Public Key Cryptography using Neural Networks and Genetic Algorithms

Smita Jhajharia
Department of Computer Engineering
Delhi Technological University
Delhi, India
smita00786@gmail.com

Swati Mishra, Siddharth Bali
Department of Computer Engineering
Delhi Technological University
Delhi, India
m.swatimishra13@gmail.com, siddharthbali92@gmail.com

***Abstract*—By making use of Artificial Intelligence (AI), Human Intelligence can be simulated by a machine, Neural Networks is one such sub field of AI. Artificial Neural Networks (ANN) consists of neurons and weights assigned to inter neuron connections helps in storing the acquired knowledge. This paper makes use of Hebbian learning rule to train the ANN of both sender and receiver machines. In the field of Public Key Cryptography (PKC), Pseudo Random Number Generator (PRNG) are widely used to generate unique keys and random numbers used in ANN which are found to possess many types of possible attacks. It is essential for a key to possess randomness for key strength and security. This paper proposes key generation for PKC by application of ANN using Genetic Algorithm (GA). It was noticed that use of ANN along with GA has not as yet been explored. GA approach is often applied for obtaining optimization and solutions in search problems. GA correlates to the nature to a large extent producing population of numbers where number possessing higher fitness value is replicated more. Thus, making GA a very good contender for PRNGs. Good Fitness function helps in exploring search space of random numbers in more efficient manner. GA PRNGs result samples satisfies frequency test and gap test. Thus the numbers generated after each iteration by GA PRNG are statistically verified to be random and nonrepeating, having no prior relation of next number from the previous ones, acting as an essential initialization parameter for neural algorithm overcomes the problem of acknowledging the random number generated by traditional PRNG. For generating public and private keys, different number of rounds of mixing is used. This ensures that the private key generated cannot be derived from public key. Our algorithm was observed to give fast and improved performance results having practical and feasible implementation.**

Keywords*—*artificial neural networks, neural cryptography, hebbian theory, genetic algorithm, public key cryptography, random number.

I. Introduction

Cryptography is the science of encryption and decryption of data. It allows secure communication across insecure networks like Internet and storage of secret data. Public Key Cryptography (PKC) is an asymmetric approach that uses a pair of keys–public key for encryption and private key for decryption. A key is used to produce a specific cipher text in cryptographic techniques. Keys are large numbers whose size is measured in bits. In PKC, larger keys ensures safe cipher text. Selected Keys should be big so as to be secure, but should be small for faster access.

Artificial Intelligence (AI) is a branch of computer science that emphasis on developing intelligent machines and software using applied logic. It claims the simulation of intelligence of humans by a machine by employing reasoning, learning, communication and manipulation. AI is found to have intense applications in various fields. One such sub field, Neural Networks and its implementation in the area of cryptography is discussed in this paper.

An Artificial Neural Network (ANN) is a mathematical model consisting of an interconnected group of artificial neurons motivated by the working of brain. The brain learns from experience and adaptation to environment. Also, inter neuron connection strengths, called weights, are used to store the acquired knowledge. ANN is a nonlinear parallel adaptive system that is used to model variegated relationships between inputs and outputs. The output of a unit is decided by I/O characteristics while the overall working of ANN is determined by its structure and the training algorithm.

Advantages of ANN include adaptive interaction between elements, self-organization, real time operation, parallel computation, and Fault Tolerance. ANN helps in handling complex problems and are used in robotics, pattern recognition, medicine, manufacturing, optimization, signal processing, system modeling & identification, control of power-generation systems.

II. Structure And Working of Artificial Neural Networks

A. Network Parameters

ANN parameters includes number of training iterations, the number of hidden neurons (K), the input layer units for each hidden layer unit (N), range of synaptic weight values (MIN-MAX) {-L...+L}. Neurons which are not present in input and output layer are hidden from view. Presence of hidden neurons enhances the flexibility of system and increases processing power, but if the number of hidden neurons is taken too small then robustness of the system can reduce due to improper fitting of input data.

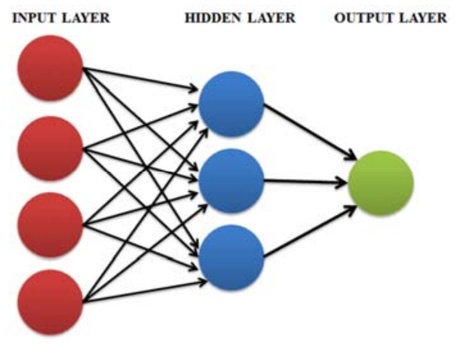

Fig. 1. An Artificial Neural Network.

