Research Article

*ACCENTS Transactions on Information Security, Vol 2(8)*
*ISSN (Online): 2455-7196*
*http://dx.doi.org/10.19101/TIS.2017.28003*

# Security analysis and improvement to permutation parity machine

**Vidya M H[1*], Ganesan P[2], Sathyanararyana S V[1] and G K Patra[2]**
Jawaharlal Nehru National College of Engineering, Shivamogga, Karnataka, India[1]
CSIR Fourth Paradigm Institute, Bengaluru, Karnataka, India[2]



## Abstract
***Security is a major aspect of communication in every day to day life. Neural cryptography is a non-classical cryptographic technique which aims to derive a common key through learning mechanism between two neural networks. It makes use of lightweight permutation parity machines (PPM) to generate a common secret key over a public channel. The basic arithmetic computation on PPM leads to secret key generation between two entities through the process of synchronization. However, the process can be impersonated by an attacker using probabilistic attack on PPM. The proposed technique mitigates the probabilistic attack using feedback mechanism for the synchronization of PPM.***

## Keywords
***Neural cryptography, Probabilistic attack, Permutation parity machine.***

## 1.Introduction
In this era, building a secure channel is one of the major challenging areas of research and development in modern communication. Cryptography is the practice of ensuring confidentiality, integrity and availability [1-5]. It is a crucial viewpoint for secure correspondence. Cryptography permits two parties to communicate over an insecure channel so that attacker cannot perceive and decode the original message. Public key cryptography is accessible in several forms however it needs an immense time consumption, complexity and huge processing power and thus it is unsuitable for energy constraint devices [6-8]. An artificial neural network (ANN) will be the most effective way to overcome these issues in such devices. The association between an ANN and cryptography is providing the good facilitate for the protection considerations [9].Neural cryptography is a branch of cryptographic technique which makes use of concepts of ANNs for secure key agreement via public channel. Some of the well-known ANNs for neural key exchange through insecure channel are tree parity machine (TPM) and permutation parity machine (PPM). In this approach, the participants only need to perform basic arithmetic operations instead of complex number theoretic approach. The two participants who mutually exchange their input and output bits in public domain can securely synchronize each other by Permutation Parity Machine architecture [1, 2].

*Author for correspondence

This class of light-weight cryptography model should not compromise in security with the effect of probabilistic attacks. The proposed work mainly concentrates on overcoming probabilistic attack.The Permutation Parity Machine is an artificial, multi-layer feed-forward neural network proposed as a binary variant of Tree Parity Machine as shown in *Figure 1*. PPM consists of three layers input layer, hidden layer and output layer.

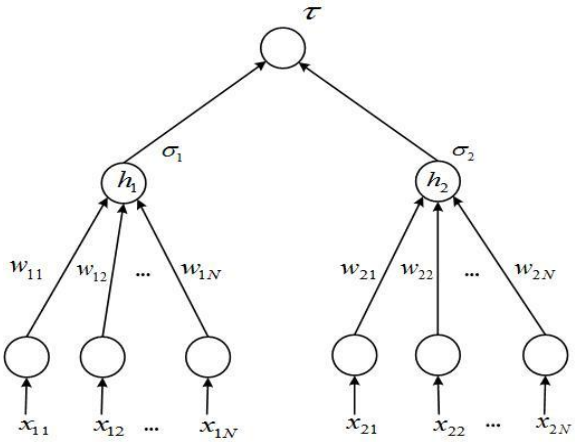


**Figure 1** General structure of a permutation parity machine

Each hidden units in the hidden layer has N input neuron and one output neuron. Information is transmitted from one neuron to other neuron through weights, which can be altered using learning

mechanism such that network refine to the required task. The input vector, permutation matrix and output bit of PPM are public parameters where the state vector and hidden outputs are private parameters [1, 8].

