Hindawi
Security and Communication Networks
Volume 2021, Article ID 6680782, 12 pages
https://doi.org/10.1155/2021/6680782

Research Article

Neural Cryptography Based on Generalized Tree Parity Machine for Real-Life Systems

Sooyong Jeong,[1] **Cheolhee Park**,[2] **Dowon Hong**,[2] **Changho Seo**,[1] **and Namsu Jho**[3]

[1]*Department of Convergence Science, Kongju National University, Kongju 32588, Republic of Korea*
[2]*Department of Mathematics, Kongju National University, Kongju 32588, Republic of Korea*
[3]*Electronics and Telecommunications Research Institute (ETRI), Daejeon 34129, Republic of Korea*

Correspondence should be addressed to Dowon Hong; dwhong@kongju.ac.kr

Received 29 October 2020; Revised 30 December 2020; Accepted 19 January 2021; Published 4 February 2021

Academic Editor: Flavio Lombardi

Traditional public key exchange protocols are based on algebraic number theory. In another perspective, neural cryptography, which is based on neural networks, has been emerging. It has been reported that two parties can exchange secret key pairs with the synchronization phenomenon in neural networks. Although there are various models of neural cryptography, called Tree Parity Machine (TPM), many of them are not suitable for practical use, considering efficiency and security. In this paper, we propose a Vector-Valued Tree Parity Machine (VVTPM), which is a generalized architecture of TPM models and can be more efficient and secure for real-life systems. In terms of efficiency and security, we show that the synchronization time of the VVTPM has the same order as the basic TPM model, and it can be more secure than previous results with the same synaptic depth.

1. Introduction

Traditionally, protocols based on public key cryptosystems have been widely used in key exchange (e.g., Diffie–Hellman [1] and RSA [2]) and the shared secret keys can be used in many applications, such as digital certificate, digital signature, and embedded system [2–4]. These key exchange protocols are fundamentally based on algebraic number theory [5]. As an alternative approach for a public key system, research studies on neural cryptography have been conducted [6–11]. Instead of the traditional number theory-based cryptography, neural cryptography can build key exchange protocols based on the synchronization phenomenon in neural networks [8]. Moreover, the neural cryptography ensure that the key cannot be inferred, even if an attacker knows the details of the algorithm and can monitor the communication channel. By sharing the same neural network structure (called a Tree Parity Machine, TPM), both entities that are involved in key exchange protocol can share a secret key by synchronizing the shared neural network.

In the conventional neural cryptography [8], each party constructs own TPM with shared parameters and chooses initial random weight values for the TPM. Then, they generate a random input vector and calculate their own output values by feeding the generated common input into the TPM. By exchanging the output values, they update their own weight values with a given learning rule. These procedures are repeated until the weight vectors are fully synchronized, and the synchronized weight vector can be used as a shared secret key (Note that the initial weight vector should be kept secret, as with the private key of the PKC system, while on the other hand, the input/output values can be known to anyone, including adversaries). Similar to the existing PKC system, the exchanged keys can be used in various applications. For example, the key which is generated by neural cryptography can be used in block cipher for encryption such as SDES, AES, and Rijndael [12, 13]. Moreover, it can be used in the stream cipher by applying it to the LFSR structure [14].

In the field of neural cryptography, various key exchange protocols have been proposed to improve security and efficiency. These protocols can be divided into three categories according to their approaches: sanitizing input value,

disturbing output value, and reconstructing model architecture. In the case of sanitizing input value, input values are generated by adding uncertainty to input values depending on the hidden units of participants to make it advantageous for bidirectional learning (participant-side) and disadvantageous for unidirectional learning (adversary-side) [15, 16]. In the case of disturbing output value, participants add noise to calculated output values to prevent an adversary from identifying the real output values [17, 18]. However, this approach requires the additional assumption that participants have to previously share auxiliary information. In terms of reconstructing model architecture, research has been conducted to improve the security and efficiency of neural cryptographic algorithms by rebuilding internal components of the TPM [19–21]. It was reported that the efficiency of the TPM can be improved by transforming the process of output calculation and expanding the number of output values [19, 20]. Recently, a method that can improve the security of the TPM while preserving its efficiency by extending the structure of the TPM was proposed [21]. In [21], the authors applied complex numbers to all internal components, instead of an integer system, to extend the original TPM model. Especially, they showed that participants can exchange a pair of secret keys with a higher level of security. However, it is still difficult to ensure the reasonable levels of security required in real-life systems with these results.

In this paper, we propose a Vector-Valued Tree Parity Machine (VVTPM), which is an extended model of the basic TPM architecture, in which we apply a vector-valued system to the internal components of the TPM. Since the parameters of the VVTPM are vector values, our architecture can generate multiple pairs of secret keys in a run of the protocol while improving security and preserving efficiency. Moreover, the VVTPM can be a generalized model for architecturally extended TPM models, including the original TPM. Since our approach can control the size of the secret key by varying the number of vectors, the VVTPM can achieve a reasonable security level to apply to real-life systems. In order to verify the improvement of security, we theoretically analyze the synchronization process on both participant-side and adversary-side. Then, we show that the VVTPM can improve security with the same degree of synchronization as the original TPM. Furthermore, we show that our model can be applied to real-life systems with comprehensive experiments under various conditions.

◉ Contribution: this paper provides four main contributions:

(1) A novel model of neural cryptography: we propose a novel model of neural cryptography, called Vector-Valued Tree Parity Machine, which can generate flexible lengths of secret keys by varying the number of vectors while increasing security and preserving the synchronization time.

(2) A generalized version of the existing TPM models: from the perspective of reconstructing model architecture, the VVTPM can be a generalized version of the structural expanded models, including the original TPM model. Besides the basic TPM, we show that recent results (i.e., the complex-valued model [21]) can be interpreted by our model.

(3) Theoretical proof of security and efficiency: we first show that the synchronization time of the VVTPM has the same order as the original TPM model, which means that efficiency can be preserved. Then, we prove that, with the same synaptic depth, the security of the VVTPM can be increased according to the number of vectors.

(4) Experimental verification: in our experiment, we apply the most powerful attacker in an adversarial scenario that has not been considered in recent work [21, 22]. Furthermore, we explore various learning rules to show that our model does not depend on a specific learning rule. Along with the theoretical analysis, we experimentally show, from the perspective of the efficiency, that the VVTPM can synchronize a shared model with the same rounds as the basic TPM. Finally, we demonstrate the possibility that the VVTPM can achieve the reasonable security level required in real-life systems.

This paper is organized as follows: Section 2 describes the related work that is the basis of this study, and we explain the existing TPM and attack scenarios in Section 3. Section 4 proposes the novel model, which is called Vector-Valued Tree Parity Machine and describes the learning process. Then, we theoretically analyze synchronization time and security in Section 5 and show the empirical results of our experiments in Section 6. Finally, we conclude in Section 7.