B. *Learning*

Learning is an essential part of an intelligent system. A learning rule is used to train the system by updation of the inter-neuronal synaptic weights during each training iteration. After learning phase, the process of production of independent results starts. Networks which are able to continue learning during this process are known as dynamical systems. Learning can be supervised, unsupervised or hybrid.

C. *Feed-Forward Neural Network*

A feed-forward neural network is a Neural Network (NN) where no directed cycle is formed in between the units and the information moves in only one direction, from input nodes to the output nodes via the hidden nodes. Such ANN with continuous output operates on back-propagation which is a supervised learning method.

D. *Tree Parity Machine*

It is a multi-layer feed-forward NN consisting of single output nodes, K hidden nodes and K*N input nodes. Inputs to NN (x_{ij}) are in binary form {-1, 0, +1} and the weights between input and hidden neurons, w_{ij} {-L,... , 0 , ..., +L}. Calculation of Output value of each hidden node is done by summing all products of NN input neurons with their weights. The binary output of neural network is then calculated by multiplying all values obtained by hidden neurons. Synchronization can be done using learning rules such as Hebbian rule, Random walk rule and Anti-Hebbian rule.

Neural networks possess ability to derive meaning from complex data and detect trends that are too complex to be recognized by computer systems. Synchronization of two tree parity machines can be used to exchange keys between two machines A and B.

III. Introduction to Genetic Algorithms

The main idea behind Genetic Algorithms (GAs) is to replicate the randomness of the nature where population of individuals adapts to its surroundings through natural selection process and behavior of natural system. GAs produce a population in such a way that the trait which is dominant, that is has higher fitness value is replicated more likewise rest is rejected based on threshold which is then evolved by the iterative application of a set of stochastic operators like mutation, crossover, and selection. Highest rank signifies the better fitness. The results obtained should be good in terms of coefficient of autocorrelation. The samples satisfy the tests including gap test, and frequency test. A simple GA makes use of following operators to transform a population into new population with better fitness :

A. *Crossover*

Crossover is a genetic operator that helps in joining two chromosomes to form a new chromosome. Number of crossovers depends on crossover-rate. Generally crossover rate is 2 to 5%. For binary string individuals, one-point ring crossover, two-point, and uniform crossover are often used.

B. *Mutation*

Mutation is a genetic operator which changes one or more bit values in a chromosome. It is performed on a child after crossover which guarantees the entire state-space will be searched.

C. *Selection*

Selection is the stage of GAs in which individual chromosomes are chosen from a population for crossover. The chromosome with higher fitness value will be considered better.

D. *Fitness Function*

The fitness function plays a vital role in guiding GA. Good fitness functions will help GA to explore the search space more effectively and efficiently. Bad fitness functions, can easily make GA get trapped in a local optimum solution and lose the discovery power.

IV. Literature Review

Many Research papers dealing with Neural Networks and key generation in cryptography have been studied and analyzed to get an idea of the previous attempts made in this field. It was found that the use of ANN along with GA has not as yet been explored. It is essential for a key to possess randomness for key strength and security. Hence, making the code hard to break.

One of the paper presents the generation of secret key over a public channel which is simulated and synthesized using VHDL[1]. In another paper, synchronization of neural networks is used for exchanging keys [2]. Regarding GA, in one of the work, it is used for searching the key domain of encryption method [3]. Also, the concept of GAs has been described in one of the thesis, GAs in Cryptography by Bethany Delman [4].

Some of the commonly used techniques for generating unique keys between two communicating machines in cryptography are one time pad and Pseudo Random Number Generator (PRNG). But PRNG's are found to have many types of possible attacks as discussed in [5]. An attacker may cause a given PRNG to fail to appear random, or ways he can use knowledge of some PRNG outputs (such as initialization vectors) to guess other PRNG outputs (such as secret key).

Hence to overcome this drawback, we have proposed the use of ANN along with GA to generate the keys for PKC. GA helps in providing Input vectors to each iteration of ANN, as GA itself operates on random population so the corresponding parameters are also always unique, non-repeating and cannot be deduced or acknowledged by the attacker. Hence, the generated public and private keys by ANN are purely random between the synchronized machines in its process.