The genuine participants having the same PPM structure with parameter G, K and N begins the synchronization pro-cess by exchanging the output of their machines over public channel [8]. Input layer takes binary input {0, 1} and an initial random weight vector is assigned by the entries of the state vector according to the position given by the matrix entries as per equation (1). The output of the hidden unit is computed using the equation (2). The output of the network is computed using the equation (3).

$$w_{ij} = s_{\pi_{ij}}; 1 \le i \le N\,; 1 \le j \le K \quad (1)$$

$$\sigma_j^{A/B} = sign\left(\sum_{i=1}^{N} x_{ij} \oplus w_{ij}\right), 1 \le j \le K \quad (2)$$

$$\tau^{A/B} = \oplus_{j=1}^{K} \sigma_j \quad (3)$$

When the outputs of both the machines are equal then they train the network by applying learning rule. The learning rule involves two rounds inner round and outer round. An outer round comprises of series of inner round that are used to store output of the first hidden unit $\sigma_1^{A/B}$ in the correspondingempty position of the buffer of size G [2]. When the buffers are completely permeated, the outer round replaces the state vector by the respectively permeated buffer. Thus, two PPMs converge to identical weights which are interpreted as symmetric keys for encryption and decryption. After achieving full synchronization, A and B use the synchronized state vectors as common secret key [8].

Some of the attacks such as simple, geometric and majority attacks are proposed for neural cryptography, tries to mimic the synchronization process to guess the state vector of one of participants using single or ensemble of adequately many machines. These attacks show outstanding performance in TPM. Each attacks use a permutation parity machine that has the same structure and parameters such as K, N and G as those of the participants A and B. The attacker's machine also starts with randomly chosen state vector and applies the same learning mechanism as the partners A and B. These attacks give poor performance in guessing state vector of one of the participants. In case of TPM the effects of the training rule on the group of machine will be assessed right ahead whereas for PPMs, it is necessary to attend an entire outer round before any result will be ascertained. Hence these attacks are most suited for TPM. Recently proposed probabilistic attack on neural cryptography does not mimic the synchronization process, instead tries to guess the state vector of one of the participant by taking sample of weights in a space using Monte Carlo approach and transfer the information acquired in the preceding outer round to the next one using analytical approach [3].

The rest of the paper is organized as follows. In section II Probabilistic Attack, a successful attack on PPM is explained in detail. Section III describes the proposed algorithm used for improvement of PPM. Section IV discusses the security analysis of proposed algorithm. Finally conclusions are drawn based on the results in section V.

## 2.Probabilistic attack

Probabilistic attack employs different methods to break the secure key exchange performed using Permutation Parity Machine. The most successful methods are Monte Carlo simulation in predicting values of the inner rounds and binomial distribution in transferring knowledge obtained during a complete outer round $P^E$ to a next outer round $P^{E+}$. The architecture and parameters of attacker's PPM is equivalent to the synchronizing machines $PPM^{A=B}$.

In this approach, the attacker chooses a $PPM^E$ which has sufficiently large M state vector (S) to simulate inner rounds using Monte Carlo approach. The weight vector W is selected from the state vector S using position of the matrix. $PPM^E$ has parameters (G, K, N) equal to $PPM^{A=B}$. Additionally, the eavesdropper has probabilistic vector $P^E$ which is initialized to $^1/_2$at the beginning of the attack since there is no prior knowledge obtained about the synchronization [3], [4].

Whenever the participants A and B communicate $\tau^{A/B}$ in an open insecurechannel, the eavesdropper uses the corresponding public parameter X vector, $\tau^{A/B}$and $\pi$ to compute M valid state vectors i.e., $\tau_i^E = \tau^{A/B}$where i = 1, … ,M. The M state vector ($s(\pi_{ij})$) are sampled using Bernoulli distribution with the probabilities P ($s_i$ = 0) = $p_i$ and P ($s_i$ = 1) = $1 - p_i$

Then, the marginal posterior probability $P(s_i = 0|p^E, X, \pi, \tau^A)$ can be calculated by counting the

number of 0s in M state vectors at position i where i = 1, … ,M. The results are updated to the respective position in the probability vector $P^E$ based on π. This process is continued for subsequent inner rounds and $P^E$ is updated at each rounds. If any probability $p_i$ is collapsed to wrong value, M valid state vectors cannot be obtained as the Bernoulli sampling runs in an infinite loop. To overcome this, the probability $p_i$ which has closed to either 0 or 1 should be reset to neutral probability$1/2$. This avoids the sampling process to run in an infinite loop [4].