2. Related Work

Various research studies have been studied on neural cryptography, and we briefly review prior work on neural cryptography. We categorize the previous studies into four categories:

Original TPM: extensive studies have been conducted on neural cryptography. Mislovaty et al. [6], Rosen-Zvi et al. [7], and Kanter et al. [8] proved that the synchronization of two Tree Parity Machines (TPMs) can be achieved by mutual learning rules. In particular, it was shown that the synchronization of the TPMs can be used as a cryptographic key exchange protocol [8, 9]. Starting with these results, Ruttor et al. [10] analyzed the process of synchronization in the overall TPM structure and showed that there are two main steps, attractive and repulsive steps. Based on these steps, Ruttor et al. [11] proved that the synchronization time of two TPMs depends on the synaptic depth L. In [11], the authors showed that the synchronization of a single weight value between TPMs is identical to the gamblers'

ruin problem [23], and the synchronization of the TPMs can be proved by an extended theorem of gamblers' ruin problem.

Advanced protocols: in order to use TPM-based neural cryptography, various results that extend the basic concept of the TPM have been proposed. Santhanalakshmi et al. [24] and Allam and Abbas [25] proposed new protocols for exchanging group keys by extending basic neural cryptography. As an extended building block of key exchange, Volkmer [26] proposed an authenticated key exchange protocol based on the TPM. In addition, Allam et al. [27] showed that key exchange with authentication can be achieved through secret boundaries. As mentioned above, since it is difficult to use the traditional PKC algorithms in resource-constrained environments, Chen et al. [28] proposed TinyTPM to enable key exchange in embedded systems. For the practical use of TPM-based key exchange protocols, Volkmer and Wallner [29] proposed a rekeying algorithm for generating new keys by reusing the previously exchanged key.

Security under attack scenarios: to analyze the security of the TPM models, various attack models have been proposed [30–32] (e.g., simple attack, geometric attack, genetic attack, and majority attack). In security analysis under various attack scenarios, it was reported that participants can prevent attacks by increasing synaptic depth. However, if the synaptic depth is increased, the efficiency of the TPM is decreased; that is, the synchronization time of the TPM can increase along with the synaptic depth. In order to investigate TPM parameters that satisfy reasonable security, Salguero Dorokhin et al. [22] experimentally analyzed security with a geometric attack by varying the internal parameters. However, they only considered the geometric attack, and hence, it is unclear whether their optimal parameters satisfy reasonable security against a majority attack, which is the most powerful attack.

Variations of TPM: in order to improve efficiency and security, various key exchange protocols have been proposed. These protocols can be divided into three categories according to their approaches: sanitizing input values, disturbing output values, and reconstructing model architecture. In the case of sanitizing input values, hidden units are used for generating input values in order to confuse the input values [15]. Since the attacker cannot know the hidden units of the two participants, public input values are partially changed into private values. Therefore, it is more difficult for the attacker to attack TPMs than before due to the confidentiality of the input values. On the other hand, hidden units can indirectly affect the generation of input values. In order to accelerate bidirectional learning, participants generate input values related to hidden units of them, instead of random values [16]. In the case of disturbing output values, a mechanism, called Do not Trust My Partner (DTMP), has been proposed [17, 18]. DTMP allows two participants to add preagreed noise to a calculated output value to confuse an attacker who tries to use public output values maliciously. However, this mechanism requires an additional condition that the two participants must have a prior consultation for noise generation. In terms of reconstructing model architecture, new mechanisms that are modified from the original TPM by rebuilding internal components are proposed. While the original TPM generates an output value by the product of hidden units, the mechanism proposed in [19] generates the output value through a more complicated process in the output calculation. By extending this method, the Two-layer Tree-connected Feedforward Neural Network (TTFNN), which generates a 2 bit output value, has been proposed [20]. Recently, an architecture that can improve the security of the TPM while preserving the efficiency was proposed by extending the construct of the original TPM [21]. In [21], the authors proposed the Complex-Valued Tree Parity Machine (CVTPM) that applies complex numbers for all internal parameters and showed that the CVTPM can ensure a higher level of security. However, they only considered the geometric attack, and hence, it is unclear whether the CVTPM satisfies reasonable security against the majority attack.

In this paper, we propose a novel model of the TPM, called Vector-Valued Tree Parity Machine (VVTPM), in which the architecture can generalize the previously proposed algorithms and satisfies a reasonable level of security for real-life systems while preserving efficiency.

3. Background

Before discussing our neural cryptography model, we explain the original neural cryptography called a Tree Parity Machine. Moreover, we also explain attack scenarios to experimentally verify the security of various TPMs, including our model which is presented in Section 4.

3.1. Tree Parity Machine. The original TPM is a multilayer feedforward network that consists of an input layer, an output layer, and one hidden layer. The input layer consists of $N \times K$ binary values $x_{i,j} \in \{-1, 1\}$, and participants determine random (common) values for each round. The hidden layer consists of independent K values, and each value is connected with N input values. Generally, the number of hidden unit K is fixed at 3 considering the security and efficiency. Each hidden unit is calculated using N input values and weight values, where the weight values have integer values between –L and L ($w_{i,j} \in \{-L, -L+1, \ldots, L\}$). Note that the index $i = 1, \ldots, K$ denotes the i-th hidden unit, and the index $j = 1, 2, \ldots, N$ denotes the j-th input value. The i-th hidden unit σ_i is calculated by the product of the corresponding input and weight values as follows [8]:

$$\sigma_i = \operatorname{sgn}(h_i),$$
$$h_i = \frac{1}{\sqrt{N}}\mathbb{W}_i \cdot \mathbb{X}_i = \frac{1}{\sqrt{N}}\sum_{j=1}^{N} w_{i,j} \cdot x_{i,j}, \tag{1}$$

where $\mathbb{W}_i$ is the vector of weight values (e.g., $\mathbb{W}_i = [w_{i,1}, w_{i,2}, \ldots, w_{i,N}]$) and $\mathbb{X}_i$ is the vector of input values (e.g. $\mathbb{X}_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,N}]$). Moreover, the intrinsic value h_i is called a local field of the hidden layer, and $\operatorname{sgn}(\cdot)$ is a sign function. If the local field h_i of the hidden layer is 0, hidden unit σ_i is set by -1. Consequently, the output τ of the TPM is calculated by the product of the hidden units:

$$\boldsymbol{\tau} = \prod_{i=1}^{K} \sigma_i, \tag{2}$$

where the result becomes a binary value either 1 or -1.

