Generative adversarial network guided mutual learning based synchronization of cluster of neural networks

Arindam Sarkar¹

Received: 8 August 2020 / Accepted: 11 February 2021 / Published online: 12 March 2021
© The Author(s) 2021

Abstract

Neural synchronization is a technique for establishing the cryptographic key exchange protocol over a public channel. Two neural networks receive common inputs and exchange their outputs. In some steps, it leads to full synchronization by setting the discrete weights according to the specific rule of learning. This synchronized weight is used as a common secret session key. But there are seldom research is done to investigate the synchronization of a cluster of neural networks. In this paper, a Generative Adversarial Network (GAN)-based synchronization of a cluster of neural networks with three hidden layers is proposed for the development of the public-key exchange protocol. This paper highlights a variety of interesting improvements to traditional GAN architecture. Here GAN is used for Pseudo-Random Number Generators (PRNG) for neural synchronization. Each neural network is considered as a node of a binary tree framework. When both i -th and j -th nodes of the binary tree are synchronized then one of these two nodes is elected as a leader. Now, this leader node will synchronize with the leader of the other branch. After completion of this process synchronized weight becomes the session key for the whole cluster. This proposed technique has several advantages like (1) There is no need to synchronize one neural network to every other in the cluster instead of that entire cluster can be able to share the same secret key by synchronizing between the elected leader nodes with only logarithmic synchronization steps. (2) This proposed technology provides GAN-based PRNG which is very sensitive to the initial seed value. (3) Three hidden layers leads to the complex internal architecture of the Tree Parity Machine (TPM). So, it will be difficult for the attacker to guess the internal architecture. (4) An increase in the weight range of the neural network increases the complexity of a successful attack exponentially but the effort to build the neural key decreases over the polynomial time. (5) The proposed technique also offers synchronization and authentication steps in parallel. It is difficult for the attacker to distinguish between synchronization and authentication steps. This proposed technique has been passed through different parametric tests. Simulations of the process show effectiveness in terms of cited results in the paper.

Keywords Neural synchronization · Tree parity machine (TPM) · Session key · Three layer tree parity machine (TLTPM) · Neural network · Mutual learning · Generative adversarial network (GAN)

Introduction

Cryptography is an art of converting the plaintext into ciphertext using a secret key and vice-versa. It aims to secure information from eavesdroppers, interceptors, rivals, intruders, enemies, and assailants Bauer [5]. The prime concentration is on the frameworks for ensuring the confi-

dentiality of information and techniques for the exchange of authentication keys and protocols between communicators Lindell and Katz [29]. The best-known approach is the use of the symmetrical and asymmetrical methods of encryption and decryption.

Modern cryptography uses algebraic number theory Diffie and Hellman [11], Steiner et al. [49], Balasubramaniam and Muthukumar [3], Eftekhari [16] to endorse a variety of cryptography algorithms. In authentication and encryption, the RSA cryptosystem and elliptic curve-based schemes are the best known public-key cryptography Zhou and Tang [56]. The RSA algorithm protection depends, however, on its prime factorization Meneses et al. [32]. If the size of the

✉ Arindam Sarkar
arindam.vb@gmail.com

¹ Department of Computer Science and Electronics,
Ramakrishna Mission Vidyamandira, Belur Math, Howrah,
West Bengal 711202, India

prime numbers is increased then the security of the encryption algorithm also increased; however, which also increases the calculation complexity and costs.

Diffie and Hellman [11]’s public-key exchange algorithm is a key distribution algorithm. It enables two communicating devices to agree upon a common encryption key through an unstable medium by sharing a key between them. In several fields, this exchange process, and the secret key are used, such as authentication of identity, data encryption, the security of data privacy. This scheme suffers from a MITM attack.

It is, therefore, necessary to search for innovative methods of secured and low costs protocol for the generation/exchange of cryptographic keys. This is a big challenge as attacker E may not be able to deduce the key, although he may partake the ability to follow the algorithm framework. The process of synchronization of the neural network provides an ability to address immense exchange problems Chen et al. [7], Liu et al. [31], Chen et al. [6], Wang et al. [52], Wang et al. [53], Xiao et al. [54], Zhang and Cao [55], Wang et al. [51], Dong et al. [15]. Neural cryptography has recently been recognized as capable of achieving this aim through neural synchronization Rosen-Zvi et al. [40] through in-depth integration and analysis of Artificial Neural Network (ANN) Lakshmanan et al. [28], Ni and Paul [35].

A special Artificial Neural Network (ANN) framework called the Tree Parity Machine (TPM) is used in this proposed technique. On both the sender and the receiver end, two TPM networks with similar configurations are used. These two networks are synchronized by generating random input vector and exchanging the outputs of such networks and preserving the secret of the synaptic weight. Two users A and B can generate a cryptographic key that is difficult for the attacker to infer, even though the attacker is aware of the algorithm structure and communication channel. In this paper, a Generative Adversarial Network (GAN)-based PRNG is proposed for generating the input vector to the neural synchronization process. This research is motivated by the work of Abadi and Andersen [1] in the learning of encoding methods by the neural network, which suggests that even a neural network can represent a good PRNG function. The intention is also derived from security requirements: there are some potentially useful characteristics of a hypothetical neural-network-based PRNG. This requires the ability, through more training, to make ad hoc alterations to the generator, which can constitute the backbone of strategies for dealing with the type of non-statistical threats presented by Kelsey et al. [25].

The rest of this paper is organized accordingly. Section 2 deals with related works. Section 3 deals with the proposed methodology. Sections 4 and 5 deal with security analysis and the results respectively. Conclusions are given in Sect. 6 and references are given at the end.

Related works

Several efforts are being made with neural networks to produce PRNG sequences by Desai et al. [10], Desai et al. [9], Tirdad and Sadeghian [50], Jeong et al. [21]. Tirdad and Sadeghian [50] and Jeong et al. [21], have described the most effective methods. The previous was using Hopfield neural networks to avoid convergence and promote dynamic behavior, whereas the latter used a random data sample-trained LSTM to acquire indices in the pi digits. In statistical randomness tests, these two papers reported a great result. However, no method sought to train a PRNG neural network end-to-end, rather than using the networks as modules of more complicated algorithms. RosenZvi et al. [40] and Kanter et al. [23] successfully created equal states of their internal synaptic weight when two ANNs have been trained using a particular learning rule. Kinzel and Kanter [26] and Ruttort et al. [42] described that a chance of an attack is declining if the network’s weight range get increases and the assailant’s calculation cost increases, as its effort increases exponentially, as users’ effort becomes polynomial. Sarkar and Mandal [44], Sarkar et al. [48], Sarkar et al. [46], Sarkar and Mandal [45], Sarkar et al. [47] proposed schemes which enhanced the security of the protocol by enhancing the synaptic depths of TPM and henceforth counteracting the attacks of the brute power of the attacker. It is found that the amount of security provided by the TPM synchronization also can be improved by introducing a large set of neurons and entries of each neuron in the hidden layers. Allam et al. [2] described an authentication algorithm using previously shared secrets. As a result, the algorithm attains a very high degree of defense without increasing synchronization time. Ruttort [41] described that lower values of the hidden unit have negative safety implications. Klimov et al. [27], compute whether or not two networks synchronize their weights. In connection with performance measurements of TPM network performance measurements, Dolecki and Kozera [12] proposed a frequency analysis technique that permits the assessment before completion of the synchronization rate of two TPM networks with a determined value that is not related to their differences in synaptic weights. Santhanalakshmi et al. [43] and Dolecki and Kozera [13] evaluate the efficiency of coordinated usage of genetic algorithm and the Gaussian distribution respectively. As a result, the random weight’s replacement with optimum weights decreases the time of synchronization. Pu et al. [39] carry out an algorithm, which combines “true random sequences” with a TPM network that shows more complex dynamic behaviors which enhances encryption efficiency and attack resistance. According to the analysis, a safe, uniform and distribution for the synaptic weight values of the TPM networks are produced. Further, the Poisson’s distribution changes in each simulation run with the results as steps are move ahead towards synchronization. In the light of rules

leading to the creation of TPM synchronization, Mu and Liao [33] and Mu et al. [34] describes the heuristic of minimum hamming distances. To test the security level of the final TPM network structure, the proposed heuristic rule was used. Concerning improvements to initial TPM network infrastructure, Gomez et al. [19] observed that the synchronization period is reduced from 1.25 ms to less than 0.7 ms with an initial assignment of the weights between 15 to 20%. The number of steps is also decreased from 220 to less than 100. Niemiec [36] is proposing a new concept for the main quantity reconciliation process using TPM networks. Dong and Huang [14] proposed a complex value-based neural network for neural cryptography. Here all the inputs and the outputs are complex value. But, this technique takes a significant amount of time to complete the synchronization process.