At the end of G inner rounds, the values of state vector G is replaced with values of temporary buffer B. Hence the attacker needs to transfer the knowledge obtained from the previous outer round to the next outer round. This is done by finding the probability distribution of $\sigma_1^E$ at each inner round conditioned on the parameters matrix, probability state vector $P^E$, input vector X. As the probability state vector $P^E$ provides information about occurrence of 0s in the inner rounds, the posterior probability P ($\sigma_j$= n| $P^E$, X,π) should be calculated for the occurrence of 0s for that inner round. The value of 0 in the hidden unit at a particular inner round depends on the number of 1s at the local field $h_j$ which is equal to the summation of $x_i \oplus s(\pi_{ij})$. Therefore the average probability of finding a 1 is calculated as per the equation 4.

$$q_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}\, p_{\pi_{i,j}} + (1 - x_{i,j})\left(1 - p_{\pi_{i,j}}\right) \qquad (4)$$

Here j indicates the inner round. The probability of $\sigma_j = 0$ is calculated using binomial distribution by substituting $q_j$ in equation 5.

$$P(\sigma_j = 0 | p^{E-}, X, \pi) = \sum_{n=0}^{N/2} \binom{N}{n} q_j^n (1 - q_j)^{N-n} \qquad (5)$$

Equation 5 gives the result of $P^{E+}$at the end of an outer round, these results are stored and replaced to the probability vector $P^E$. Thus, the probability information about the first hidden unit at the previous outer round is transferred to the next outer round. The Bernoulli sampling procedure is followed in the next outer round using the obtained probability $P^{E+}$. When this process continues, the probabilistic attack is considered to be success as soon as the state vector $S^E$ is equal to the state vector $S^A$, it is known as break time $t_b$ of an attack.

Break time $t_b$ is represented in the *Figure 3*, which shows the synchronization time of participants $t_s$, break time of an attack $t_b$ plotted with the parameters number of input (N) and outer round. The *Figure 3* clearly shows that the attack is faster than the genuine participant's synchronization i.e., $t_b < t_s$.

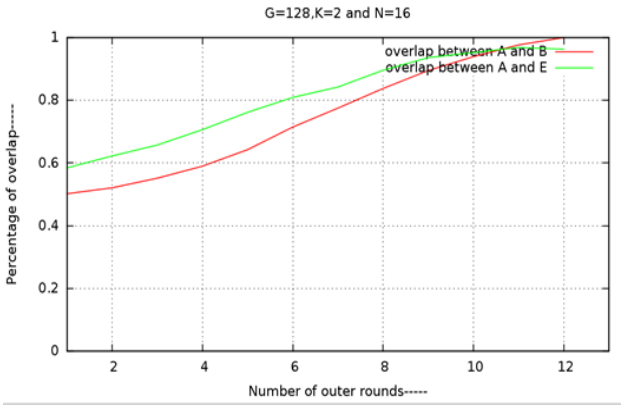


**Figure 2** Overlap between an attacker and participant as a function of outer rounds for the parameter G=128, K=2 and N=16 averaging similar outer rounds

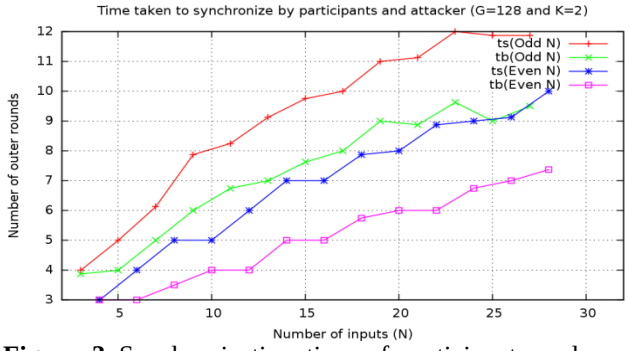


**Figure 3** Synchronization time of participants and attacker measured in outer rounds as a function of N for the parameter G=128 and K=2 in probabilistic attack. Results are averaged over 1000 runs

In *Figure 2*, which depicts the percentage of overlap of the probabilistic attack as a function of outer rounds for parameter G=128 and K=2, shows that the attacker's percentage of overlap reaches to 1.

In *Figure 4*, the performance of the algorithm is discussed in terms of probability of success $P_s$ and number of input N for the values G=128 and K=2. The probability of success of the attacker touches the maximum probability 1 for each and every value of N.

The probabilistic attack suggested in PPM uses an efficient method instead of imitating the learning rule to guess the secret key. The performance of the attack

is presented in terms of probability of success and number of outer rounds for synchronization. Accordingly, the results clearly shows that PPM based neural cryptosystem analyzed here is not secure enough for any cryptographic applications. The proposed method, feedback mechanism in PPM, concentrates on these weaknesses of PPM and tries to mitigate those weaknesses.