Based on a given structure of the TPM, the sender and receiver randomly initialize weight values, generate new random input values for each round, and exchange calculated output values. Then, they update their own weight values according to learning rule when the outputs of the two parties are the same value, as follows [8]:

(a) Hebbian learning rule:

$$w_{i,j}^{+} = g\left(w_{i,j} + x_{i,j}\tau\theta(\sigma_i\tau)\theta\left(\tau^A\tau^B\right)\right). \tag{3}$$

(b) Anti-Hebbian learning rule:

$$w_{i,j}^{+} = g\left(w_{i,j} - x_{i,j}\tau\theta(\sigma_i\tau)\theta\left(\tau^A\tau^B\right)\right). \tag{4}$$

(c) Random walk learning rule:

$$w_{i,j}^{+} = g\left(w_{i,j} + x_{i,j}\theta(\sigma_i\tau)\theta\left(\tau^A\tau^B\right)\right). \tag{5}$$

Note that the function $\theta(\cdot)$ returns 1 if the input is positive, otherwise 0, and $g(\cdot)$ is a function that bounds the maximum (or minimum) of weights:

$$\mathrm{g}(w) = \begin{cases} \operatorname{sgn}(w) \cdot L, & \text{for } |w| > L, \\ w, & \text{otherwise.} \end{cases} \tag{6}$$

When the parties agree to update the weight by the given learning rule, only weight values where the related hidden unit is equal to the output value are updated. Otherwise, the parties skip that round without updating weights and proceed to the next round. These procedures are repeated until the weight vectors are fully synchronized, and identical weight vectors can be used as a shared secret key.

3.2. Attack Scenarios. In the original TPM, the synchronized weight values can be used as a shared secret key. Similar to the PKC system, the goal of an attacker is to disclose the synchronized weight values of TPM. The main problem to achieve the adversarial purpose is that the internal representations $(\sigma_1, \sigma_2, \ldots, \sigma_K)$ of sender and receiver are unknown. Since the update of weights depends on hidden units, the success of attack depends on the prediction of hidden units. In order to predict hidden units accurately, various attack scenarios have been proposed [30–32]. These attack scenarios can be divided into two categories according to the resource of attacker: using single TPM and multiple TPMs.

In the case of using a single TPM, the simple attack and geometric attack are proposed. The simple attacker performs with the same structure and learning process as the two parties. In order to guess the hidden units of two the parties, the attacker uses the output value of the two parties instead of their own output value for updating weight values. However, since the simple attacker only uses public information and targets the vulnerability of the most basic property on the TPM, various high-dimensional attacks have been proposed [30–32]. The geometric attack, which performs better than the simple attack, uses the property of a local field [30]. If an absolute value of the local field is low, the two hidden units will be different, with high probability according to the geometric property of the local field. Therefore, when the output values of the participants and attacker are different, the attacker changes the hidden unit with the minimum absolute value of the local field (e.g., $|h_1^E|, |h_2^E|, \ldots, |h_K^E|$) and updates their own weight values with changed hidden units. Although the geometric attack is considered a more powerful scenario than the simple attack, it can easily be prevented by increasing the synaptic depth L of the TPM.

To overcome this limitation, the genetic attack and majority attack are proposed from the perspective of using multiple TPMs [31, 32]. The genetic attacker predicts hidden units using an evolutionary algorithm, which is different from previous approach. The attacker starts with single TPM and proceeds with the mutation step to generate multiple TPMs. In the mutation step, the attacker considers all conditions of hidden units that can occur in each round according to the output value of two participants. Consequently, genetic attacker can use up to M neural networks and can be more effective when the synaptic depth is relatively small [31]. As an efficient approach for the relatively large value of synaptic depth, the majority attack based on the geometric attack is proposed. Similar to the genetic attacker, majority attack uses an ensemble of M-TPMs and predicts hidden units by using the geometric property of local field. Initially, similar to the geometric attack, each hidden unit is changed by the minimum absolute value of local field for M-TPMs [32]. Then, the most frequently raised internal representation is selected through a majority vote in the changed hidden units, and applied to M-TPMs to update weight values. Although geometric attack can be prevented by increasing the synaptic depth L, majority attacker can perform a relatively efficient attack even if the synaptic depth is increased. Therefore, the majority attack can be considered as the most powerful attack compared to the previous attacks (i.e., simple attack, geometric attack, and genetic attack).

Based on these attack scenarios, previous studies have tried to find the optimal learning rule and the number of hidden units in terms of security and efficiency. Consequently, since other rules have limitations under certain

conditions, the random walk learning rule is generally used in many studies. In the case of the Hebbian learning rule, it can be very vulnerable to various attacks [33] even when the synaptic depth is relatively large. Conversely, when one uses the anti-Hebbian learning rule, such attacks can be resisted. However, in this case, the synchronization time increases exponentially compared to the other rules. Therefore, the random walk learning rule is widely exploited, considering the trade-off between security and efficiency. From a similar viewpoint, the number of hidden units K is usually fixed at 3. If $K = 1, 2$, it can be very vulnerable to a simple attacker who updates his/her TPM only using the output values of both parties, and if $K > 3$, the synchronization time increases exponentially [30].

Previous studies measured the security by using the probability of various attacks. Most recent studies measured the security by applying the geometric attack [21, 22]. However, the majority attack which is based on the geometric attack can be applied regardless of synaptic depth L unlike other attacks (i.e., simple attack, genetic attack, and geometric attack), and hence, it can be considered as the most powerful attack. Therefore, we apply the majority attack to measure the security and compare with the previous TPM.

4. Vector-Valued Tree Parity Machine

Although various models of TPM have been proposed, many of them are not suitable for practical use in terms of security and efficiency. For this reason, we propose a novel model of neural cryptography (called a Vector-Valued Tree Parity Machine, VVTPM) which is the generalized model of the original TPM. First, we show the architecture of VVTPM and discuss the synchronization algorithm including learning rules to update the weight vectors.

4.1. Architecture. The architecture of the Vector-Valued Tree Parity Machine (VVTPM) is shown in Figure 1. The structure of the VVTPM is similar to the existing TPM, but all internal parameters of the VVTPM are vectorized values. In our architecture, input values are defined as

$$X^k = \begin{bmatrix} \mathbb{X}_1^k \\ \mathbb{X}_2^k \\ \vdots \\ \mathbb{X}_n^k \end{bmatrix} = \begin{bmatrix} x_{11}^k & x_{12}^k & \cdots & x_{1N}^k \\ x_{21}^k & x_{22}^k & \cdots & x_{2N}^k \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1}^k & x_{n2}^k & \cdots & x_{nN}^k \end{bmatrix}, \tag{7}$$

where the index n denotes the number of vectors and $k = 1, 2, \ldots, K$ denotes the k-th hidden unit. Note that, as with the previous studies, we set the number of hidden units K at 3. If $K = 1, 2$, the simple attacker can easily synchronize and succeed the attack, and if $K > 3$, the synchronization time increases exponentially which is inefficient. The weight values which map input values to hidden units are defined as

$$W^k = \begin{bmatrix} \mathbb{W}_1^k \\ \mathbb{W}_2^k \\ \vdots \\ \mathbb{W}_n^k \end{bmatrix} = \begin{bmatrix} w_{11}^k & w_{12}^k & \cdots & w_{1N}^k \\ w_{21}^k & w_{22}^k & \cdots & w_{2N}^k \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1}^k & w_{n2}^k & \cdots & w_{nN}^k \end{bmatrix}, \tag{8}$$

where $w_{i,j}^k \in \{-L, -L+1, \ldots,\}$, $i = 1, \ldots, n$ denotes the i-th vector of weight, $j = 1, 2, \ldots, N$ denotes the j-th input value, and L is the synaptic depth of the VVTPM.