From this survey, it has been observed that all works in progress in neural cryptography are based on the mutual synchronization of two neural networks. To synchronize with n numbers of neural networks needs n^2 number of synchronization. Also, it is rare to study the generation of good random input vector through robust pseudo-random number generator in the neural synchronization process. Existing TPM does not perform authentication steps in parallel with synchronization. The proposed technique performs cluster synchronization in logarithmic time instead of generating separate keys for individual communication in quadratic time. By implementing a deep learning technique such as GAN Goodfellow et al. [20] to design a PRNG that produces PRNG sequences explicitly, this paper develops the specific task. This paper, using the NIST Bassham et al. [4] test suite, introduces two simple structures and tests their power as PRNGs. A framework that is simple, conceptually efficient, and reliable is the general result of these modifications. Nearly 95 % of NIST tests can be consistently passed by the trained generator, indicating that the adversarial procedure is extremely effective in training the network to act as a PRNG. Findings are comparable to those of Tirdad and Sadeghian [50], and Jeong et al. [21], outperforming a range of regular PRNGs. This proposed cluster synchronization has several advantages that are as follows:

- (1) This proposed technology provides GAN-based robust random input vectors for the synchronization process.
- (2) To enhance the security a complex internal architecture of TPM is used.
- (3) An increase in the weight range of the neural network increases the complexity of a successful attack exponentially but the effort to build the neural key decreases over the polynomial time.
- (4) The proposed technique also offers synchronization and authentication steps in parallel.

Table 1 Symbol Description of GAN-based PRNG for Neural Synchronization

Symbol	Description
G	Generator
D	Discriminator
P	Predictor
U	The set of range of input and output values
T	Tuples set
n	The dimension of network output
O_t	Input vector
r	Eight bit long vector
r_{split}	The output vector of the generator
ϕ	PRNG function

Different parametric tests are performed to check the robustness and randomness of the GAN generated input vector. Python is used for the implementation of the technique and statistical analysis is done using R.

Proposed methodology

A key aspect of many security algorithms is Pseudo-Random Number Generators (PRNG). One of the focus is to decide how a GAN system can learn how to produce randomized sequences of numbers, and if such a system might be used in a security context as a PRNG. By suggesting the use of GAN to train an ANN to act as a PRNG, this paper presents a new approach to its implementation. Consequently, the generator learns to generate values that cannot be predicted by the attacker.

This paper demonstrates that even a tiny feed-forward completely linked ANN can be efficiently trained by a GAN to generating PRNG sequences with strong statistical characteristics. In this paper, a GAN-based common input vectors generation for neural synchronization is proposed. The proposed technique considers the range of input and output values as a set U . Table 1 lists the variables used in this GAN-based PRNG for neural synchronization.

The members of set U are all unsigned integer of 16 bits. PRNG can be represented as a following Eq. 1.

$$\phi(s) : U \rightarrow U^n \quad (1)$$

Here, s is the seed value. The seed value is random which helps to generate a complete random common input vector that in turn satisfies the randomness in the purview of the NIST test suite for randomness. Here n can be considered a very large number. The model also can be characterized by individual outputs using Eq. 2.

$$\phi^\nabla(s, C_i) : T \rightarrow U \quad (2)$$

Here C_i is the present internal state, and T represents the all tuples set (s, C_i) . To approximate, $\phi(s)$ a function $G(s)$ is used. Consider an input O_t that leads to the following Eq. 3, which approximates $\phi^\nabla(s, C_i)$, with the dimension network output.

$$G^\nabla(s, O_t) : U^{t+1} \rightarrow U^n \quad (3)$$

For any fixed s value the model $G(s)$ is presented by concatenating generator's output sequences $\forall t \ G^\nabla(s, O_t)$.

$$|G(s)| \in \theta(n^t) \quad (4)$$

It follows the Eq. 4, for the length of the full sequence $G(s)$. The model will reduce the adversary's probability of accurately predicting future outputs from previous outputs. This model can be subdivided into discriminative and predictive model.

Each n -bit produced by the generator in the predictive approach is divided in such a way so that the $(n - 1)$ bits act as an input for the predictor, and the label is designated by the n -th bit. Following Fig. 1 describes the discriminative approach.

Figure 2 describes the predictive approach.

The generator consists of a fully connected feed-forward neural network, which is shown in the Eq. 5.

$$G^\nabla(s, O_1) : U^2 \rightarrow U^8 \quad (5)$$

The input vector contains a s value and a scalar O_1 . In this model, four hidden layers of thirty units and one output layer of eight units is considered shown in Fig. 3. The Leaky ReLU activation function gets used by the input layer and the hidden layer. Mod activation function is applied to the output layer, mapping values into the desired range while avoiding other pitfalls of *sigmoid* and *tan h*.

A Convolution Neural Network (CNN) is used as a discriminator using the following Eq. 6.

$$D(r) : U^8 \rightarrow [0, 1] \quad (6)$$

where, r is an 8-long vector. The generator generates this r . The discriminator generates a scalar $p(true)$ inside $[0, 1]$, which is likely to belong to either class. The discriminator is made up of 4 stacked CNN layers each having four filters, a kernel size of two, and stride one, followed by a layer of max-pooling and 2 fully connected layers each having four and one respectively. The stack of convolution layers allows the network to identify complex input patterns. The least-squares loss is considered for discriminative purpose. Figure 4 shows the structure of the discriminator.

A Convolution Neural Network (CNN) is used as a predictor using the following Eq. 7.

$$P(r_{split}) : U^7 \rightarrow U \quad (7)$$

where r_{split} is the output vector of the generator with the last variable is discarded. The label is designated by the n -th bit which is served as a predictor's input. Both the discriminator and predictor have the same design. But their input size and meaning of the result are different from each other. An absolute difference loss is used for the predictive purpose. NIST statistical test for randomness is performed on a variety of output generated for a particular seed for both before and after training. Next, the initialization of the predefined evaluation data set D is performed. It consists of input vector $I_i \in U^2$ in the form $[s, O_{1i}]$, so that the random $s \in I_i$ is fixed to the same random value $\forall i$ and all experiments. For I_i the offset O_{1i} starts at zero for I_0 and increases sequentially for the following vectors. Consider, arbitrarily $s = 20$ then $D = [[20, 0], [20, 1], [20, 2], \dots]$

A generator is used to produce vectors for all vectors in D before training. Here, the only floating-point output is considered. Floating-point output is rounded down to the nearest integer. When the outputs are distributed uniformly across a range $[a, b]$ where $a, b \in \mathbb{R}^+$, then they distributed uniformly across the range $[a, b]$ where $a, b \in \mathbb{Z}^+$.

Generator performance is being assessed after and before training using the NIST test. The procedure is repeated for at least twenty times for discriminative and predictive approaches.

Here, a complete binary tree framework is used to synchronize the cluster of TLTPM. Each TLTPM is considered as a node of a tree framework shown in Fig. 5. In a complete binary tree, a node with no child is called a leaf node. Consider $j = 1$ at initial round (j is the round number) and *leaves* = 8. Then the binary tree gets divided into $\frac{\text{no. of leaves}}{2^j}$ subtrees each with 2^j leaves i.e. four subtrees with two leaves. Siblings are involved in mutual learning at height four.

In round 1 siblings become synchronized using the mutual learning step shown in Fig. 6. From each subtree, a node is nominated as a leader among the nodes having the same parents to perform the next round of operation.

In round 2 when $j = 2$ and *leaves* = 8, the tree gets divided into two subtrees with four leaves at height 3 shown in Fig. 7. In this way next rounds are performed until the root node at height 1 is synchronized.

On the completion of the synchronization process, all the nodes in the cluster become synchronized based on a common cluster session key. Table 2 lists the parameters used in this article.