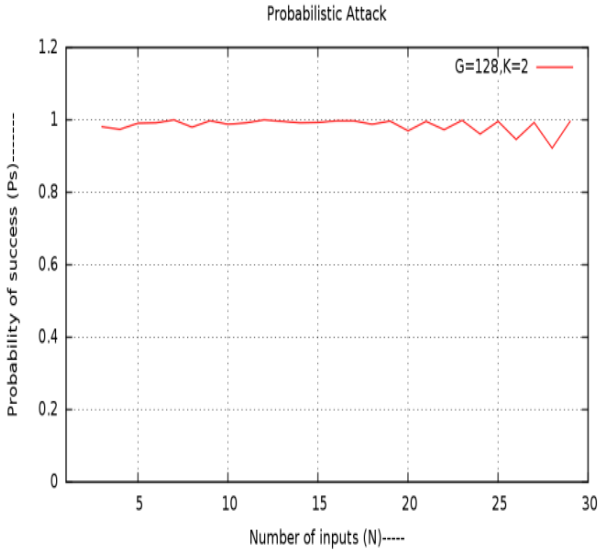


**Figure 4** Probability of success of an attacker as a function of N for the parameter G=128 and K=2 in probabilistic attack, Results are averaged over 1000 runs

## 3. PPM using feedback mechanism

The neural cryptography is based on fragile competition n between stochastic attractive and repulsive behaviour of ANNs. When this competition is vigilantly balanced, two participants A and B are able to synchronize whereas an eavesdropping machine E has only a remarkably low probability to find the common state vector of the communicating participants [5].

The input vectors X are crucial source of information for the attacker to synchronize with one of the participants. So, keeping input vector X at least partly secret ought to improve the protection of the neural key-exchange protocol[5]. With the intention of making the few bits of input vector private, proposed a feedback mechanism in PPM as shown in *Figure 5*.

Here we examine a mechanism which decrease the probability of success of attacker by incorporating feedback in the neural network. The input vector X is no longer an identical public random number instead few bits in X are made private by the corresponding output of hidden units and parity of weights of each perceptron. At the initial synchronization steps value at the hidden units and parity of weights of the two participants might be distinct. To allow the synchronization between the participants they have to receive common inputs for some interval. Consequently, after some synchronization steps few bits of input vector are made private, during this intermission the mutual synchronizing PPMs have achieved fair synchronization between them. The parity of weight vectors of each perceptron is computed using equation (6).

$$p_j = \oplus_{i=1}^{N} s[\pi_{ij}], 1 \le j \le K \quad (6)$$

Where $p_j$ is the parity of each perceptron. The following algorithms explain the PPM synchronization with feedback input.

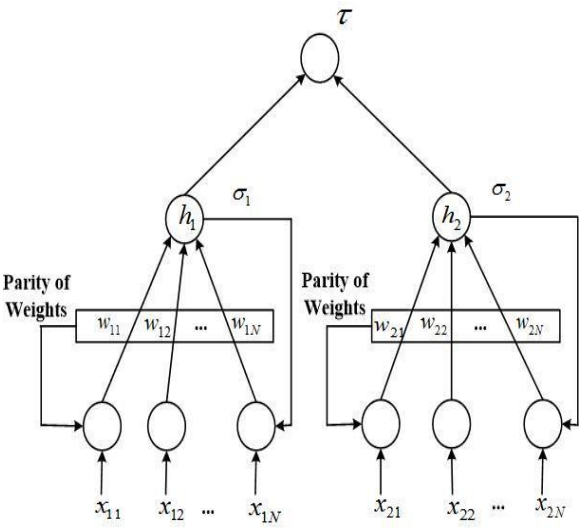


**Figure 5** General structure of PPM with feedback

### A. Algorithm 1: PPM with feedback input

//Synchronization of PPM using feedback input.
//Input:
Private: Random state vector $s^{A=B}$, hidden neurons $\sigma^{A=B}$
Public: Random input to PPM $X^{A=B}$,
Permutation matrix $\pi^{A=B}$, output $\tau^{A=B}$
//Output:

Synchronized state vector $s^{A=B}$.
//Begin:
Initialize the random state vector $s^{A=B}$