The k-th hidden unit vector σ^k is calculated as follows:

$$\boldsymbol{\sigma}^k = \begin{bmatrix} \sigma_1^k \\ \sigma_2^k \\ \vdots \\ \sigma_n^k \end{bmatrix} = \operatorname{sgn}\left(h^k\right) \triangleq \begin{bmatrix} \operatorname{sgn}\left(h_1^k\right) \\ \operatorname{sgn}\left(h_2^k\right) \\ \vdots \\ \operatorname{sgn}\left(h_n^k\right) \end{bmatrix}, \tag{9}$$

where $h^k = \begin{bmatrix} h_1^k \\ h_2^k \\ \vdots \\ h_n^k \end{bmatrix}, h_i^k = 1/\sqrt{N}\mathbb{X}_i^k \cdot \mathbb{W}_i^k = 1/\sqrt{N}\sum_{j=1}^N x_{i,j}^k \cdot w_{i,j}^k, \quad (k = 1, 2, \ldots, K,\ i = 1, 2, \ldots, n).$

Finally, the output of the VVTPM is generated as

$$\boldsymbol{\tau} = \begin{bmatrix} \tau^1 \\ \tau^2 \\ \vdots \\ \tau^n \end{bmatrix} = \boldsymbol{\sigma}^1 \odot \boldsymbol{\sigma}^2 \odot \cdots \odot \boldsymbol{\sigma}^K \triangleq \left[\prod_{k=1}^K \sigma_1^k \prod_{k=1}^K \sigma_2^k \vdots \prod_{k=1}^K \sigma_n^k \right], \tag{10}$$

where $\odot$ denotes the Hadamard product.

When the number of vector $n = 1, 2$, the VVTPM is identical to the TPM [6] and CVTPM [21], respectively. If $n = 1$, all parameters that are vectors in the VVTPM become a single variable with one integer value as a TPM. Furthermore, if $n = 2$, all parameters become vectors with two elements, and these states can interpret the real part and the imaginary part of the complex value. Therefore, in this case, the VVTPM is identical to the CVTPM. Consequently, we stress that the VVTPM can generalize structural expanded models including TPM and CVTPM.

Moreover, since each element of the output vector can be generated independently, the synchronization time of the VVTPM has the same order as the existing TPM (we will prove this in Section 5). Additionally, the VVTPM can exchange flexible-sized secret key pairs (i.e., $n \times N \times K$-sized keys) in a run of the protocol while the conventional TPM can share $K \times N$-sized secret keys. As a result, it is possible to improve security over the conventional TPM while preserving efficiency.

In addition, the VVTPM can be used in various applications. For example, since two participants can exchange the synchronized key by using VVTPM, they can create a secure channel to mutual communication (Figure 2). Instead of existing PKC, the original TPM also can be applied as the key exchange protocol and can be used with block cipher or stream cipher. Since the existing TPM takes exponentially

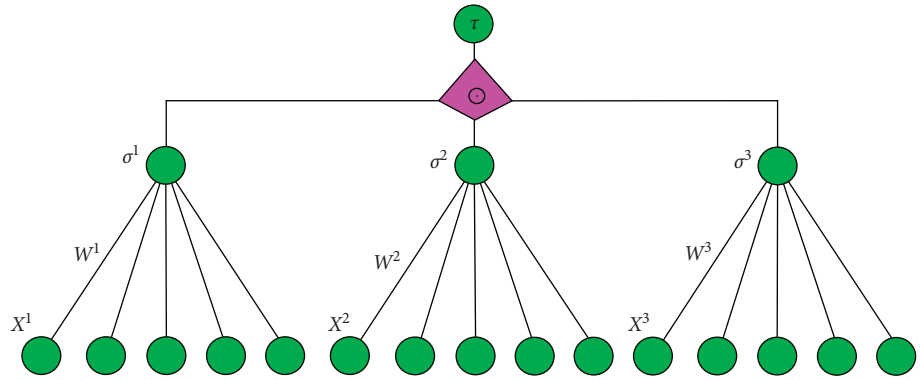

Figure 1: VVTPM architecture with $K = 3$, $N = 5$.

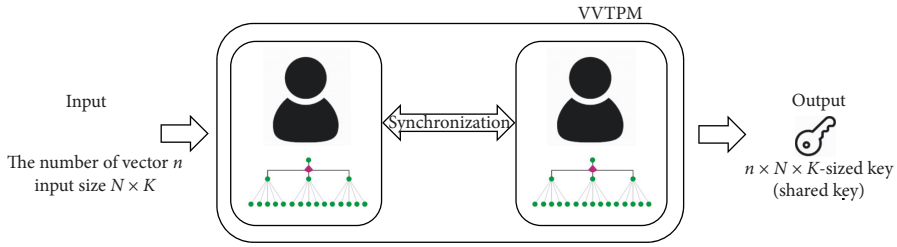

Figure 2: Architecture of sharing the key by using the VVTPM.

long time to share large keys, it is not suitable to use in real-life systems. However, the VVTPM can generate various sizes of key by adjusting the number of vector n and share a key within polynomial time.

4.2. Synchronization Algorithm. Two parties use the same VVTPM structure for synchronization, and the VVTPM can synchronize weights with Algorithm 1.

The inputs of the learning process are parameters for the VVTPM, i.e., the number of hidden units, the synaptic depth L, the number of input values for each hidden unit N, and the number of vector n. Note that W^S(or W^R) denotes the set of weight matrices (i.e., $W^S = \{W^{1^S}, W^{2^S}, \ldots, W^{K^S}\}$) of the sender S (or the receiver R). First, the sender and receiver initialize their own weight values $W^{k^{S/R}}$ to a random integer from –L to L where $k = 1, \ldots, K$. Then, public common input vectors $\mathbb{X}_i^k$ are randomly generated for all i and k, and the two parties calculate the local field vector of the hidden layer by the inner product of the input vector and the weight vector. In order to calculate the output vector, they extract sign values of the local field vector and generate the output vector of the VVTPM by the Hadamard product of hidden unit vectors. Finally, they share the own output vector in public. If the output values of the two parties are identical, they update weight vectors where $\tau^{i^{S/R}} = \sigma_i^{k^{S/R}}$ for all i and k. This process is repeated until synchronization is complete, i.e., until the sets of weight matrices W^S and W^R are identical. When the weight vectors are fully synchronized, the identical weight matrix becomes the output of the algorithm.