V. Algorithm for Pseudo Random Number Generation Using Genetic Algorithm

Step 1. Initial Population Generation : In GA first we obtain a randomly generated group of individuals which is called initial population. Initial population array having values of 7 bit each respectively. Each bit is randomly assigned 0 or 1 bit value corresponding to a random generator function. If value obtained from generator is greater than 50 then bit value 1 is assigned otherwise 0 is assigned. All 7 bits of chromosomes cell values are randomly assigned. Size of initial 2D population array depends on value of MAX_POPULATION which is defined as macro.

Data Structure used : *initPop*[MAX_POPULATION][7], *finalPop*[MAX_POPULATION][7].

Step 2. Calculation of Chromosome Number and Threshold Check : Each chromosome should meet a threshold standard. This means that above average chromosomes should have more copies in the population, while below average chromosomes are subjected to extinction based on threshold. For each chromosome, a number is calculated, if it is found to be greater than threshold then the corresponding chromosome is selected otherwise rejected. This threshold check is also performed in later stages.

Step 3. Now the GA enters a loop. At the end of each iteration, stochastic operators are applied to the previously obtained population for generation of new population. Each such iterative step is a generation.

Step 4. Selection and Crossover : First a selection operator is applied, in which two parents are selected randomly from initial Population. The selected parents are used to produce the individuals for next generation through the application of crossover operator. To apply a crossover operator, parents are paired together using one-point Ring Crossover. The Two parents are combined in the form of ring and a random cut point is generated.

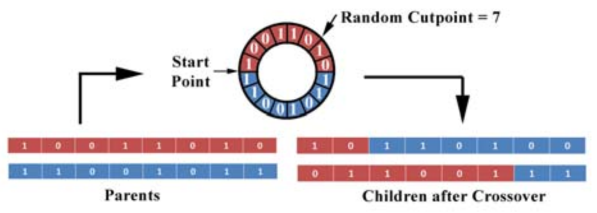

Fig. 2. Ring Crossover.

With reference to this cutting point, one of the children (C1[ ] array) is created in the clockwise direction, while the other one (C2[ ] array) is created in anti-clockwise direction, as shown in Fig. 2. Each of child bits must pertain a value from corresponding bit in parents and should exhibit valid permutation. After Crossover, threshold check is performed.

Step 5. Mutation : Next, apply the mutation operator in which a child is randomly changed from what its parents produced in crossover. Mutation rate taken as 0.5%. Number of Mutation is given by :

$$\frac{(No.of\ cells\ in\ a\ chromosome * No.of\ chromosomes * mutation\ rate)}{10}$$

= (7 * 10 * 0.5) / 10 = 3.

For each mutation, select bit number to be modified using random number function : rand()%7 (so one bit is only selected and flipped between 0 and 1). Threshold check is performed after mutation.