This special neural network TLTPM is composed of M no. of input neurons for each H no. of hidden neurons. TLTPM

Fig. 1 Discriminative approach

Fig. 2 Predictive approach

Fig. 3 Structure of the generator: Fully Connected Layers and ReLU and mod activation function

Fig. 4 Structure of discriminator

Fig. 5 Group TLTPM synchronization at Initial State

Fig. 6 Group TLTPM synchronization at first round

Fig. 7 Group TLTPM synchronization at second round

has an only neuron in its output. TLTPM works with binary input, $\alpha_{u,v} \in \{-1, +1\}$. The mapping between input and output is described by the discrete weight value between $-\kappa$ and $+\kappa$, $\beta_{u,v} \in \{-\kappa, -\kappa + 1, \dots, +\kappa\}$.

In TLTPM u -th hidden unit is described by the index $u = 1, \dots, H$ and that of $v = 1, \dots, M$ denotes input neuron corresponding to the u -th hidden neuron of the TLTPM. Consider there are $H1$, $H2$, $H3$ numbers of hidden units in the first, second, and third hidden layers respectively. Each hidden unit of the first layer calculates its output by performing the weighted sum over the present state of inputs on that particular hidden unit. Each hidden unit of the second layer calculates its output by performing the weighted sum over the hidden units of the first layer on that particular hidden unit. Similarly, each hidden unit of the third layer calculates its output by performing the weighted

sum over the hidden units of second layer on that particular hidden unit. The calculation for the first hidden layer is given by Eq. 8.

$$h_u = \frac{1}{\sqrt{M}} \alpha_u \cdot \beta_u = \frac{1}{\sqrt{M}} \sum_{v=1}^M \alpha_{u,v} \beta_{u,v} \quad (8)$$

$\text{signum}(h_u)$ define the output γ_u of the u -th hidden unit. (in Eq. 9),

$$\gamma_u = \text{signum}(h_u) \quad (9)$$

If $h_u = 0$ then γ_u is set to -1 to make the output in binary form. If $h_u > 0$ then γ_u is mapped to $+1$, which represents that the hidden unit is active. If $\gamma_u = -1$ then it denotes that

Table 2 Symbol Description of TLTPM

Symbol	Description
M	Total number of input neurons to each hidden neuron
H	Total number of hidden neurons
$H1$	Total number of hidden neurons in first hidden layer
$H2$	Total number of hidden neurons in second hidden layer
$H3$	Total number of hidden neurons in third hidden layer
κ	Weight range
α_u	The input vector of the u -th hidden unit
β_u	The weight vector of the u -th hidden unit
$\alpha_{u,v}$	v -th element of α_u
$\beta_{u,v}$	v -th element of β_u
γ_u	u -th hidden unit's output
ζ	Final output of the TLTPM
h_u	u -th hidden unit's local field
S	Weight's entropy
$pb_{a,b}^u$	The likelihood distribution of the weight values in u -th hidden neuron of a 's and b 's TLTPM
Q_u	The standard order parameter
ρ_u	The normalized overlap

the hidden unit is inactive (in Eq. 10).

$$\text{signum}(h_u) = \begin{cases} -1 & \text{if } h_u \leq 0 \\ +1 & \text{if } h_u > 0 \end{cases} \quad (10)$$

The product of the hidden neurons of the third layer denotes the ultimate result of TLTPM. This is represented by ζ is (in Eq. 11),

$$\zeta = \prod_{u=1}^{H3} \gamma_u \quad (11)$$

The value of ζ is mapped in the following way (in Eq. 12),

$$\zeta = \begin{cases} -1 & \text{if } \gamma_u = -1, \text{ is odd} \\ +1 & \text{if } \gamma_u = -1, \text{ is even} \end{cases} \quad (12)$$

$\zeta = \gamma_1$, if only one hidden unit ($H = 1$) is there. ζ value can be the same for 2^{H-1} different $(\gamma_1, \gamma_2, \dots, \gamma_H)$ representations.

If the output of two parties disagrees, $\zeta^A \neq \zeta^B$, then no update is allowed on the weights. Otherwise, follow the following rules:

s TPM be trained from each other using Hebbian learning rule inzel and Kanter [26] (in Eq. 13).

$$\beta_{u,v}^+ = fn(\beta_{u,v} + \alpha_{u,v} \zeta \Theta(\gamma_u \zeta) \Theta(\zeta^A \zeta^B)) \quad (13)$$

In the Anti-Hebbian learning rule, both TPM is learned with the reverse of their output Kinzel and Kanter [26] (in Eq. 14).

$$\beta_{u,v}^+ = fn(\beta_{u,v} - \alpha_{u,v} \zeta \Theta(\gamma_u \zeta) \Theta(\zeta^A \zeta^B)) \quad (14)$$

If the set value of the output is not imperative for tuning given that it is similar for all participating TPM then random-walk learning rule Kinzel and Kanter [26], is used (in Eq. 15).

$$\beta_{u,v}^+ = fn(\beta_{u,v} + \alpha_{u,v} \Theta(\gamma_u \zeta) \Theta(\zeta^A \zeta^B)) \quad (15)$$

If $X = Y$ then $\Theta(X, Y) = 1$ Otherwise, If $X \neq Y$ then $\Theta(X, Y) = 0$. Only weights are updated which are in hidden units with $\gamma_u = \zeta$. $fn(\beta)$ is used for each learning rule (in Eq. 16).

$$fn(\beta) = \begin{cases} \text{signum}(\beta)\kappa & \text{for } |\beta| > \kappa \\ \beta & \text{otherwise} \end{cases} \quad (16)$$

The likelihood distribution of the weight values in u -th hidden neuron of two TLTPM is represented by $(2\kappa + 1)$ (in Eq. 17).

$$pb_{a,b}^u = P(\beta_{u,v}^A = a \wedge \beta_{u,v}^B = b) \quad (17)$$

The standard order parameters [18] can be calculated as functions of $pb_{a,b}^u$ shown in Eq. 18, 19, and 20.

$$\begin{aligned} Q_u^A &= \frac{1}{M} \beta_U^A \beta_U^A \\ &= \sum_{a=-\kappa}^{\kappa} \sum_{b=-\kappa}^{\kappa} a^2 pb_{a,b}^u \end{aligned} \quad (18)$$

$$\begin{aligned} Q_u^B &= \frac{1}{M} \beta_U^B \beta_U^B \\ &= \sum_{a=-\kappa}^{\kappa} \sum_{b=-\kappa}^{\kappa} b^2 pb_{a,b}^u \end{aligned} \quad (19)$$

$$\begin{aligned} Q_u^{AB} &= \frac{1}{M} \beta_U^A \beta_U^B \\ &= \sum_{a=-\kappa}^{\kappa} \sum_{b=-\kappa}^{\kappa} ab pb_{a,b}^u \end{aligned} \quad (20)$$

Tuning is represented by the normalized overlap [18] given in Eq. 21.

$$\rho_u^{AB} = \frac{\beta_u^A}{\sqrt{\beta_u^A \cdot \beta_u^A}} \frac{\beta_u^B}{\sqrt{\beta_u^B \cdot \beta_u^B}} = \frac{R_u^{AB}}{\sqrt{Q_u^A \cdot Q_u^B}} \times \quad (21)$$

Calculate the entropy [8] using Eq. 22.

$$S_u^{AB} = -M \sum_{a=-\kappa}^{\kappa} \sum_{b=-\kappa}^{\kappa} pb_{a,b}^u \ln pb_{a,b}^u \quad (22)$$

Table 3 Comparison of p_Value between proposed TLTPM and the existing CVTPM

NIST Test	p_Value of Proposed TLTPM (Average of 5 iterations)	p_Value of Proposed TLTPM (Average of 10 iterations)	p_Value of CVTPM (Average of 10 iterations) Dong and Huang [14]
Frequency	0.559752	0.696413	0.512374
Frequency within a block	0.579271	0.613725	0.538721
Runs	0.516802	0.527912	0.489451
Longest run of ones in a block	0.089541	0.098364	0.042081
Binary matrix rank	0.679758	0.736108	0.417393
Discrete fourier transform	0.404732	0.595301	0.392762
Non-overlapping template matching	0.460642	0.519630	0.431759
Overlapping (Periodic) template matching	0.212969	0.369312	0.170426
Maurer's "Universal statistical"	0.786158	0.793612	0.689478
Linear complexity	0.655287	0.698362	0.594917
Serial	0.520863	0.594207	0.478902
Approximate entropy	0.285963	0.364871	0.204175
Cumulative sums	0.693105	0.716983	0.587290
Random excursions	0.349342	0.459321	0.299875
Random excursions Variants	0.223387	0.368273	0.180542