**While** # iterations <boot_N**do**

X=rand(X)
$\pi$=rand ($\pi$)
Counter ← 0, $B_{buf} = 0$
**for** j ← 1 **to** K **do**
$\sigma_j = \theta\left(\sum_{i=1}^{N} s[\pi_{ij}] \oplus x_{ij}\right)$//Where $\theta(h) =$
$\begin{cases} 1, & h > N/2 \\ 0, & h \leq N/2 \end{cases}$

$$\tau^{A/B} = \oplus_{j=1}^{K} \sigma_j$$

**if**$\tau^A = \tau^B$
Counter ← Counter + 1
// Training the perceptrons
Learning$\left(B_{buf}, \sigma, s\right)$
**else**
Counter ← 0

**Continue**
**While** # iteration >boot_N and Synchronized!=1 **do**
X=feedback_input(X)
$\pi$=rand ($\pi$)
**for** j← 1 **to** K **do**
$\sigma_j = \theta\left(\sum_{i=1}^{N} s[\pi_{ij}] \oplus x_{ij}\right)$// Where $\theta(h) =$
$\begin{cases} 1, & h > N/2 \\ 0, & h \leq N/2 \end{cases}$

$$\tau^{A/B} = \oplus_{j=1}^{K} \sigma_j$$

**if**$\tau^A = \tau^B$
Counter ← Counter + 1
// Training the perceptrons
Learning$\left(B_{buf}, \sigma, s\right)$
**else**
Counter ← 0
**Continue**
**if** Counter > threshold
Synchronized ← 1
**else**
**Continue**

**B. Algorithm 2 :Feedback_input(X)**
// Making a part of input private
// Input:
An input array X, hidden neuron $\sigma$ and state vector s from iteration - 1.

//Output:
An output array X in which few bits are private.

j← 1

$$p_j = \oplus_{i=1}^{N} s[\pi_{ij}]$$
$$X[j] = p_j$$
$$X[N] = \sigma_j$$

j← 2

$$p_j = \oplus_{i=N+1}^{2N} s[\pi_{ij}]$$
$$X[N+1] = p_j$$
$$X[2N] = \sigma_j$$

**return** X

**C. Algorithm 3: learning**$\left(\boldsymbol{B_{buf}}, \boldsymbol{\sigma}, \boldsymbol{s}\right)$
//Training the perceptron with respect to $\tau$ exchanges.

//input :
PPM input X, state vector s, hidden neuron $\sigma$
//Output :
state vector s
**if** g!=G

**if**$\tau^A = \tau^B$

$$B_{buf}\ [g++] = \sigma^{A/B}[0]$$

**else**

**Continue**
**else**
**for** g=0 **to** G **do**

s[g]=$B_{buf}[g]$

**return** s

As the synchronization goes on, there is an overlap of weights between the participants as represented in *Figure 6*, which makes the PPM to get more attractive forces i.e., the value at the hidden units are identical and parity of the weights chosen based on permutation matrix are also equal between the machines.

## 4. Security analysis

Here, we discuss the effect of feedback on the security of neural network. The most successful attack on the PPM based key exchange protocol is the probabilistic attack [3]. The eavesdropper E uses similar PPM architecture with parameter G, K and N and it is assumed that eavesdropper also knows the synchronization algorithm, such that eavesdropper applies a feedback to $PPM^E$ network using output of hidden neuron $\sigma_j^E$ and parity of weights $p_j^E$ of each perceptron.

As per the proposed algorithm, after some synchronization steps attacker applies the feedback

method. Since the probabilistic attack uses an ensemble of M state vector S, $PPM^E$ computes the output of the hidden units $\sigma_j^E$ and parity of weights $p_j^E$ of each perceptrons of all the machines and consider the maximum occurred bit for feedback. This is how the eavesdropper tries to guess input vector X of the participants. The following algorithms explain the probabilistic attack against the PPM with feedback input.

**A. Algorithm 4: Attacker PPM with feedback input**

// Probabilistic attack against the PPM with feedback input.
// Input:
Private: Random state vector $s^E$, hidden neuron $\sigma^E$ .
Public : Random input $X^E$, permutation matrix $\pi$ , output $\tau^E$.
// Output:
Synchronized state vector $s^E$.
//Begin:
While # iterations < boot N do _
X=rand(X)
$\pi$=rand ($\pi$)
Probabilistic Attack()
While # iterations > boot N do _

X=Feedback input(X)

$\pi$=rand ($\pi$)

Probabilistic Attack()

**B. Algorithm 5: Probabilistic Attack()**

// Input:
Private: Random state vector $s^E$, hidden neuron $\sigma^E$.
Public : Random input $X^E$, permutation matrix $\pi$, output $\tau^E$.
// Output:
Synchronized state vector $s^E$ Begin:
Initialize M state vectors $s^E$ of size G.
Initialize probabilistic state vector $P^E$ of size G. Each elements $p_i$ are set to $1/2$.