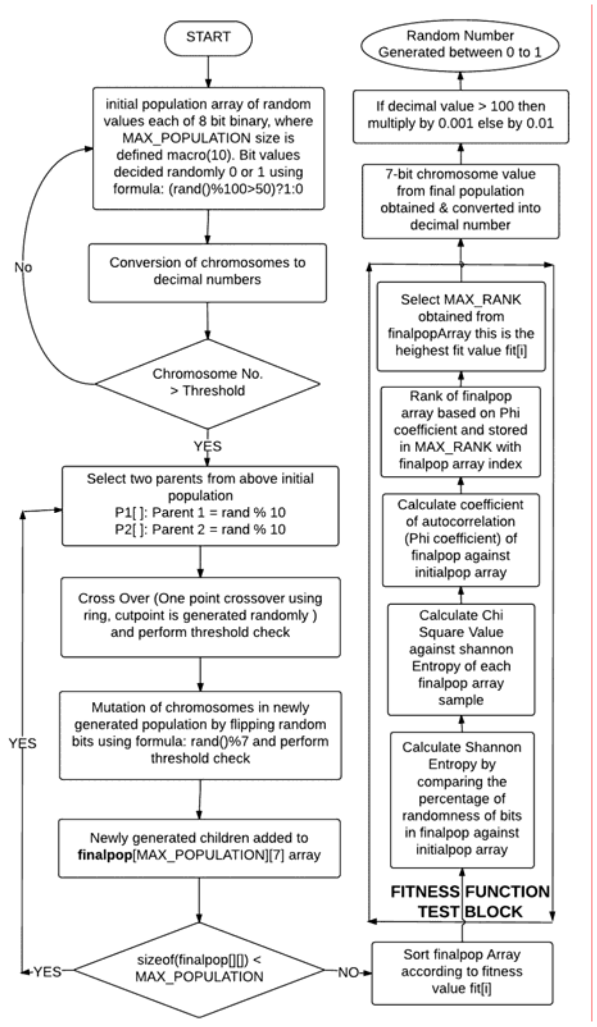
START
initial population array of random values each of 8 bit binary, where MAX_POPULATION size is defined macro(10). Bit values decided randomly 0 or 1 using formula: (rand()%100>50)?1:0
Conversion of chromosomes to decimal numbers
Chromosome No. > Threshold
No
YES
Select two parents from above initial population P1[ ]: Parent 1 = rand % 10 P2[ ]: Parent 2 = rand % 10
Cross Over (One point crossover using ring, cutpoint is generated randomly) and perform threshold check
Mutation of chromosomes in newly generated population by flipping random bits using formula: rand()%7 and perform threshold check
Newly generated children added to finalpop[MAX_POPULATION][7] array
YES
sizeof(finalpop[][]) < MAX_POPULATION
YES
NO
Sort finalpop Array according to fitness value fit[i]
FITNESS FUNCTION TEST BLOCK
Calculate Shannon Entropy by comparing the percentage of randomness of bits in finalpop against initialpop array
Calculate Chi Square Value against shannon Entropy of each finalpop array sample
Calculate coefficient of autocorrelation (Phi coefficient) of finalpop against initialpop array
Rank of finalpop array based on Phi coefficient and stored in MAX_RANK with finalpop array index
Select MAX_RANK obtained from finalpopArray this is the heighest fit value fit[i]
7-bit chromosome value from final population obtained & converted into decimal number
If decimal value > 100 then multiply by 0.001 else by 0.01
Random Number Generated between 0 to 1

Fig. 3. Flowchart for Random No. Generation using Genetic Algorithm.

Step 6. Calculation of Fitness of Random Number : Repeat the above mentioned steps until the final population array gets full. Chromosomes in final population are arranged according to their fitness values and the chromosome with the highest fitness value is selected. Fitness value is calculated using following steps :

1) Calculate Shannon Entropy H(X) by comparing the percentage of randomness of bits in *finalPop*[][] against *initialPop*[][] array using the formula :

$$H(X) = -(p * \log_2(p)) - ((1 - p) * \log_2(1 - p)) \qquad (1)$$

where p denotes percentage of randomness of *finalpop*[i][j] from *initialpop*[i][j] i.e. degree of movement of each bit.