The weight's entropy in a single hidden neuron is represented by Eqs. 23 and 24.

$$S_u^A = -M \sum_{a=-\kappa}^{\kappa} \left(\sum_{b=-\kappa}^{\kappa} p b_{a,b}^u \right) \quad (23)$$

$$\ln \left(\sum_{b=-\kappa}^{\kappa} p b_{a,b}^u \right) \\ S_u^B = -M \sum_{b=-\kappa}^{\kappa} \left(\sum_{a=-\kappa}^{\kappa} p b_{a,b}^u \right) \quad (24) \\ \ln \left(\sum_{a=-\kappa}^{\kappa} p b_{a,b}^u \right)$$

Using Eqs. 22, 23, and 24 the common information Cover and Thomas [8] of A's and B's represented using Eq. 25.

$$In^{AB} = \sum_{u=1}^H (S_u^A + S_u^B - S_u^{AB}) \quad (25)$$

The likelihood to observe $\gamma_u \alpha_{u,v} = +1$ or $\gamma_u \alpha_{u,v} = -1$ are not equal, but depend on the related weight $\beta_{u,v}$ (in Eq. 26).

$$P(\gamma_u \alpha_{u,v} = 1) = \frac{1}{2} \left(1 + \operatorname{erf} \left(\frac{\beta_{u,v}}{\sqrt{M Q_u - \beta_{u,v}^2}} \right) \right) \quad (26)$$

$\gamma_u \alpha_{u,v} = \operatorname{signum}(\beta_{u,v})$ occurs more frequently than $\gamma_u \alpha_{u,v} = -\operatorname{signum}(\beta_{u,v})$, the stationary likelihood distribution of the weights for $t \rightarrow \infty$, is computed using Eq. 19

for the transition likelihood Rutter et al. [42]. This is represented using Eq. 27.

$$P(\beta_{u,v} = \beta) = p_0 \prod_{i=1}^{|\beta|} \frac{1 + \operatorname{erf} \left(\frac{i-1}{\sqrt{M Q_u - (i-1)^2}} \right)}{1 - \operatorname{erf} \left(\frac{i}{\sqrt{M Q_u - i^2}} \right)} \quad (27)$$

Here the normalization constant p_0 is given by Eq. 28.

$$p_0 = \left(\sum_{\beta=-\kappa}^{\kappa} \prod_{i=1}^{|\beta|} \frac{1 + \operatorname{erf} \left(\frac{i-1}{\sqrt{M Q_u - (i-1)^2}} \right)}{1 - \operatorname{erf} \left(\frac{i}{\sqrt{M Q_u - i^2}} \right)} \right)^{-1} \quad (28)$$

For $M \rightarrow \infty$, the parameter of the error functions will be ruled out, so that the weights stay consistently dispersed shown in Eq. 29.

$$\sqrt{Q_u(t=0)} = \sqrt{\frac{\kappa(\kappa+1)}{3}} \quad (29)$$

Otherwise, if M is finite, the likelihood distribution represented using order parameter Q_u , shown in the following Eq. 30.

$$Q_u = \sum_{\beta=-\kappa}^{\kappa} \beta^2 P(\beta_{u,v} = \beta) \quad (30)$$

Intensifying it in terms of $M^{-1/2}$ results Ruttor et al. [42] in Eq. 31.

$$Q_u = \frac{\kappa(\kappa+1)}{3} + \frac{8\kappa^4 + 16\kappa^3 - 10\kappa^2 - 18\kappa + 9}{15\sqrt{3\pi\kappa(\kappa+1)}} \frac{1}{\sqrt{M}} + O\left(\frac{\kappa^4}{M}\right) \quad (31)$$

In the case of $1 \ll \kappa \ll \sqrt{M}$, the asymptotic performance of the order parameter is represented using Eq. 32.

$$Q_u \sim \frac{\kappa(\kappa+1)}{3} \left(1 + \frac{8}{5\sqrt{3\pi}} \frac{\kappa}{\sqrt{M}}\right) \quad (32)$$

First-order approximation [42] of Q_u is given by Eq. 33.

$$Q_u = \frac{\kappa(\kappa+1)}{3} - \frac{8\kappa^4 + 16\kappa^3 - 10\kappa^2 - 18\kappa + 9}{15\sqrt{3\pi\kappa(\kappa+1)}} \frac{1}{\sqrt{M}} + O\left(\frac{\kappa^4}{M}\right) \quad (33)$$

This systematically converges to (in Eq. 34).

$$Q_u \sim \frac{\kappa(\kappa+1)}{3} \left(1 - \frac{8}{5\sqrt{3\pi}} \frac{\kappa}{\sqrt{M}}\right) \quad (34)$$

If $\zeta^A = \zeta^B$, but $\gamma_u^A \neq \gamma_u^B$, the weight of one hidden neuron is changed. The weights execute an anisotropic diffusion in case of attractive steps (in Eq. 35).

$$pb_{a,b}^{u+} = \frac{1}{2} (pb_{a+1,b+1}^u + pb_{a-1,b-1}^u) \quad (35)$$

Repulsive steps, as an alternative, are equal to normal diffusion steps (in Eq. 36),

$$pb_{a,b}^{u+} = \frac{1}{4} (pb_{a+1,b}^u + pb_{a-1,b}^u + pb_{a,b+1}^u + pb_{a,b-1}^u) \quad (36)$$

on the same lattice. $\Delta\rho_{sattr}(\rho)$ and $\Delta\rho_{srepu}(\rho)$ are random variables. $\Delta\rho_{sattr}$ and $\Delta\rho_{srepu}$ are the step size for attractive and repulsive respectively shown in Eqs. 37 and 38.

$$\Delta\rho_{sattr} = \frac{3}{\kappa(\kappa+1)} \left(1 - \sum_{v=-\kappa}^{+\kappa} (2v+2) pb_{\kappa,v} + pb_{\kappa,\kappa}\right) \quad (37)$$

$$\Delta\rho_{srepu} = -\frac{3}{\kappa(\kappa+1)} \left(\sum_{v=-\kappa}^{+\kappa} \frac{v}{2} (pb_{\kappa,j} - pb_{-\kappa,v})\right) \quad (38)$$

At the initial state of the synchronization, it has its highest Klimov et al. [27] effect (in Eq. 39),

$$\Delta\rho_{sattr}(\rho=0) = \frac{12\kappa}{(\kappa+1)(2\kappa+1)^2} \sim \frac{3}{\kappa^2} \quad (39)$$

Weights are uncorrelated as shown in Eq. 40.

$$pb_{a,b}(\rho=0) = \frac{1}{(2\kappa+1)^2} \quad (40)$$

Highest consequence (in Eq. 41),

$$\Delta\rho_{srepu}(\rho=1) = \frac{3}{(\kappa+1)(2\kappa+1)} \sim -\frac{3}{2\kappa^2} \quad (41)$$

is achieved for complete harmonized weights (in Eq. 42).

$$pb_{a,b}(\rho=1) = \begin{cases} (2\kappa+1)^{-1} & \text{for } a=b \\ 0 & \text{for } a \neq b \end{cases} \quad (42)$$

The weights are updated if $\zeta^A = \zeta^B$. ϵ_u is the generalization error. In common overlap for H hidden neurons, $\epsilon_u = \epsilon$, the likelihood is represented using Eq. 43.

$$Pb_{id} = Pb(\zeta^A = \zeta^B) = \sum_{u=0}^{H/2} \left(\frac{H}{2u}\right) (1-\epsilon)^{H-2u} \epsilon^{2u} \quad (43)$$

For synchronization of TLTPM with $H > 1$, the likelihood of attractive as well as repulsive are represented using Eqs. 44 and 45.

$$Pb_{attr}^B = \frac{1}{2Pb_{id}} \sum_{u=0}^{\left(\frac{H-1}{2}\right)} \left(\frac{H-1}{2u}\right) (1-\epsilon)^{H-2u} \epsilon^{2u} \quad (44)$$

$$Pb_{repu}^B = \frac{1}{Pb_{id}} \sum_{u=1}^{H/2} \left(\frac{H-1}{2u-1}\right) (1-\epsilon)^{H-2u} \epsilon^{2u} \quad (45)$$

For $H = 3$, this leads to Klimov et al. [27] following Eqs. 46 and 47.