In the running of the algorithm, the two parties use a learning rule to update the weight vectors. There are three learning rules as follows:

(a) Hebbian learning rule:

$$\mathbb{W}_i^{k+} = g\left(\mathbb{W}_i^k + \left(\mathbb{X}_i^k\right)^T \odot \tau^{i^{S/R}} \odot \theta\left(\sigma_i^k \odot \tau^{i^{S/R}}\right) \odot \theta\left(\tau^{i^S} \odot \tau^{i^R}\right)\right). \tag{11}$$

(b) Anti-Hebbian learning rule:

$$\mathbb{W}_i^{k+} = g\left(\mathbb{W}_i^k - \left(\mathbb{X}_i^k\right)^T \odot \tau^{i^{S/R}} \odot \theta\left(\sigma_i^k \odot \tau^{i^{S/R}}\right) \odot \theta\left(\tau^{i^S} \odot \tau^{i^R}\right)\right). \tag{12}$$

(c) Random walk learning rule:

$$\mathbb{W}_i^{k+} = g\left(\mathbb{W}_i^k + \left(\mathbb{X}_i^k\right)^T \odot \theta\left(\sigma_i^k \odot \tau^{i^{S/R}}\right) \odot \theta\left(\tau^{i^S} \odot \tau^{i^R}\right)\right). \tag{13}$$

5. Analysis of Security and Efficiency

The VVTPM, which is an extended model of the TPM is expected to achieve a reasonable security level while the synchronization time has the same degree as the original TPM. To prove these improvements, we analyzed the synchronization phenomenon on both the participant-side and adversary-side. In particular, the process of performing bidirectional learning by exchanging calculated output values by two participants can be interpreted as a synchronization of the participant-side. Conversely, the process

of performing unidirectional learning by observing the exchanged output values of the two participants can be interpreted as a synchronization of the adversary-side.

5.1. Synchronization Time. To prove the synchronization time of the two participants, the overlap of internal representations (or hidden units) must be precisely recognized. However, the internal representations $(\sigma^1, \sigma^2, \ldots, \sigma^K)$ are invisible to each other, so we have to consider two main possibilities during synchronization.

Case 1. If output values for both participants are identical and each hidden unit of the participants is the same ($\tau^{i^S} = \sigma_i^{k^S} = \sigma_i^{k^R} = \tau^{i^R}$ where $i = 1, .., n$ and $k = 1, \ldots, K$), the participants update their own weight vectors $\mathbb{W}_i^{k^S}$, $\mathbb{W}_i^{k^R}$ corresponding to the hidden unit $\sigma_i^{k^{S/R}}$ in the same direction. If one of two weight values is –L or L, the distance of both weights will be decreased. Consequently, these attractive steps accelerate synchronization.

Case 2. If output values for both participants are identical ($\tau^{i^S} = \tau^{i^R}$) and each hidden unit of the participants is not the same ($\sigma_i^{k^A} \neq \sigma_i^{k^B}$), only one of the two weights will be changed. Then, when the two weight vectors were perfectly synchronized, the synchronization will be broken, except that they are adjusted by the boundary value. These repulsive steps reduce the overlap between both weight vectors, which delays synchronization.

In the case of bidirectional learning, the attractive and repulsive steps occur appropriately, and finally, perfect synchronization can be achieved. However, since the attacker synchronizes by observing the output values of the two parties, the repulsive step will occur more frequently than the attractive step. Therefore, unidirectional learning needs much more time than bidirectional learning.

The synchronization of two weight values has the same property as the two random walkers with boundary values [11]. In the case of a random walk with reflecting boundaries, the two random walkers exist within the range of 1 to d, and we can define that the initial position of the left walker is z. Moreover, the right walker starts at a distance d from the left random walker, and the two points move one by one in the same direction in each round. If either point reaches the boundary value, the distance between the two points decreases to $d - 1$. This procedure is repeated until the distance between the two points is 0, and synchronization is complete. Since the learning process of the TPM is an extension of two random walkers, full synchronization of two weight values in the TPM can be proved theoretically by using the classical gamblers' ruin problem [23].

Since the synchronization time of two random walkers depends on the boundary value, the overall synchronization time of the original TPM $\langle t_{\text{sync}} \rangle$ increases in proportion to L^2 [33]:

$$t_{\text{sync}} \propto \frac{4}{3} L^2. \tag{14}$$

Unlike bidirectional synchronization, the attacker is only allowed to synchronize by observing the input and output values. Therefore, unidirectional synchronization of attacker requires a relatively longer time than the two participants. Consequently, bidirectional synchronization increases linearly, but the synchronization time of the attacker $t_{\text{sync}}^{\text{att}}$ increases exponentially with synaptic depth L:

$$t_{\text{sync}}^{\text{att}} \propto e^{c_1 L + c_2 L^2}. \tag{15}$$

In the case of the VVTPM, each output τ^k can be calculated independently. Intuitively, if the number of vector $n = 3$, elements of the output vector τ^1, τ^2, τ^3 are calculated independently, and these calculations can be performed in parallel. In other words, the synchronization time of the VVTPM $t_{\text{sync}}^{\text{VVTPM}}$ also increases in proportion to L^2 similar to the original TPM:

$$t_{\text{sync}}^{\text{VVTPM}} \propto \max\left\{L_1^2, L_2^2, \ldots, L_n^2\right\}. \tag{16}$$

Consequently, if VVTPM and TPM have the same synaptic depth L, their synchronization time has the same order.

5.2. Security Analysis. The sender and receiver can achieve the full synchronization by exchanging the output values related to their internal representations, so the attractive step and the repulsive step occur in an appropriate proportions. However, as mentioned above, unidirectional learning takes a relatively long time to achieve synchronization. To overcome this limitation, various attack scenarios have been proposed. Among them, we consider the majority attack scenario which is the most powerful attack, to prove its security.

In all attack scenarios including majority attack, the goal of attackers is to synchronize the weighs before the two parties achieve full synchronization. In other words, we can say that the attack is successful when the synchronization time of unidirectional learning is faster than bidirectional learning. Therefore, the probability of success of an attack can be expressed as follows [33]:

$$\mathrm{P}\left(t_{\text{sync}}^{\text{att}} \leq t_{\text{sync}}\right) = \int_{t=0}^{\infty} P_{\text{sync}}^{\text{att}}(t) \frac{d}{dt} P_{\text{sync}}(t) \mathrm{d}t, \tag{17}$$

which is the probability of $t_{\text{sync}}^{\text{att}} \leq t_{\text{sync}}$ under the assumption that the two synchronization times are uncorrelated random variables. In this equation, $t_{\text{sync}}^{\text{att}}$ and t_{sync} are the synchronization time between the attacker and the two parties, and the synchronization time between the two parties. Additionally, $P_{\text{sync}}^{\text{att}}(t)$ and $P_{\text{sync}}(t)$ are the cumulative probability distribution of each synchronization time. Using the Gumbel distribution [34], equation (17) can be approximated as follows:

$$\mathrm{P}\left(t_{\text{sync}}^{\text{att}} \leq t_{\text{sync}}\right) \approx 1 - \exp\left(-\frac{t_{\text{sync}}}{t_{\text{sync}}^{\text{att}}}\right). \tag{18}$$

This means that the probability of success of an attack is proportional to the ratio of the average values of the two synchronization times which are functions of the synaptic depth L. With equations (14) and (15), the ratio of the two synchronization times, which are a function of L, can be calculated as follows:

Input K, L, N, n
(1) **Initialize** W^{k^S} (W^{k^R}) randomly for $k = 1, \ldots, K$
(2) **while** $W^S \neq W^R$ **do**
(3) **for** k **from** 1 **to** K
(4) **for** i **from** 1 **to** n
(5) generate $\mathbb{X}_i^k$ randomly
(6) **end**
(7) $h^{k^{S/R}} \longleftarrow 1/\sqrt{N} \begin{bmatrix} \mathbb{X}_1^k \cdot \mathbb{W}_1^{k^{S/R}} \\ \mathbb{X}_2^k \cdot \mathbb{W}_2^{k^{S/R}} \\ \vdots \\ \mathbb{X}_n^k \cdot \mathbb{W}_n^{k^{S/R}} \end{bmatrix}$
(8) $\sigma^{k^{S/R}} \longleftarrow \mathrm{sgn}(h^{k^{S/R}})$
(9) **end**
(10) $\tau^{S/R} \longleftarrow \sigma^{1^{S/R}} \odot \sigma^{2^{S/R}} \odot \ldots \odot \sigma^{k^{S/R}}$
(11) **for** i **from** 1 **to** n
(12) **for** k **from** 1 **to** K
(13) **if** $\tau^{i^S} = \tau^{i^R}$ **then**
(14) $\mathbb{W}_i^{k^S} \longleftarrow \mathrm{Learningrule}(\mathbb{W}_i^{k^S})$ where i satisfies $\tau^{i^S} = \sigma_i^{k^S}$
(15) $\mathbb{W}_i^{k^R} \longleftarrow \mathrm{Learningrule}(\mathbb{W}_i^{k^R})$ where i satisfies $\tau^{i^R} = \sigma_i^{k^R}$
(16) **end**
(17) **end**
(18) **end**
(19) **end**
Output W^k

Algorithm 1: Learning process of key exchange for VVTPM.

$$\frac{t_{\mathrm{sync}}^{\mathrm{att}}}{t_{\mathrm{sync}}} \propto \frac{L^2}{e^{c_1 L + c_2 L^2}}. \tag{19}$$

Consequently, the synaptic depth L is most important for the security of neural network key exchange protocols. When $L \gg 1$, the ratio of the two synchronization times becomes very small (equation (19) and the probability of success of the attacker can be approximated as

$$P\left(t_{\mathrm{sync}}^{\mathrm{att}} \leq t_{\mathrm{sync}}\right) \propto L^2 e^{-c_1 L} e^{-c_2 L^2}. \tag{20}$$

Since the probability of success of the attacker decreases exponentially as synaptic depth L increases, the two parties can adjust L to achieve the desired level of security. According to the experimental results in [33], the probability of success of a majority attack decreases exponentially in proportion to L and is approximated as follows:

$$P_{\mathrm{att}} \approx e^{-0.17(L-2.0)}, \tag{21}$$

which is based on equation (20).

In the case of the VVTPM, the attack probability of a majority attacker can be analyzed by extending the security analysis proved above. Since output vectors $\tau^1, \tau^2, \ldots, \tau^n$ are all independently calculated, the majority attacker has to use more resources than the TPM to infer each weight vectors. In other words, the attack must be performed with respect to each output vector separately, and the probability of success of the majority attacker who has to disclose all key values can be reduced in proportion to n. Since each attack divided by n is the same as the attack of the original TPM, the attack probability of a majority attack for the VVTPM is calculated as follows:

$$P_{\mathrm{att}}^{\mathrm{VVTPM}} = P_{\mathrm{att}}^{\tau^1} P_{\mathrm{att}}^{\tau^2} \cdots P_{\mathrm{att}}^{\tau^n} \approx \left(P_{\mathrm{att}}^{\mathrm{TPM}}\right)^n < P_{\mathrm{att}}^{\mathrm{TPM}}, \tag{22}$$

where $P_{\mathrm{att}}^{\tau^i}$ is the probability of success of a majority attack on each weight vector and is calculated with the same value as $P_{\mathrm{att}}^{\mathrm{TPM}}$ which is the probability of success of the original TPM.

6. Implementation

The synchronization time of the VVTPM increases in proportion to L^2 as in the original TPM, and the attack probability of a majority attack decreases exponentially in proportion to $n \times L$. To show these results experimentally, we analyzed the VVTPM under various conditions. Since the original TPM fixes the number of hidden units K at 3 and the weight values are updated by the random walk learning rule, we followed the same configurations for all experiments to compare it with the original TPM. All experimental results were measured on a computer with 3.70 GHz eighth generation Intel Core i7-8700K CPU.

To prove that the synchronization times of the VVTPM and TPM increase in proportion to the same order (L^2), we used the Frobenius norm of a matrix to show the variation of the overlap. The Frobenius norm can be defined as follows:

$$\left\| W^{k^S} - W^{k^R} \right\|_F \triangleq \sqrt{\sum_{i=1}^{n} \sum_{J=1}^{N} \left| w_{i,j}^{k^S} - w_{i,j}^{k^R} \right|^2}, \quad \text{for } k = 1, \ldots, K. \tag{23}$$

When the two weight vectors are fully synchronized, the Frobenius norm decreases to 0. To compare TPM, CVTPM, and VVTPM, without loss of generality, we set the number of vectors as $n = 1, 2$, and 5. Figure 3 shows the relationship

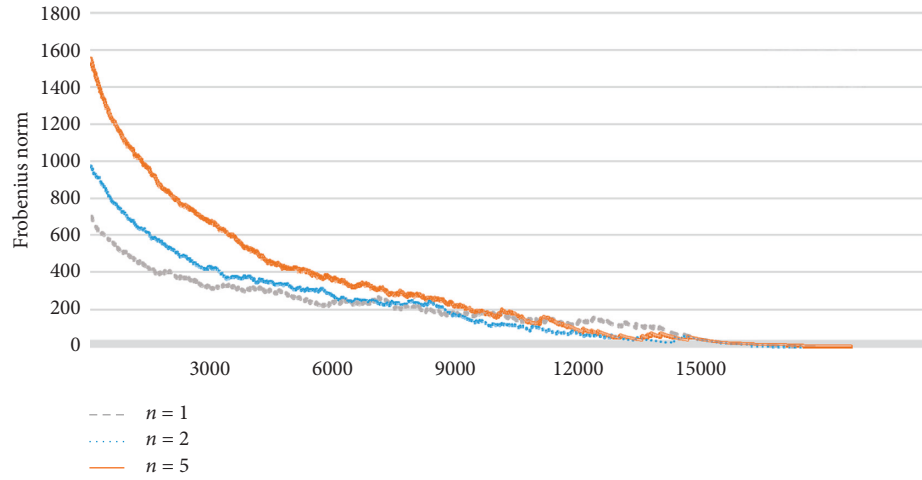

Figure 3: Frobenius norm between the weight matrix of TPM, CVTPM, and VVTPM.

between the number of iterations and Frobenius norm. Each line represents one synchronization process, and the repeated experimental results all exhibit a similar pattern, so only one result is shown for each n values. At the beginning of synchronization, since the VVTPM has a larger number of weight values than other models, the starting values of the Frobenius norm can be higher. However, during synchronization, the flow of the decreasing Frobenius norm proceeds similarly. As a result, we verified that the synchronization time has the same order regardless of internal structure.