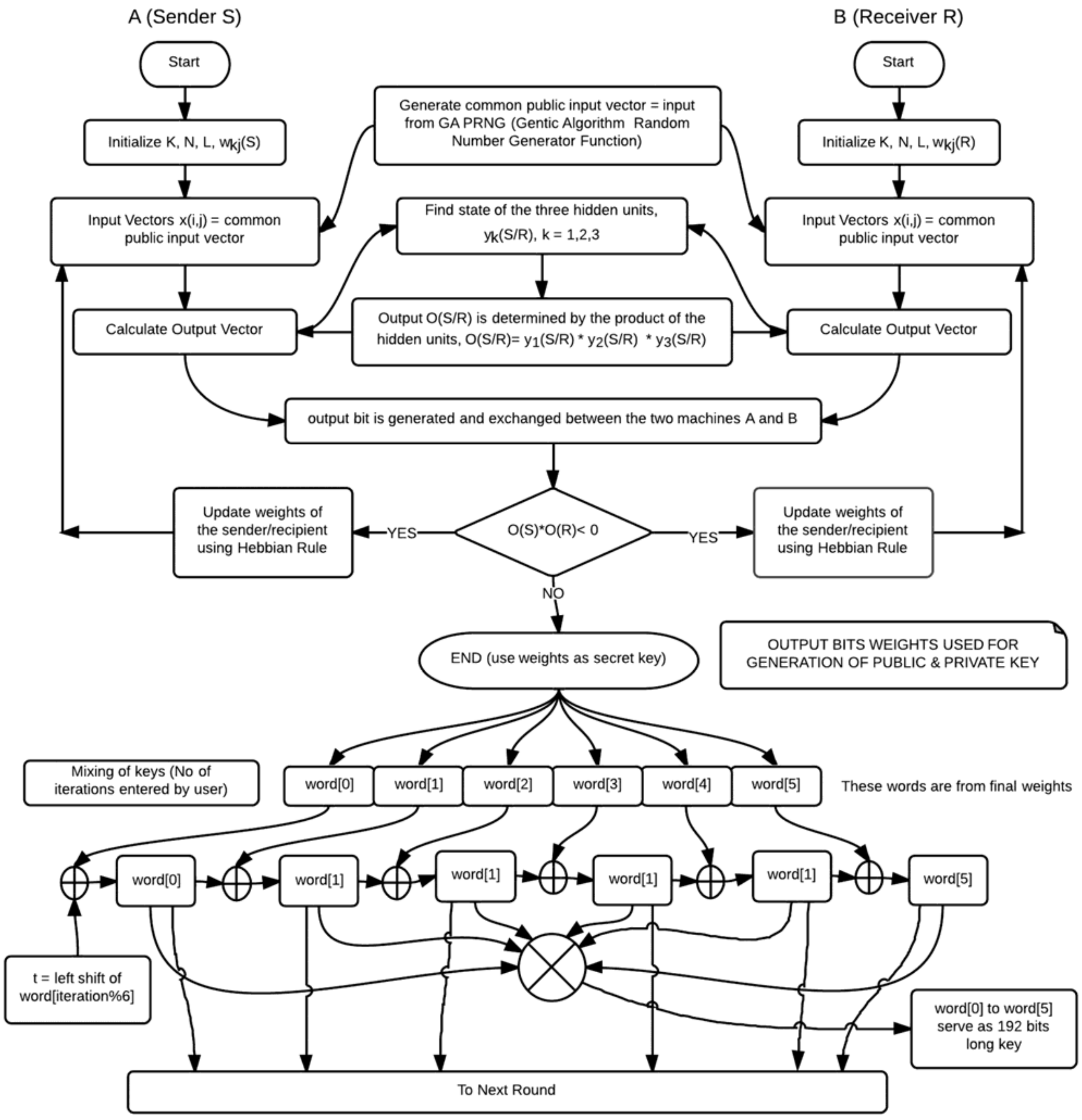
A (Sender S)
B (Receiver R)
Start
Start
Generate common public input vector = input from GA PRNG (Gentic Algorithm Random Number Generator Function)
Initialize K, N, L, w_{kj}(S)
Initialize K, N, L, w_{kj}(R)
Input Vectors x(i,j) = common public input vector
Find state of the three hidden units, y_k(S/R), k = 1,2,3
Input Vectors x(i,j) = common public input vector
Calculate Output Vector
Output O(S/R) is determined by the product of the hidden units, O(S/R)= y_1(S/R) * y_2(S/R) * y_3(S/R)
Calculate Output Vector
output bit is generated and exchanged between the two machines A and B
Update weights of the sender/recipient using Hebbian Rule
YES
O(S)*O(R)< 0
YES
Update weights of the sender/recipient using Hebbian Rule
NO
END (use weights as secret key)
OUTPUT BITS WEIGHTS USED FOR GENERATION OF PUBLIC & PRIVATE KEY
Mixing of keys (No of iterations entered by user)
word[0]
word[1]
word[2]
word[3]
word[4]
word[5]
These words are from final weights
word[0]
word[1]
word[1]
word[1]
word[1]
word[5]
t = left shift of word[iteration%6]
word[0] to word[5] serve as 192 bits long key
To Next Round

Fig. 4. Flowchart for Public Key Cryptography using Neural Networks and Genetic Algorithms.

2) Calculate Chi Square value against Shannon Entropy of each *finalPop*[][] array sample using the formula :

$$X^2 = (observed\_H - expected\_H)^2 / (expected\_H) \qquad (2)$$

where, *observed_H* is calculated using (1) and *expected_H* is calculated for chromosome with equal number of 0's and 1's.

3) Calculate the Coefficient of Auto-correlation (Phi-coefficient) of *finalpop*[][] against *initialpop*[][] array.

$$\text{Phi-Coefficient of Auto-correlation, } \varphi^2 = X^2 \div n \qquad (3)$$

where X is chi-square value and n is sample space (n = 10).

This coefficient of auto-correlation serves as fitness function and is used to find best fit random number.

Step 7. Ranking of Chromosomes on the basis of Phi-coefficient : Calculate Ranks of chromosomes in *finalpop*[][] array based on Phi-coefficient and store the index of chromosome with maximum rank in *MAX_RANK* variable which serves as rank of maximum fit chromosome.