$$Pb_{attr}^B = \frac{1}{2} \frac{(1-\epsilon)^3 + (1-\epsilon)\epsilon^2}{(1-\epsilon)^3 + 3(1-\epsilon)\epsilon^2} \quad (46)$$

$$Pb_{repu}^B = \frac{2(1-\epsilon)\epsilon^2}{(1-\epsilon)^3 + 3(1-\epsilon)\epsilon^2} \quad (47)$$

Harmonization time $Tm_{r,s}$ for the two random walk beginning at position s and distance r . $Rf_{r,s}$ is the time of the first reflection Ruttor et al. [42]. $Tm_{r,s}$ is given by Eq. 48.

$$Tm_{r,s} = Rf_{r,s} + \sum_{v=1}^{r-1} Rf_{v,1} \quad (48)$$

The Eq. 41 leads to Eq. 49.

$$\langle Tm_{r,s} \rangle = \langle m-r+1 \rangle s - s^2 + \frac{1}{2}(r-1)(2m-r) \quad (49)$$

Tuning time Tm for arbitrarily selected beginning positions of the 2 random walkers represented using Eqs. 50 and 51.

$$\langle Tm \rangle = \frac{2}{m^2} \sum_{r=1}^{m-1} \sum_{s=1}^{m-r} \langle Tm_{r,s} \rangle = \frac{(m-1)^2}{3} + \frac{(m-1)}{3m} \quad (50)$$

$$\langle Tm^2 \rangle = \frac{2}{m^2} \sum_{r=1}^{m-1} \sum_{s=1}^{m-r} \langle Tm^2_{r,s} \rangle \quad (51)$$

$$= \frac{(17m^5 - 51m^4 + 65m^3 - 45m^2 + 8m + 6)}{90m}$$

The mean attractive steps needed to achieve a harmonized state increases nearly proportional to m^2 is given by Eq. 52.

$$\langle Tm \rangle \sim \frac{1}{3m^2} \sim \frac{4}{3\kappa^2} \quad (52)$$

This result is fixed with the scaling performance $\langle tm_{synch} \rangle \propto \kappa^2$ found for TPM harmonization Kanter et al. [23].

The standard deviation of Tm is shown in Eq. 53,

$$SD_{Tm} = \sqrt{\frac{(7m^6 - 11m^5 - 15m^4 + 55m^3 - 72m^2 + 46m - 10)}{90m^2}} \quad (53)$$

system size $m = 2\kappa + 1$. SD_{Tm} is proportional asymptotically to $\langle Tm \rangle$ shown in Eq. 54.

$$SD_{Tm} \sim \sqrt{\frac{7}{10}} \langle Tm \rangle \quad (54)$$

Security analysis

Geometric attack

In this proposed technique a geometric attack is considered on TLTPM. The likelihood of $\gamma_u^E \neq \gamma_u^A$ is represented using the prediction error Ein-Dor and Kanter [17] using Eq. 55,

$$\varepsilon_u^p = \frac{1}{2} \left[1 - \operatorname{erf} \left(\frac{\rho_u}{\sqrt{2(1-\rho_u^2)}} \frac{|h_u|}{\sqrt{Q_u}} \right) \right] \quad (55)$$

of the perceptron. When the u -th hidden neuron does not agree and rest of all hidden units has a condition of $\gamma_v^E \neq \gamma_v^A$, then the likelihood Ruttur et al. [42] of a modification using the geometric attack is shown using Eq. 56.

$$P_k^+ = \int_0^\infty \left(\frac{2}{\sqrt{2\pi Q}} \right)^H \left(\int_{h_u}^\infty \frac{1 - \epsilon^P(h)}{1 - \epsilon} e^{-\frac{h^2}{2Q}} dh \right)^{H-k} \times \left(\int_{h_u}^\infty \frac{\epsilon^P(h)}{\epsilon} e^{-\frac{h^2}{2Q}} dh \right)^{k-1} \frac{\epsilon^P(h_u)}{\epsilon} e^{-\frac{h_u^2}{2Q}} dh_u \quad (56)$$

For identical order parameters $Q = Q_v^E$ and $R = R_j^{AE}$ and various outputs $\gamma_v^A \neq \gamma_v^E$. Then the likelihood for a modification of $\gamma_u^E \neq \gamma_u^A$ is shown using Eq. 57.

$$P_k^+ = \int_0^\infty \left(\frac{2}{\sqrt{2\pi Q}} \right)^H \left(\int_{h_u}^\infty \frac{1 - \epsilon^P(h)}{1 - \epsilon} e^{-\frac{h^2}{2Q}} dh \right)^{H-k} \times \left(\int_{h_u}^\infty \frac{\epsilon^P(h)}{\epsilon} e^{-\frac{h^2}{2Q}} dh \right)^{k-1} \frac{\epsilon^P(h_u)}{\epsilon} e^{-\frac{h_u^2}{2Q}} dh_u \quad (57)$$

By using the same equation the likelihood for an incorrect modification of $\gamma_u^E = \gamma_u^A$ is shown with the help of equation 58.

$$P_k^- = \int_0^\infty \left(\frac{2}{\sqrt{2\pi Q}} \right)^K \left(\int_{h_i}^\infty \frac{1 - \epsilon^P(h)}{1 - \epsilon} e^{-\frac{h^2}{2Q}} dh \right)^{K-k-1} \times \left(\int_{h_i}^\infty \frac{\epsilon^P(h)}{\epsilon} e^{-\frac{h^2}{2Q}} dh \right)^k \frac{1 - \epsilon^P(h_u)}{1 - \epsilon} e^{-\frac{h_u^2}{2Q}} dh_u \quad (58)$$

$\gamma_u^E \neq \gamma_u^A$ condition is satisfied and in total there is an even number of hidden units that satisfied this condition then no geometric modification is done. Equation 59 represents this.

$$P_{r,1}^E = \sum_{u=1}^{H/2} \left(\frac{H-1}{2u-1} \right) (1 - \epsilon)^{H-2u} \epsilon^{2u} \quad (59)$$

Second part of P_r^E can be represented using Eq. 60.

$$P_{r,2}^E = \sum_{u=1}^{H/2} \left(\frac{H-1}{2u-1} \right) P_{2u-1}^- (1 - \epsilon)^{H-2u+1} \epsilon^{2u-1} \quad (60)$$

Third part of P_r^E can be represented using Eq. 61.

$$P_{r,3}^E = \sum_{u=1}^{(H-1)/2} \left(\frac{H-1}{2u} \right) (1 - P_{2u+1}^+) (1 - \epsilon)^{-2u-1} \epsilon^{2u+1} \quad (61)$$

If $H > 1$ then the probability value of attractive steps and repulsive steps in the u -th hidden unit are represented using Eqs. 62 and 63.

$$P_a^E = \frac{1}{2} \left(1 - \sum_{v=1}^3 P_{r,v}^E \right) \quad (62)$$

$$P_r^E = \sum_{v=1}^3 P_{r,v}^E \quad (63)$$

Attractive steps are performed when $H = 1$. If $H = 3$ probability value can be calculated using Eq. 56, that forms Eq. 64 and 65.

$$P_a^E = \frac{1}{2} (1 + 2P_g) (1 - \epsilon)^2 \epsilon + \frac{1}{2} (1 - \epsilon)^3 + \frac{1}{2} (1 - \epsilon)^2 + \frac{1}{6} \epsilon^3 \quad (64)$$

$$P_r^E = 2(1 - P_g) (1 - \epsilon)^2 \epsilon + 2(1 - \epsilon) \epsilon^2 + \frac{2}{3} \epsilon^3 \quad (65)$$

Table 4 Comparison of p_Value of NIST Frequency Test

Techniques	p_Value of NIST Frequency Test
Proposed TLTPM	0.696413
CVTPM Dong and Huang [14]	0.512374
Karakaya et al. [24]	0.1329
Patidar et al. [38]	0.632558
Liu et al. [30]	0.629806

Authentication Steps

In TLTPM input of both parties A and B acts as a common secret. The probability of an input vector $\alpha^{A/B}(t)$ having a particular parity $p \in \{0, 1\}$ is 0.5. For authentication purpose this parity will at this moment use the output bit $\zeta^{A/B}$. At any given time t with common inputs for both parties, the probability of identical output is given in Eq. 66.