**for** h←1 **to** M **do**
**for** j←1 **to** K **do**

$$\sigma_j^E = \theta\left(\sum_{i=1}^{N} s^E[h][\pi_{ij}] \oplus x_{ij}\right) \text{ // Where}$$
$$\theta(h) = \begin{cases} 1, & h > N/2 \\ 0, & h \le N/2 \end{cases}$$
$$\tau^E[h] = \oplus_{j=1}^{K} \sigma_j^E[h]$$

**if** $\tau^A = \tau^B$
**for** i←1 **to** K*N **do**

$$X\_save[g][i] = X[i]$$

**for** j←1 **to** G **do**

$$\pi\_save[g][j] = \pi[j]$$
$$g++$$

$stateE\_cnt = 0$
**for** h←1 **to** M **do**
**if** $\tau^E[h] = \tau^A$
**for** j←1 **to** K*N **do**
// Save state vector as valid state vector

$$stateE\_cnt++$$

restart = 0
**for** h←1 **to** M **do**
**if** $\tau^E[h] != \tau^A$
// Sample the state vector of particular attacker using Bernoulli trial

$$\sigma_j^E = \theta\left(\sum_{i=1}^{N} s^E[h][\pi_{ij}] \oplus x_{ij}\right) \text{// Where}$$
$$\theta(h) = \begin{cases} 1, & h > N/2 \\ 0, & h \le N/2 \end{cases}$$
$$\tau^E[h] = \oplus_{j=1}^{K} \sigma_j^E[h]$$

**if** $\tau^E[h] = \tau^A$

// Save state vector as valid state vector

$$stateE\_cnt++$$

**if** restart > limit

restart =0

// The elements $p_i$ of $P^E$, which is closest to collapse are reset to $1/2$

**for** i←1 **to** G **do**

$$q_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j} p^E_{\pi_{ij}} + \left(1 - x_{i,j}\right)\left(1 - p^E_{\pi_{ij}}\right)$$
$$p^{E+}[i] = \sum_{n=0}^{N/2} \binom{N}{n} q_j^n \left(1 - q_j\right)^{N-n}$$

// change the state vectors based on $p^E$

**for** j←1 **to** G **do**
**if** $p^{E+}[i] > 1/2$

$s^E[i] = 0$

**else**

$s^E[i] = 1$

**return** $s^E$

**C. Algorithm 6: Feedback input(X)**

// Making a part of input private.

//input:

An input array X, hidden neuron $\sigma$ and state vector $s^E$
from iteration - 1.
//output :
An output array X in which few bits are private.
j←1
**for** h←1 **to** M **do**

$$p_j^E = \oplus_{i=1}^{N} s^E[h][\pi_{ij}]$$

$$\sigma_j^E = \theta\left(\sum_{i=1}^{N} s^E[h][\pi_{ij}] \oplus x_{ij}\right)$$

X[j]=max_occurrence($p_j^E[h]$)
X[N]=max_occurrence($\sigma_j^E[h]$)
j←2
**for** h←1 **to** M **do**

$$p_j^E = \oplus_{i=N+1}^{2N} s^E[h][\pi_{ij}]$$

$$\sigma_j^E = \theta\left(\sum_{i=N+1}^{2N} s^E[h][\pi_{ij}] \oplus x_{ij}\right)$$

X[N+1]=max_occurrence($p_j^E[h]$)
X[2N]=max_occurrence($\sigma_j^E[h]$)
**return** X

The following *Figure 6*, illustrates the percentage of overlap of probabilistic attack as a function of outer rounds for the parameter G=128, K=2 and N=16. In this attacker percentage of overlap of attacker lies between 60% and 85%. In this approach attacker cannot be able to guess the state vector of one of the participants.