In order to verify a more reliable process of synchronization, we also measured the time of synchronization of the TPM, CVTPM, and VVTPM based on these results. To obtain more fair results in the measurement of synchronization, we measured not the time but the number of rounds that occurred up to full synchronization for both users using each model with $K = 3, N = 1000$.

Figure 4 shows a graph showing the number of rounds that occurred up to full synchronization for both users using VVTPM. Experiments were performed for three values of synaptic depth, and each point represents the average value of the experimental results repeated 10,000 times. When comparing the original TPM ($n = 1$), CVTPM ($n = 2$), and VVTPM, the average number of rounds does not make a difference despite the increase in the value of n. This shows a similar aspect not only in the average value, but also in the median value (see Table 1). Since each vector value can be calculated independently, the number of vector n does not affect the synchronization time. Consequently, the synchronization time of the VVTPM has the same degree as the original TPM regardless of the number of vector n.

Most recent studies have conducted experiments using the geometric attack to measure the security level [18, 19]. However, the majority attack, which is improved from the geometric attack, is the most powerful attack against the original TPM. Therefore, in order to verify a rigorous security level experimentally, we measured the probability of success of a majority attack on the VVTPM.

In the majority attack scenario, the condition of success is for the attacker to find out all weight values before the participants are fully synchronized. In other words, if the attacker's unidirectional learning finishes synchronization earlier than the bidirectional learning between the participants, it is considered that the attack is successful. However, the participants only exchange common inputs and calculated outputs, and it is difficult to determine when full synchronization is achieved. Therefore, an attack is defined as successful if the attacker achieves synchronization of 98% or more of the weights when the weights of both participants are exactly the same.

In previous studies, experiments were conducted by setting the target security level to 10^{-4}, and accordingly, we also set 10^{-4} to conduct experiments. Moreover, we verified the possibility that the security of the VVTPM can be improved to the reasonable security level required in real-life systems.

Figure 5 shows a graph, showing the probability of success of a majority attacker according to the size of vector n. Similar to synchronization time, we set the parameter $K = 3, N = 1000$ and the number of networks for the majority attacker $M = 100$. Furthermore, we only used the random walk learning rule and each point represents the attack probability among a total of 10,000 attacks. As mentioned above, the success of an attack is defined as when the attacker achieves synchronization of 98% or more of the weights when both participants achieve full synchronization. The attack probability P_E decreases as the number of vectors increases, as shown in equation (18). Also, confirming the value in Table 2, when $L = 15$, the original TPM($n = 1$) has an attack probability of 3%, but a VVTPM with n of 3 or more has an attack probability of 10^{-4} which is the target security level. When we use the original TPM with $L = 57$, we can achieve the target security level 10^{-4}. However, if we set $L = 57$, a very long time is required for complete synchronization, and it is practically impossible to use in real-life systems. As a result of the experiment, when the case of $L = 40$, it takes about

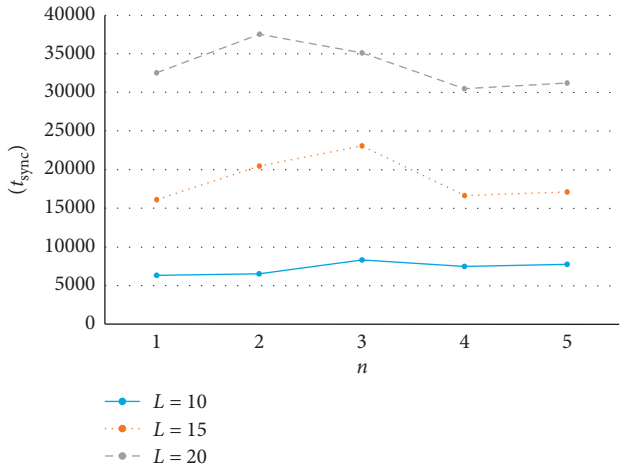

Figure 4: The number of rounds that occurred before full synchronization according to the number of vectors. From the top, each line means the average value of a total of 10,000 experiments for synaptic depth $L = 10$, 15, and 20.

Table 1: The average and median values for the number of rounds until full synchronization.

Number of vectors	$L = 10$		$L = 15$		$L = 20$	
	Median	Average	Median	Average	Median	Average
$n = 1$	6317	6625	16130	16814	32558	33781
$n = 2$	6544	6811	20462	21184	37575	42052
$n = 3$	8332	8688	23095	24039	35114	35655
$n = 4$	7512	7766	16651	17047	30506	31088
$n = 5$	7780	8016	17090	17476	31281	31905

1,270,000 rounds and about 2 minutes to complete synchronization. On the other hand, when we use the VVTPM with $n = 3, L = 15$, it takes only about 24,000 rounds and about 2 seconds. In other words, we can achieve synchronization in an incomparably faster time than the original TPM. Moreover, when the participants use the VVTPM, a reasonable security level can be achieved in real-life systems by increasing parameters n and L.

Similar to using the random walk learning rule, the experimental results can be applied to the rest of the learning rules (i.e., Hebbian learning rule and anti-Hebbian learning rule). Figure 6 shows a graph that shows the probability of success of a majority attacker according to the learning rules. Similar to previous results, the VVTPM can increase security in an attack scenario regardless of learning rule. In particular, when we apply the Hebbian learning rule to the original TPM, it can be very vulnerable to various attacks even when the synaptic depth is relatively large. However, in the case of the VVTPM, we can increase security even with the Hebbian learning rule by varying the number of vectors. Therefore, we can perform secure key exchange with the Hebbian learning rule. On the contrary, when we apply the anti-Hebbian learning rule, we can exchange the key securely against various attacks regardless of models (e.g., TPM and VVTPM). However, the anti-Hebbian learning rule still has limitations in terms of efficiency and is not suitable to apply to real-life systems.

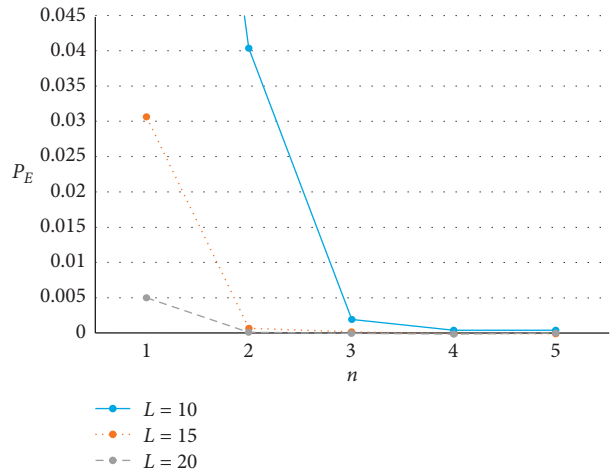

Figure 5: The probability of success of a majority attack with $K = 3$, $N = 1000$, and $M = 100$. We used the random walk learning rule to update the weight vectors and measure the values of the result with 10,000 repeated experiments.