Step 8. Chromosome with *MAX_RANK* is 7 bit binary number which is converted to decimal value. If decimal value obtained is greater than 100 then random number is generated by multiplying decimal value with 0.001, otherwise 0.01 is multiplied.

Step 9. A random number between 0 and 1 is generated which acts as an essential initialization parameter for neural algorithm described in subsequent steps.

VI. Public Key Generation using Neural Networks & Genetic Algorithm

Step 1. Machine A and Machine B uses its own ANN whose parameters are initialized as follows :

The number of hidden neurons (K) is usually between 3 to 5, the input layer units for each hidden neuron (N) range from 5 to 10, the range of synaptic weight values is {-L...+L}.

Step 2. The network weights are initialized randomly using GA PRNG as stated above. Weights for Sender (w_{kj}(S)) and Weights for Receiver (w_{kj}(R)) are defined between {-L...+L}.

Step 3. Following steps are repeated until synchronization occurs :

1) A new common PUBLIC input vector x_{kj} is obtained using GA PRNG which is required by both the sender and the recipient. The Input Vector of the hidden units (denoted by x_{kj}) is utilized for the synchronization as well as for the encryption/decryption step.

2) Calculation of O/P Vector : Output Vector is calculated in two steps. In the first step, the state of the three hidden units, y_k(S/R), k = 1, 2, 3 of the sender and the recipient are determined using formula :

$$y_k(S/R) = sign\left(\sum_{j=1}^{N} w_{kj}(S/R)x_{kj}\right) \qquad (4)$$

where x_{kj} denotes input vector and w_{kj} denotes weights.

Signum function returns -1, 0 or 1. If product is found to be 0, then value of output of the hidden node equals -1 to ensure a binary value.

In the next step the output O(S/R) is determined by the product of the state of hidden units using formula :

$$O(S/R) = y_1(S/R) * y_2(S/R) * y_3(S/R) \qquad (5)$$

3) Comparison of Output Vectors from two branches of machine A and B respectively : The sender sends its output bit to the recipient, the recipient sends its output to the sender and both networks are trained with the output of each other. In the event that they do not agree on the current output i.e. when O(S)O(R) < 0, the weights of the sender and recipient are updated according to the following Hebbian learning rule :

```
if ( $O(S/R)\, y_k(S/R) > 0$ ) then
        $w_{kj}(S/R) = w_{kj}(S/R) - O(S/R)\, x_{kj}$
if (| $w_{kj}(S/R)$ | > L) then
        $w_{kj}(S/R) = sign(w_{kj}(S/R))\, L$
```

In each of the two machines, only the weights belonging to one of the three hidden units which are in the same state as that of their output unit are updated. Note that by using this dynamical rule, the sender is trying to imitate the response of the recipient and the recipient is trying to imitate that of the sender. This method is known as Bidirectional Learning.

4) After the completion of synchronization (weights w_{ij} of both tree parity machines are same), the weights obtained are used for generation of public key and private key in subsequent steps.

Step 4. word[6][32] array is initialized with final weights obtained after synchronization in above step.

Step 5. Mixing of Keys : Iteration are directly proportional to the degree of complexity as more number of rounds results in good mixing. User specifies the number of iterations (*no_of_iterations*) to be performed. Two random numbers in range [1, *no_of_iterations*] are generated using GA PRNG. One of the random number serve as no of iterations required to obtain public key and the other serve as number of iterations required to obtain private key. It should satisfy the condition that number of iteration for public key should not be equal to the no of iteration for private key. It should be noted that, after every iteration, 192 bit keys are generated.

Step 6. Thus, 192-bit public and private keys are obtained which are random in nature.

VII. Results And Analysis

Tests Performed on GA PRNG :

1) Frequency test (Chi-Square Test) : The chi-square test is used to determine whether there is a significant difference between the expected frequencies and the observed frequencies in one or more categories. In the case of binary sequences, it focuses on the relative frequency of 1's with respect to 0's. Goodness of fit determines whether or not an observed frequency distribution of random number is generated.

