
CTHLPSCT		0.998941	Success
CGTHLPSCT		1.000000	Success

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.

Overlapping (Periodic) Template Matching Test

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing
TPM	0.972766	0.970173	Unsuccess
KSOMSCT		0.980201	Success
DHLPSCT		0.982107	Success
CDHLPSCT		0.983932	Success
CTHLPSCT		0.985028	Success
CGTHLPSCT		0.985739	Success

Maurer's Universal Statistical Test

The focus of this test is the number of bits between matching patterns. 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.

Maurer's Universal Statistical Test

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing
TPM	0.972766	1.000000	Success
KSOMSCT		1.000000	Success
DHLPSCT		1.000000	Success
CDHLPSCT		1.000000	Success
CTHLPSCT		1.000000	Success
CGTHLPSCT		1.000000	Success

Linear Complexity Test

The focus of this test is 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.

Linear Complexity Test

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing
TPM	0.972766	0.973333	Success
KSOMSCT		0.975318	Success
DHLPSCT		0.994763	Success
CDHLPSCT		1.000000	Success
CTHLPSCT		1.000000	Success
CGTHLPSCT		1.000000	Success

Serial Test

The focus of this test is 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.

Serial Test

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing
TPM	0.977814	0.971333	Unsuccess
KSOMSCT		0.973903	Unsuccess
DHLPSCT		0.979874	Success
CDHLPSCT		0.980850	Success
CTHLPSCT		0.981476	Success
CGTHLPSCT		0.991667	Success

Approximate Entropy Test

The focus of this test is 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.

Approximate Entropy Test

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing
TPM	0.972766	0.983333	Success
KSOMSCT		0.985830	Success
DHLPSCT		0.987328	Success
CDHLPSCT		0.991739	Success
CTHLPSCT		0.9968372	Success
CGTHLPSCT		0.998174	Success

Cumulative Sum (Cusum) 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.

Cumulative Sum (Cusum) Test

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing
TPM	0.977814	0.953762	Unsuccess
KSOMSCT		0.971289	Unsuccess
DHLPSCT		0.980000	Success
CDHLPSCT		0.987291	Success
CTHLPSCT		0.995218	Success
CGTHLPSCT		0.998543	Success

Random Excursions Test

The focus of this test is 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 Test

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing
TPM		0.935000	Unsuccess
KSOMSCT		0.942500	Unsuccess
DHLPSCT	0.983907	0.987359	Success
CDHLPSCT		0.993964	Success
CTHLPSCT		0.996667	Success
CGTHLPSCT		0.997274	Success