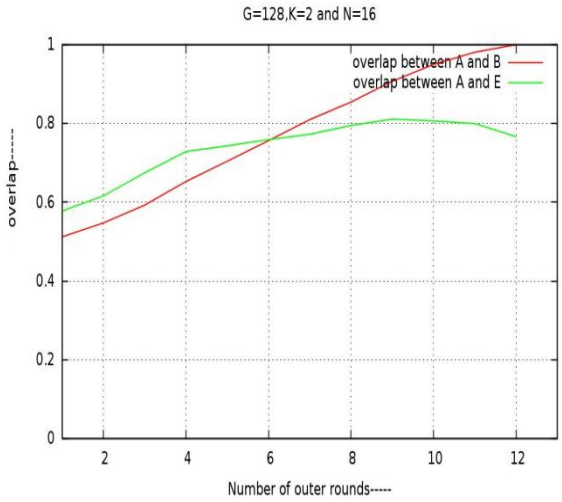


**Figure 6** Overlap between an attacker and participants as a function of outer rounds for the parameter G=128, K=2 and N=16 averaging the similar outer rounds

In the *Figure 7*, the performance of the proposed feedback method can be analyzed in terms of the probability of success $P_s$ of the attack as a function of N for the parameters G=128 and K=2, results are averaged over 1000 runs. The probability of success of the attacker lies in between 0.01 to 0.6. In this approach as number of inputs N is increased, probability of success decreased.

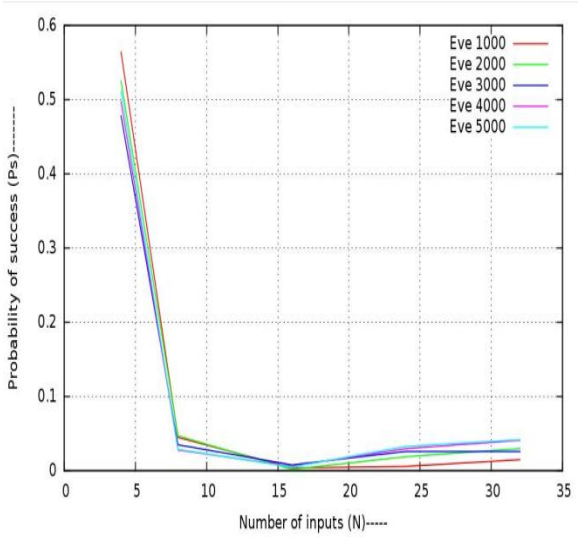


**Figure 7** Probability of success of an attacker as a function of N for the parameter G=128 and K=2 in probabilistic attack. Results are averaged over 1000 runs

## 5.Conclusion

The permutation parity machine, a binary variant of tree parity machine is secure against majority flipping attack and geometric attack which are known successful attacks in TPM. But, the advent of probabilistic attack makes PPM insecure for key exchange. The proposed technique reduces the probability of success in case of probabilistic attack in PPM.

### Acknowledgment

This work is developed as part of the CySeRo component of the ARiEES project at CSIR-4PI, Bengaluru and supported by Student Programme for Advancement in Research Knowledge (SPARK). Authors are grateful to the Head, CSIR-4PI for giving us the opportunity to carry out this work.

### Conflicts of interest

The authors have no conflicts of interest to declare.

## References


[1] Reyes OM, Kopitzke I, Zimmermann KH. Permutation parity machines for neural synchronization. Journal of Physics A: Mathematical and Theoretical. 2009; 42(19):195002.

[2] Reyes OM, Zimmermann KH. Permutation parity machines for neural cryptography. Physical Review E. 2010; 81(6):066117.

[3] Seoane LF, Ruttor A. Successful attack on permutation-parity-machine-based neural cryptography. Physical Review E. 2012; 85(2):025101.

[4] Iglesias LF. Probabilistic attack on neural cryptography. 2012.

[5] Ruttor A, Kinzel W, Shacham L, Kanter I. Neural cryptography with feedback. Physical Review E. 2004; 69(4):046110.

[6] Kinzel W, Kanter I. Neural cryptography. arXiv preprint cond-mat/0208453. 2002.

[7] El-Zoghabi A, Yassin AH, Hussien HH. Survey report on cryptography based on neural network. International Journal of Emerging Technology and Advanced Engineering. 2013; 3(12):456-62.

[8] Ruttor A. Neural synchronization and cryptography. arXiv preprint arXiv:0711.2411. 2007.

[9] Agarwal N, Agarwal P. Use of artificial neural network in the field of security. MIT International Journal of Computer Science & Information Technology. 2013; 3(1):42-4.


This paper is selected from proceedings of National Workshop on Cryptology-NWC 2016 organized at JNN College of Engineering Shimoga, Karnataka, India during 11-13, August 2016.