TABLE I. CHI-SQUARE TEST RESULTS FOR RANDOM NUMBERS GENERATED USING GENETIC ALGORITHM

Observed (d1)	Expected (d2)	$p = \frac{(d1-d2)}{(d1+d2)}$	H(X) observed	H(X) expected	X^2	$\varphi = \sqrt{\frac{X^2}{n}}$
85	21	0.60377358490566	0.96870034365086	0.5	0.439360024276869	0.296432125208072
49	91	0.3000000000000	0.881290899230693	0.5	0.290765499672301	0.241149538532546
16	100	0.724137931034483	0.849751137253297	0.5	0.24465171601995	0.221202041590917
11	20	0.290322580645161	0.869137580612638	0.5	0.272525106841104	0.23346310493999
32	113	0.558620689655172	0.990061847024213	0.5	0.480321227817566	0.309942326189105

TABLE II. GAP TEST RESULTS FOR SAMPLES OF RANDOM NUMBERS[0,1] GENERATED USING GENETIC ALGORITHM

Gap length	Frequency = f	F*0.01	Cumulative Freq S(N)	F(N)	| S(N) – F(N) |
0-4	6	0.06	0.06	0.40951	0.34951
5-9	3	0.03	0.09	0.6513215599	0.5613215599
10-14	2	0.02	0.11	0.794108867905351	0.684108867905351
15-19	1	0.01	0.12	0.878423345409431	0.758423345409431
20-24	1	0.01	0.13	0.928210201230815	0.798210201230815

From Table I, the average value of Chi-Square was found to be 0.34 which satisfies and lies under the limits of chi square distribution table i.e. 6.63. The maximum rank was found to be 0.096 hence corresponding index 5 was selected as best fit for the random number. Thus the numbers generated after each iteration by GA PRNG are statistically verified to be random and nonrepeating.

2) Gap Test : Gap test counts the number of digits that appear between repetitions of a particular digit. It examines the length of gaps between occurrences of sample values and determines the length of consecutive subsequences with samples not in a specific range.

As observed from Table II, For alpha = 0.05, $D_{critical}$ was found to be 1.96 which is less than $D_{MAX} = 0.798$ and hence the null hypothesis is proved to be true. Thus, successful results are obtained for gap test.

VIII. CONCLUSIONS

We proposed the use of Genetic Algorithm with Artificial Intelligence (AI) for key generation process. It was found that GA in PRNGs for initialization of input parameters in ANN has valid application overcoming the problem of acknowledging the random number generated by traditional PRNGs in ANN which made it vulnerable to attackers after discovery of patterns in these random values. In each iteration, GA PRNG "unpublished" [6] helps ANN to keep the keys generated secure having high randomness. It was analyzed that the next random number generated cannot be predicted from previously generated random number. Also it can never be the case that the value of all weights are equal. The keys are obtained from initial word[6][32] by mixing using iterations.

Iteration are directly proportional to degree of complexity as more number of rounds results in good mixing. So any value of 192 bit ranging from 2^1 to 2^{192} is possible. Algorithm is secured from attack by adversary as to run the algorithm, two machines need to be synchronized and in each run of the algorithm, new set of keys are produced so even if the adversary synchronizes with sender and receiver separately, it would not be able to produce the same set of keys as present initially between sender and receiver. The new set of keys would be only applicable to communication between the adversary and sender or adversary and receiver. Thus, the adversary still cannot intercept the original message.

The results of frequency and gap test performed on GA PRNG of ANN are encouraging and verifies that the keys generated by ANN are random and nonrepeating in nature.

ACKNOWLEDGMENT

The authors thank and gratefully acknowledges the support of Miss. S. Jhajharia, Visiting Faculty DTU, Delhi.

REFERENCES

[1] T. Godhavari, N. R. Alainelu and R. Soundararajan "Cryptography using neural network," IEEE INDICON 2005 Conf., Chennai, India, pp. 258-261, 11-13 Dec. 2005.

[2] R. Mislovaty , E. Klein , I. Kanter and W. Kinzel "Security of neural cryptography," 11th IEEE International Conf. on Electronics, Circuits and Systems, pp.219-221, 13-15 Dec. 2004.

[3] S. S. Omran, A. S. Al-Khalid, D. M. Al-Saady, "A Cryptanalytic attack on Vigenère Cipher using Genetic algorithm," IEEE Conf. on Open systems, pp.59,64, 25-28 Sept. 2011.

[4] B. Delman, "Genetic algorithms in cryptography," M.S. thesis, Rochester Institute of Technology, Rochester, New York, July 2004.

[5] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Cryptanalytic attacks on pseudorandom number generators", Fast Software Encryption, Fifth International Workshop Proc. (March 1998), Springer-Verlag, 1998, pp. 168-188.