$$P(\zeta^A(t) = p = \zeta^B(t)) = \frac{1}{2} \quad (66)$$

Given a number n ($1 \leq n \leq \alpha$) of pure authentication steps, in which one transmits the parity of the consequent input vector as output $\zeta^{A/B}$ directly, the probability that the two parties subsequently produce the same output n times (and thus are likely to have the same n inputs) decreases exponentially with n i.e. $P(\zeta^A(t) = p = \zeta^B(t)) = \frac{1}{2^n}$. For statistical security of $\varepsilon \in [0, 1]$ select $n = \alpha$ authentication steps such that $1 - \frac{1}{2^\alpha} \geq \varepsilon$ which can be computed as $\alpha = \log_2 \left[\frac{1}{1-\varepsilon} \right]$. With $\alpha = 14$ the achievable statistical security $\varepsilon = 0.9999$ i.e. 99.9999%. The synchronization period for this technique therefore increases by α authentication steps depending on the necessary level of security ε . Select a certain bit subpattern in the input vector used for authentication only, such that the security threshold will be reached soon enough with a certain probability. Inputs are uniformly distributed so the last m bits is also uniformly distributed. Now select those entries that possess a defined bit sub-pattern (e.g. 0101 for $m = 4$). The probability of such a fixed bit sub pattern of m bit to occur is $\frac{1}{2^m}$, because each bit has a fixed value with a probability of 0.5. Thus, for four bits, on average every sixteenth input would be used for authentication. Authentication step is performed when the subpattern arises and then one of the party sends out the parity of the consequent input vector as output $\zeta^{A/B}$. This will only occur at the other party if it has the same inputs. Such an authentication does not manipulate the learning process at all. Because of the truth that the inputs are secret, an attacker cannot know when exactly such an authentication procedure takes place.

Secret key space analysis

Consider n number of cascading encryption/decryption techniques is used to encrypt/decrypt the plaintext with the help of a neural synchronized session key. Then a session key of length [(number of cascaded encryption technique in bits) + (three bits combinations of encryption/ decryption technique index) + (length of n number of encryption/decryption keys in bits) + (length of n number of session keys in bits)] i.e. $[8 + (3 \times n) + (128 \times n) + (128 \times n)]$ bits to $[8 + (3 \times n) + (256 \times n) + (256 \times n)]$ number of bits. So, $\frac{[8 + (3 \times n) + (128 \times n) + (128 \times n)]}{8} = \left[1 + \frac{(3 \times n)}{8} + 16n + 16n\right] = 32n$ to $\frac{[8 + (3 \times n) + (256 \times n) + (256 \times n)]}{8} = \left[1 + \frac{(3 \times n)}{8} + 32n + 32n\right] = 64n$ numbers of characters.

Therefore, the total number of keys = 256^{64n} . Attacker checks with half of the possible key in an average, the time needed at 1 decryption/ $\mu s = 0.5 \times 256^{64n} \mu s = 0.5 \times 2^{8 \times 64n} \mu s = 0.5 \times 2^{512n} \mu s = 2^{(512n-1)} \mu s$.

Consider any single encryption using the neural key of size 512 bit which is hypothetically approved and needed to be analyzed in the context of the time taken to crack a ciphertext with the help of the fastest supercomputers available at present. In this neural technique to crack a ciphertext, the number of permutation combinations on the neural key is $2^{512} = 1.340780 \times 10^{154}$ trials for a size of 512 bits only. IBM Summit at Oak Ridge, U.S. invented the fastest supercomputer in the world with 148.6 PFLOPS i.e. means 148.6×10^{15} floating-point computing/second. Certainly, it can be considered that each trial may require 1, 000 FLOPS to undergo its operations. Hence, the total test needed per second is 148.6×10^{12} . Total no. of sec. a year have = $365 \times 24 \times 60 \times 60 = 31,536,000$ sec. The total number of years for Brute Force attack: $(1.340780 \times 10^{154}) / (148.6 \times 10^{12} \times 31,536,000) = 2.86109 \times 10^{132}$ years.

Results and analysis

For result and simulation purpose, an Intel Core i7 10th Generation, 2.6 GHz processor, 16 GB RAM is used. Comprehensive and needful security views have been focused to affect acquaintance security and robustness issues. The precision of 10^{-15} decimal has been used in arithmetic operations according to the authenticated IEEE Standard 754.

True randomness was ensured in the proposed transmission technique by passing the fifteen tests contained in that suite. These tests are very useful for such a proposed technique with high robustness. A probability value (p-Value) determines the acceptance or rejection of the input vector generated by the GAN. Table 2 contains the results of

Table 5 Results after 500,000 simulations with different values of $H1 - H2 - H3 - M - \kappa$

$H1 - H2 - H3 - M - \kappa$	No. of Min Sync. Steps	No. of Max Sync. Steps	Min Sync. time (sec.)	Max Sync. time (sec.)	Successful Sync. of Attacker E	% of Successful Sync. of Attacker E (%)
(1-1-1-512-1)	10	36	0.0745	3.7865	3,47,264	69.452
(2-2-2-256-1)	11	33	0.0724	2.9823	3,42,894	68.578
(4-4-4-128-1)	11	42	0.0834	3.5783	3,44,513	68.902
(8-8-8-64-1)	9	29	0.0790	3.1976	3,49,834	69.966
(1-1-1-256-2)	11	49	0.0739	3.2387	3,37,823	67.564
(2-2-2-128-2)	11	38	0.0698	672.7910	3,32,897	66.579
(4-4-4-64-2)	12	245	0.0756	130.6702	3,01,986	60.397
(8-8-8-32-2)	13	986	0.0672	39.2398	1,18,740	23.748
(1-1-1-128-8)	16	98	0.0670	2.2398	1,13,902	22.780
(2-2-2-64-8)	20	543	0.0712	3.9823	39,198	7.8396
(4-4-4-32-8)	25	876	0.0543	6.7890	178	0.0356
(8-8-8-16-8)	45	797	0.0389	0.0789	0	0.0
(1-1-1-64-128)	11	134	0.0280	0.7907	1,29,873	25.974
(2-2-2-32-128)	19	498	0.0549	3.8763	35,289	7.057
(4-4-4-16-128)	28	984	0.0402	4.1276	569	0.1138
(8-8-8-8-128)	40	1107	0.0523	5.6392	0	0.0

Table 6 Results after 1000,000 simulations for the (8-8-8-16-8) and (8-8-8-8-128) combinations

$H1 - H2 - H3 - M - \kappa$	No. of min Sync. steps	No. of max Sync. steps	Min Sync. time (sec.)	Max Sync. time (s)	Successful Sync. of attacker E	% of Successful Sync. of attacker E (%)
(8-8-8-16-8)	46	863	0.0839	2.8185	0	0
(8-8-8-8-128)	37	937	0.0417	1.3896	2	0.0002

NIST 15 Statistical tests NIST [37] on the generated random input vector. This table also represents the comparison of p_Value between the proposed TLTPM and the existing CVTPM technique Dong and Huang [14]. Here, p_Value of the proposed TLTPM of the average of 5 and 10 iterations, and p_Value of the existing CVTPM of the average of 10 iterations are represented. From Table 3, it has been seen that in the NIST statistical test p_Value of the proposed TLTPM has outperformed than the CVTPM. This confirms that the GAN-generated input vector has better randomness than the existing CVTPM.

The result of the frequency test indicates a ratio of 0 and 1 in the generated random sequence. Here, the value of the frequency test is 0.696413 which is quite average Kanso and Smaoui [22] and better than the result of frequency test 0.1329 in Karakaya et al. [24], and 0.632558 in Patidar et al.

[38] 0.629806 in Liu et al. [30]. Comparison of p_Value of NIST frequency Test is given in Table 4.

The results of simulations for different $H1 - H2 - H3 - M - \kappa$ are shown in Table 5. The number of minimum and maximum synchronization steps indicates a number of minimum and maximum steps to synchronize the weight of the two networks. The column for the average steps indicates the sum of all simulations conducted. The minimum and maximum synchronization time columns indicate the minimum and maximum time in seconds required to synchronize the weights of the two networks. The successful synchronization of the attacker E column shows successfully how many instances that the attacking network imitated the behaviors of the other two networks for total simulations performed. Finally, the percentage of successful synchronization of the attacker E (%) column shows successfully the percentage of total simulations shown by the earlier column.