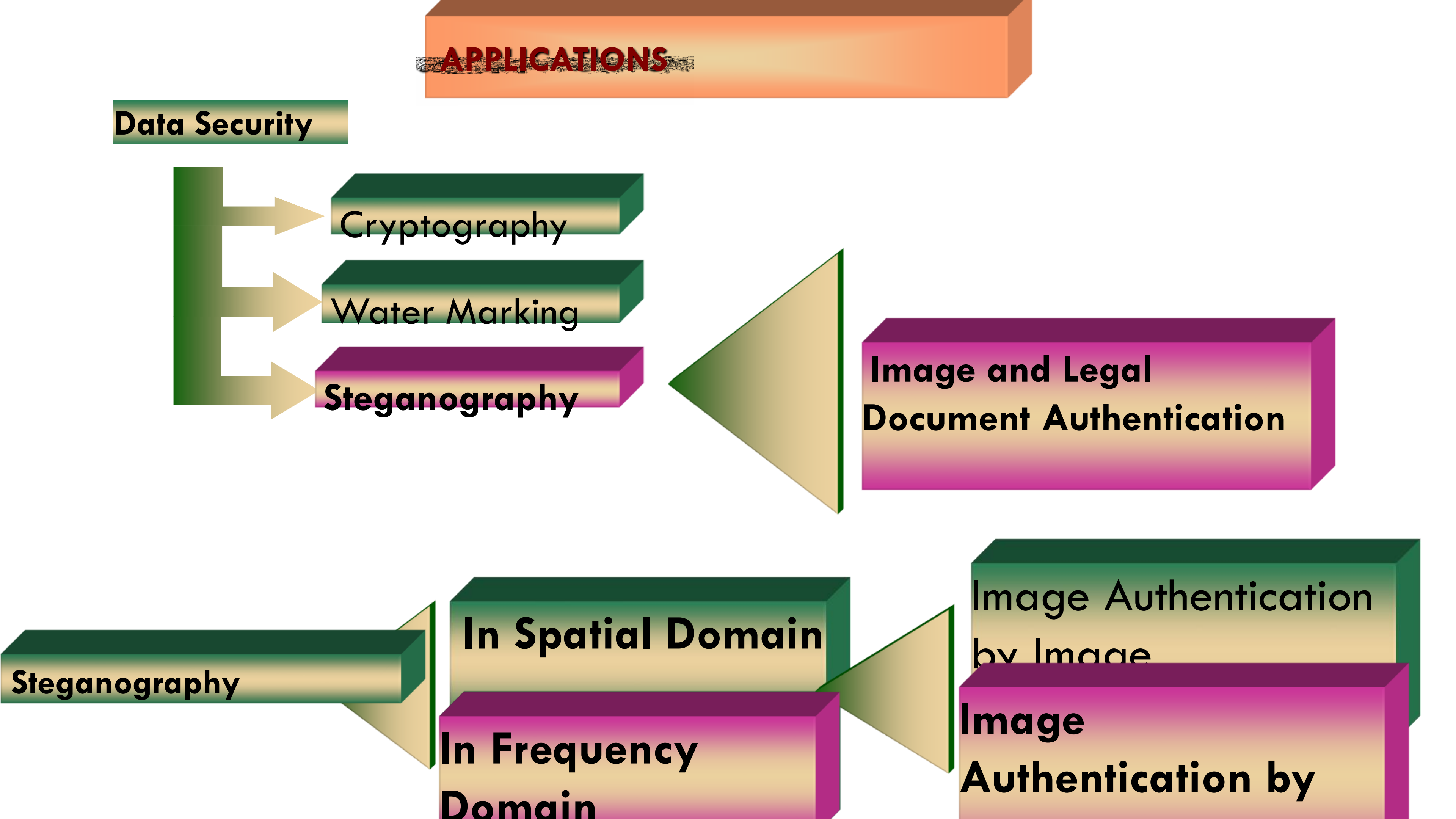
Random Excursions Variant Test

The focus of this test is 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.

Random Excursions Variant Test

Technique	Expected Proportion	Observed Proportion	Status for Proportion of passing
TPM	0.985938	0.972593	Unsuccess
KSOMSCT		0.972963	Unsuccess
DHLPSCT		0.986893	Success
CDHLPSCT		0.988286	Success
CTHLPSCT		0.989103	Success
CGTHLPSCT		0.989928	Success

Applications


APPLICATIONS
Data Security
Cryptography
Water Marking
Steganography
Image and Legal
Document Authentication
Steganography
In Spatial Domain
In Frequency
Domain
Image Authentication
by Image
Image
Authentication by

APPLICATIONS STEGANOGRAPHY

DOCUMENT AUTHENTICATION

Technique to Authenticate

DOCUMENT AUTHENTICATION

We are Indian. We are proud for our country. We always like to look ahead with positive attitude and giving maximum effort to growth our country. We are so much strong in science and Technology.

We are Indian. We are proud for our country. We always like to look ahead **with negative attitude** and giving **minimum effort to** growth our **country. We are so much weak** in science and Technology.

DOCUMENT AUTHENTICATION

Extract MD5

Compare

Generate MD5

We are Indian. We are proud for our country. We always like to look ahead with negative attitude and giving minimum effort to growth our country. We are so much weak in science and Technology.

Conclusions

- **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 is the number of cascading stages. It indicates that the **key space of the session key is very large.**

- **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 . So, model is highly secured from Brute-force attack.

- **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.

Conclusions

- **Variable size key –** ***Variable size session key with high key space*** can be used in different session.

- **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.
- **Non-homogeneity –**All the measures indicate that the ***degree of non-homogeneity of the encrypted stream with respect to the source stream is good***.

- **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.

Conclusions

- **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.

- **Correlation between source and encrypted stream –** Proposed technique may effectively resist data correlation statistical attack.

- **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.

References

1. NIST statistical test. Retrieved Mar 01 2017, from http://csrc.nist.gov/groups/ST/toolkit/rng/ stats_tests.html.
2. Feistel, H. (1973). Cryptography and Computer Privacy. Scientific American, May 1973, Vol. 228 No. 5, pp.15-23.
3. Rivest, R. L. (1990). Cryptology. In A. Jan Van Leeuwen (Ed.), Handbook of Theoretical Computer Science, chapter 13, pp.717-755, Elsevier / MIT Press.
4. Stinson, D. R. (1995). Cryptography, Theory and Practice, CRC Press.
5. Bellare, Mihir, Rogaway, & Phillip (2005). Introduction. Introduction to Modern Cryptography, pp.10.
6. Menezes, A.J., Vanstone, S.A., & Van Oorschot, P.C. (1996). Handbook of Applied Cryptography, In: Applied Cryptography, CRC Press, Boca Raton.
7. Cryptography. Retrieved August 04 2012, from http://en.wikipedia.org/wiki/Cryptography
8. Kahn, D. (1967). The Codebreakers, ISBN 0-684-83130-9.
9. Encryption, Retrieved August 05 2012, from http://en.wikipedia.org/wiki/Encryption
10. Encryption Basics, EFF Surveillance Self-Defense Project. (n.d.). Retrieved Nov 06 2013, from https://ssd.eff.org/tech/encryption.
11. Goldreich, Oded. (2004). Foundations of Cryptography: Volume 2, Basic Applications. Vol. 2. Cambridge university press.
12. Kahate, A. (2010). Cryptography and Network Security, 2nd edition, Tata McGraw Hill.
13. Cipher, Retrieved August 05 2012, from http://en.wikipedia.org/wiki/Cipher
14. Schneier, B. (1995). Applied Cryptography: Protocols, Algorithms, and Source Code in C. 2nd edition. Wiley, New York.
15. Stallings, W. (2003). Cryptography and Network Security: Principles and Practices, 3rd edition, Pearson Education.
16. 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).
17. Cryptography Key, Retrieved August 06 2012, from http://en.wikipedia.org/wiki/Key_ (cryptography)
18. Diffie, W., & Hellman, M. (1976). Multi-user cryptographic techniques. In Proceedings of the AFIPS Proceedings 45, June 8 1976, pp.109-112.
19. Kahn, D. (1979). Cryptology Goes Public, 58 Foreign Affairs 141, 151 (fall 1979), pp.153.
20. Diffie, W., & Hellman, M. (1976). New directions in cryptography, IEEE Trans. Inform. Theory, 22(6), pp.644-654.

Thank You