Table 7 Comparison of synchronization time for fixed network size and variable learning rules and synaptic depth in proposed TLTPM and existing CVTPM method

Weight range value	Sync. time in Hebbian learning rule (in cycle) in proposed TLTPM	Sync. time in Anti-Hebbian learning rule (in cycle) in proposed TLTPM	Sync. time in random walk learning rule (in cycle) in proposed TLTPM	Sync. time in Hebbian learning rule (in cycle) in existing CVTPM Dong and Huang[14]	Sync. time in Anti-Hebbian Learning Rule (in Cycle) in existing CVTPM Dong and Huang [14]	Sync. time in Random Walk Learning Rule (in Cycle) in existing CVTPM Dong and Huang [14]
5	312.56	385.35	394.03	348.16	389.26	415.63
10	681.34	739.46	696.05	813.64	753.09	713.94
15	997.36	1085.03	1017.31	1769.57	1198.47	1104.81
20	2374.63	1623.39	1650.35	2971.04	1857.24	1696.35
25	3512.09	2393.52	2483.18	4186.59	2671.17	2540.62
30	4998.46	3152.27	3294.10	5982.41	3582.75	3318.49
35	6983.27	4418.79	4128.23	7310.37	4793.62	4518.27
40	8312.03	5718.39	4673.71	8953.50	5981.43	5372.11
45	11016.29	6684.32	6393.49	11207.26	8138.39	7363.29
50	14439.59	8736.26	7786.38	14952.38	9452.57	8720.93

As shown in Table 5, the best results for the defense against the attacking network are the combinations (8-8-8-16-8) and (8-8-8-8-128), respectively. E did not mimic the behaviors of A and B's TLTPM in any of the 500,000 simulations. Table 6 shows that the best combination of values for the trio is (8-8-8-16-8).

Table 7 shows the comparison of synchronization time for fixed network size and variable learning rules and synaptic depth in the proposed TLTPM and existing CVTPM method. From the table, it has been seen that a trend towards an increase in the synchronization steps as the range of weight values κ increases in all three learning rules. For small κ values, Hebbian takes fewer synchronization steps than the other two learning rules in the range of 5 to 15, but as the κ value increases the Hebbian rule, more steps are taken to synchronize than the other two learning rules. Here, the Anti-Hebbian rules take less time than the other two learning rules in the 20 to 30 range. Random Walk outperforms 35 and beyond. As a result, an increase in the value of the κ security of the system can be increased.

Conclusion and future scope

In this paper, a GAN-based public-key exchange protocol is proposed. By presenting several alterations to the GAN system, this research makes lots of innovative contributions. This paper presents a rationalization of the GAN framework applicable to this work, in which a reference dataset that the

generator should acquire knowledge to mimic is not included in the GAN. Also, instead of a recurrent network, this work design the statefulness of a PRNG that used a feed-forward neural network with additional non-random "counter" inputs. For generating different key lengths using neural synchronization process various combinations of $H1 - H2 - H3 - M - \kappa$ with variable network size are considered. The security and synchronization time of GAN-based TLTPM is also being investigated. A geometric attack is also considered and it has been shown that it has a lower success rate. GAN-based TLTPM security is found to be higher than TPM with the same set of TPM network parameters. Finally, a variety of results and analysis are performed to confirm the experimental results. As future work, a more comprehensive analysis of security is planning to carry out. Also, different nature-inspired optimization algorithms will be considered for the optimization of weights value for faster synchronization purposes.

Acknowledgements The author expressed deep gratitude for the moral and congenial atmosphere support provided by Ramakrishna Mission Vidyamandira, Belur Math, India.

Compliance with ethical standards

Funding This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors. his research received no external fundings.

Conflicts of Interest No conflict of Interest.

financial interests The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

competing interests The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Abadi M, Andersen DG (2016) Learning to protect communications with adversarial neural cryptography. [arXiv:1610.06918](https://arxiv.org/abs/1610.06918)
- Allam AM, Abbas HM, El-Kharashi MW (2013) Authenticated key exchange protocol using neural cryptography with secret boundaries. In: Proceedings of the 2013 international joint conference on neural networks, IJCNN 2013, pp 1–8
- Balasubramaniam P, Muthukumar P (2014) Synchronization of chaotic systems using feedback controller: an application to Diffie–Hellman key exchange protocol and ElGamal public key cryptosystem. *J Egypt Math Soc* 22(3):365–372. <https://doi.org/10.1016/j.joems.2013.10.003>
- Bassham LE, Rukhin AL, Soto J, Nechvatal JR, Smid E, Leigh SD, Levenson M, Vangel M, Heckert NA, Banks DL (2010) A statistical test suite for random and pseudorandom number generators for cryptographic applications. National Institute of Standards and Technology
- Bauer FL (2011) Cryptology. In: van Tilborg HCA, Jajodia S (eds) Encyclopedia of cryptography and security, Springer, Boston, MA, pp 283–284. <https://doi.org/10.1007/978-1-4419-5906-5>
- Chen H, Shi P, Lim CC (2017) Exponential synchronization for Markovian stochastic coupled neural networks of neutral-type via adaptive feedback control. *IEEE Trans Neural Netw Learn Syst* 28(7):1618–1632. <https://doi.org/10.1109/TNNLS.2016.2546962>
- Chen H, Shi P, Lim CC (2019) Cluster synchronization for neutral stochastic delay networks via intermittent adaptive control. *IEEE Trans Neural Netw Learn Syst* 30(11):3246–3259. <https://doi.org/10.1109/tnnls.2018.2890269>
- Cover TM, Thomas JA (2006) Elements of information theory. Wiley Series in Telecommunications and Signal Processing, 2nd edition, Wiley, New York
- Desai V, Deshmukh V, Rao D (2011) Pseudo random number generator using elman neural network. In: and others (ed) 2011 IEEE recent advances in intelligent computational systems, IEEE, pp 251–254. <https://doi.org/10.1109/RAICS.2011.6069312>
- Desai V, Patil RT, Deshmukh V, Rao D (2012) Pseudo random number generator using time delay neural network. *World* 2(10):165–169
- Diffie W, Hellman M (1976) New directions in cryptography. *IEEE Trans Inf Theory* 22(6):644–654. <https://doi.org/10.1109/tit.1976.1055638>
- Dolecki M, Kozera R (2013) Threshold Method of Detecting Long-Time TPM Synchronization. In: K S, R C, A C, S W (eds) Computer Information Systems and Industrial Management. CISIM 2013, Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, vol 8104, pp 241–252. https://doi.org/10.1007/978-3-642-40925-7_23
- Dolecki M, Kozera R (2015) The Impact of the TPM Weights Distribution on Network Synchronization Time. In: K S, W H (eds) Computer Information Systems and Industrial Management. CISIM 2015, Springer, Cham, Switzerland, Lecture Notes in Computer Science, vol 9339, pp 451–460. https://doi.org/10.1007/978-3-319-24369-6_37
- Dong T, Huang T (2020) Neural cryptography based on complex-valued neural network. *IEEE Trans Neural Netw Learn Syst* 31(11):4999–5004. <https://doi.org/10.1109/TNNLS.2019.2955165>
- Dong T, Wang A, Zhu H, Liao X (2018) Event-triggered synchronization for reaction–diffusion complex networks via random sampling. *Physica A Stat Mech Appl* 495:454–462. <https://doi.org/10.1016/j.physa.2017.12.008>
- Eftekhari M (2012) A Diffie–Hellman key exchange protocol using matrices over noncommutative rings. *Groups Compl Cryptol* 4(1):167–176. <https://doi.org/10.1515/gcc-2012-0001>
- Ein-Dor L, Kanter I (1999) Confidence in prediction by neural networks. *Phys Rev E* 60(1):799–802. <https://doi.org/10.1103/physreve.60.799>
- Engel A, den Broeck CV (2012) Statistical mechanics of learning. Cambridge University Press, Cambridge. <https://doi.org/10.1017/CBO9781139164542>
- Gomez H, Reyes Óscar, Roa E (2017) A 65 nm CMOS key establishment core based on tree parity machines. *Integration* 58:430–437. <https://doi.org/10.1016/j.vlsi.2017.01.010>
- Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. In: NIPS'14: Proceedings of the 27th international conference on neural information processing systems, vol 2, pp 2672–2680
- Jeong YS, Oh K, Cho CK, Choi HJ (2018) Pseudo random number generation using lstms and irrational numbers. *Big Data and Smart Computing (BigComp)*, 2018 IEEE international conference on, pp 541–544
- Kanso A, Smaoui N (2009) Logistic chaotic maps for binary numbers generations. *Chaos Solit Fract* 40(5):2557–2568. <https://doi.org/10.1016/j.chaos.2007.10.049>
- Kanter I, Kinzel W, Kanter E (2002) Secure exchange of information by synchronization of neural networks. *Europhys Lett (EPL)* 57(1):141–147. <https://doi.org/10.1209/epl/i2002-00552-9>
- Karakaya B, Gülten A, Frasca M (2019) A true random bit generator based on a memristive chaotic circuit: analysis, design and FPGA implementation. *Chaos Solit Fract* 119:143–149. <https://doi.org/10.1016/j.chaos.2018.12.021>
- Kelsey J, Schneier B, Wagner D, Hall C (1998) Cryptanalytic Attacks on Pseudorandom Number Generators. In: S V (ed) Fast Software Encryption. FSE, Springer, vol 1372
- Kinzel W, Kanter I (2002) Interacting neural networks and cryptography. In: B K (ed) Advances in Solid State Physics, Springer, Berlin, vol 42, pp 383–391. https://doi.org/10.1007/3-540-45618-X_30
- Klimov A, Mityagin A, Shamir A (2002) Analysis of Neural Cryptography. In: Y Z (ed) Advances in Cryptology — ASIACRYPT 2002. ASIACRYPT 2002. Lecture Notes in Computer Science, Springer, Berlin, vol 2501, pp 288–298. https://doi.org/10.1007/3-540-36178-2_18
- Lakshmanan S, Prakash M, Lim CP, Rakkiyappan R, Balasubramaniam P, Nahavandi S (2018) Synchronization of an inertial neural network with time-varying delays and its application to secure com-

- munication. *IEEE Trans Neural Netw Learn Syst* 29(1):195–207. <https://doi.org/10.1109/tnnls.2016.2619345>
29. Lindell Y, Katz J (2014) Introduction to modern Cryptography. Cryptography and Network Security Series), Chapman and Hall/CRC
 30. Liu L, Miao S, Hu H, Deng Y (2016) Pseudo-random bit generator based on non-stationary logistic maps. *IET Inf Secur* 10(2):87–94. <https://doi.org/10.1049/iet-ifs.2014.0192>
 31. Liu P, Zeng Z, Wang J (2019) Global synchronization of coupled fractional-order recurrent neural networks. *IEEE Trans Neural Netw Learn Syst* 30(8):2358–2368. <https://doi.org/10.1109/TNNLS.2018.2884620>
 32. Meneses F, Fuertes W, Sancho J (2016) RSA encryption algorithm optimization to improve performance and security level of network messages. *IJCSNS* 16(8):55
 33. Mu N, Liao X (2013) An approach for designing neural cryptography. In: C G, ZG H, Z Z (eds) *Advances in Neural Networks*, Springer, Lecture Notes in Computer Science, vol 7951, pp 99–108. https://doi.org/10.1007/978-3-642-39065-4_13
 34. Mu N, Liao X, Huang T (2013) Approach to design neural cryptography: a generalized architecture and a heuristic rule. *Phys Rev E* 87(6). <https://doi.org/10.1103/physreve.87.062804>
 35. Ni Z, Paul S (2019) A multistage game in smart grid security: a reinforcement learning solution. *IEEE Trans Neural Netw Learn Syst* 30(9):2684–2695. <https://doi.org/10.1109/tnnls.2018.2885530>
 36. Niemiec M (2018) Error correction in quantum cryptography based on artificial neural networks. *Quant Inf Process* 18:174. <https://doi.org/10.1007/s11128-019-2296-4>
 37. NIST (2020) NIST Statistical Test. http://csrc.nist.gov/groups/ST/toolkit/rng/stats_tests.html
 38. Patidar V, Sud KK, Pareek NK (2009) A pseudo random bit generator based on chaotic logistic map and its statistical testing. *Informatica* 33:441–452
 39. Pu X, Tian XJ, Zhang J, Liu CY, Yin J (2017) Chaotic multimedia stream cipher scheme based on true random sequence combined with tree parity machine. *Multimed Tools Appl* 76(19):19881–19895. <https://doi.org/10.1007/s11042-016-3728-0>
 40. Rosen-Zvi M, Kanter I, Kinzel W (2002) Cryptography based on neural networks analytical results. *J Phys A: Math Gen* 35(47):L707–L713. <https://doi.org/10.1088/0305-4470/35/47/104>
 41. Ruttur A (2007) Neural synchronization and cryptography. <https://arxiv.org/abs/0711.2411>
 42. Ruttur A, Kinzel W, Naeh R, Kanter I (2006) Genetic attack on neural cryptography. *Phys Rev E* 73(3). <https://doi.org/10.1103/physreve.73.036121>
 43. Santhanalakshmi S, Sangeeta K, Patra GK (2015) Analysis of neural synchronization using genetic approach for secure key generation. *Commun Comput Inf Sci* 536:207–216
 44. Sarkar A, Mandal JK (2012a) Artificial neural network guided secured communication techniques: a practical approach. LAP LAMBERT Academic Publishing Germany
 45. Sarkar A, Mandal JK (2012b) Key Generation and certification using multilayer perceptron in wireless communication (KGCMLP). *Int J Secur Privacy Trust Manag (IJSPTM)* 1(5):2319–4103
 46. Sarkar A, Dey J, Bhowmik A (2019a) Multilayer neural network synchronized secured session key based encryption in wireless communication. *Indonesian J Electr Eng Comput Sci* 14(1):169. <https://doi.org/10.11591/ijeecs.v14.i1.pp169-177>
 47. Sarkar A, Dey J, Bhowmik A, Mandal JK, Karforma S (2019b) Computational intelligence based neural session key generation on e-health system for ischemic heart disease information sharing. In: J M, D S, J B (eds) *Contemporary Advances in Innovative and Applicable Information Technology. Advances in Intelligent Systems and Computing*, Springer, vol 812
 48. Sarkar A, Dey J, Chatterjee M, Bhowmik A, Karforma S (2019c) Neural soft computing based secured transmission of intraoral gingivitis image in e-health care. *Indonesian J Electr Eng Comput Sci* 14(1):178. <https://doi.org/10.11591/ijeecs.v14.i1.pp178-184>
 49. Steiner M, Tsudik G, Waidner M (1996) Diffie-Hellman key distribution extended to group communication. In: CCS '96: Proceedings of the 3rd ACM conference on Computer and communications security, pp 31–37. <https://doi.org/10.1145/238168.238182>
 50. Tirdad K, Sadeghian A (2010) Hopfield neural networks as pseudo random number generators. 2010 Annual Meeting of the North American Fuzzy Information Processing Society pp 1–6. <https://doi.org/10.1109/NAFIPS.2010.5548182>
 51. Wang A, Dong T, Liao X (2016) Event-triggered synchronization strategy for complex dynamical networks with the Markovian switching topologies. *IEEE Trans Neural Netw Learn Syst* 27:52–57
 52. Wang J, Cheng LM, Su T (2018) Multivariate cryptography based on clipped Hopfield neural network. *IEEE Trans Neural Netw Learn Syst* 29(2):353–363. <https://doi.org/10.1109/tnnls.2016.2626466>
 53. Wang JL, Qin Z, Wu HN, Huang T (2019) Passivity and synchronization of coupled uncertain reaction–diffusion neural networks with multiple time delays. *IEEE Trans Neural Netw Learn Syst* 30(8):2434–2448. <https://doi.org/10.1109/TNNLS.2018.2884954>
 54. Xiao Q, Huang T, Zeng Z (2019) Global exponential stability and synchronization for discrete-time inertial neural networks with time delays: a timescale approach. *IEEE Trans Neural Netw Learn Syst* 30(6):1854–1866. <https://doi.org/10.1109/TNNLS.2018.2874982>
 55. Zhang Z, Cao J (2019) Novel finite-time synchronization criteria for inertial neural networks with time delays via integral inequality method. *IEEE Trans Neural Netw Learn Syst* 30(5):1476–1485. <https://doi.org/10.1109/TNNLS.2018.2868800>
 56. Zhou X, Tang X (2011) Research and implementation of RSA algorithm for encryption and decryption. In: Proceedings of 2011 6th International Forum on Strategic Technology, IEEE, pp 1118–1121. <https://doi.org/10.1109/IFOST.2011.6021216>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.