Table 2: The probability of success of a majority attack with **K** = 3, **N** = 1000, and **M** = 100. (The value 0.0000 means that the attack was not successful in the entire 10,000 attack experiments).

Number of vectors	$L = 10$	$L = 15$	$L = 20$
$n = 1$	0.1377	0.0306	0.0050
$n = 2$	0.0403	0.0007	0.0001
$n = 3$	0.0019	0.0001	0.0000
$n = 4$	0.0004	0.0000	0.0000
$n = 5$	0.0003	0.0000	0.0000

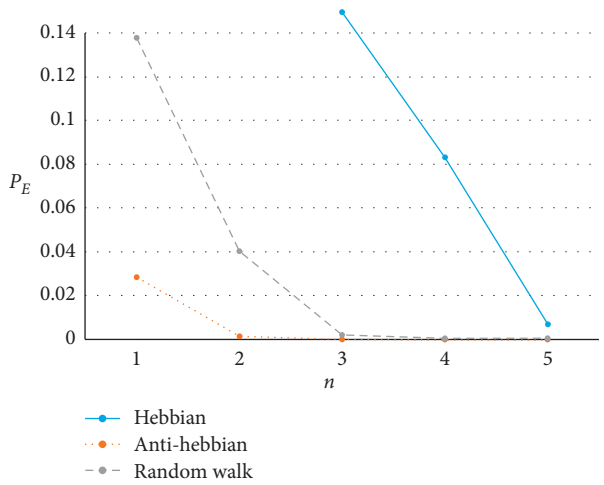

Figure 6: The probability of success of a majority attack with $K = 3$, $N = 1000$, $L = 10$, and $M = 100$ according to the learning rules. We measured the values of the result with 10,000 repeated experiments.

7. Conclusions

In this paper, we proposed a novel architecture of neural cryptography, called Vector-Valued Tree Parity Machine (VVTPM), which can be applied to generate a flexible length of secret key. In addition, the VVTPM can generalize the extended model in terms of reconstructing model architecture, including the original TPM. In particular, it is not only a generalized model, but it can also increase security while preserving efficiency. By varying the number of vectors, the VVTPM can achieve the reasonable security level required in real-life systems. To verify the improvement of security, we theoretically analyzed the process of synchronization in terms of both bidirectional learning and unidirectional learning. Then, we showed that the synchronization time of the VVTPM has the same order as the existing TPM and proved that the security of the VVTPM can be increased with the same synaptic depth while preserving the synchronization time.

In our experiment, we applied the most powerful attacker that has not been considered in recent work and set the target security level to 10^{-4}, which has been considered in previous results. In addition, we showed that the VVTPM can achieve the higher level of security required in real-life systems as well as the previously considered security level by varying the number of vectors. Moreover, to verify that the synchronization time of the VVTPM has the same order as the original TPM, we measured the number of rounds for full synchronization. As a result, we showed that the number of vectors cannot affect the synchronization time of the VVTPM and that the security level against the most powerful attack can be controlled by varying the number of vectors and synaptic depth. Additionally, we verified that the improvement of security in the VVTPM can be preserved regardless of learning rule.

Similar to original TPM, the VVPM can be applied in many applications. Especially, it can be utilized to symmetric cryptosystems (symmetric key generation) and stream cipher systems (seed value) instead of the existing PKC. Moreover, it would be interesting to analyze the effectiveness of our model with other cryptosystems and to compare the performance of neural cryptography with the existing key exchange algorithms (e.g., PKC) in various environments.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (2019R1A2C1003146) and Electronics and Telecommunications Research Institute (ETRI) funded by the Korean Government (21ZR1300, Core Technology Research on Trust Data Connectome).

References

[1] M. E. Hellman, "An overview of public key cryptography," *IEEE Communications Magazine*, vol. 40, no. 5, pp. 42–49, 2002.

[2] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[3] ITU-T, *Recommendation X.509, Information Technology–Open Systems Interconnection–The Directory: Public-Key and Attribute Certificate Frameworks*, ITU, Geneva, Switzerland, 2005.

[4] M. S. Anoop, "Security needs in embedded systems," *IACR Cryptology ePrint Archive*, vol. 198, p. 2008, 2008.

[5] N. Koblitz, *A Course in Number Theory and Cryptography*, Springer Science & Business Media, Berlin, Germany, 1994.

[6] R. Mislovaty, Y. Perchenok, I. Kanter, and W. Kinzel, "Secure key-exchange protocol with an absence of injective functions," *Physical Review E*, vol. 66, no. 6, Article ID 066102, 2002.

[7] M. Rosen-Zvi, E. Klein, I. Kanter, and W. Kinzel, "Mutual learning in a tree parity machine and its application to cryptography," *Physical Review E*, vol. 66, no. 6, Article ID 066135, 2002.

[8] I. Kanter, W. Kinzel, and E. Kanter, "Secure exchange of information by synchronization of neural networks," *Europhysics Letters (EPL)*, vol. 57, no. 1, p. 141, 2002.
[9] R. Mislovaty, E. Klein, I. Kanter, and W. Kinzel, "Public channel cryptography by synchronization of neural networks and chaotic maps," *Physical Review Letters*, vol. 91, no. 11, Article ID 118701, 2003.
[10] A. Ruttor, W. Kinzel, and I. Kanter, "Dynamics of neural cryptography," *Physical Review E*, vol. 75, no. 5, Article ID 056104, 2007.
[11] A. Ruttor, G. Reents, and W. Kinzel, "Synchronization of random walks with reflecting boundaries," *Journal of Physics A: Mathematical and General*, vol. 37, no. 36, p. 8609, 2004.
[12] J. Daemen and V. Rijmen, "AES proposal: rijndael," 1999.
[13] E. F. Schaefer, "A simplified data encryption standard algorithm," *Cryptologia*, vol. 20, no. 1, pp. 77–84, 1996.
[14] M. Murase, "Linear feedback shift register," U.S. Patent NO. 5090035, 1992.
[15] A. Ruttor, W. Kinzel, L. Shacham, and I. Kanter, "Neural cryptography with feedback," *Physical Review E*, vol. 69, no. 4, Article ID 046110, 2004.
[16] A. Ruttor, W. Kinzel, and I. Kanter, "Neural cryptography with queries," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2005, Article ID P01009, 1 page, 2005.
[17] A. M. Allam and H. M. Abbas, "On the improvement of neural cryptography using erroneous transmitted information with error prediction," *IEEE Transactions on Neural Networks*, vol. 21, no. 12, pp. 1915–1924, 2010.
[18] A. M. Allam and H. M. Abbas, "Improved security of neural cryptography using don't-trust-my-partner and error prediction," in *Proceedings of the 2009 International Joint Conference on Neural Networks*, IEEE, Atlanta, GA, USA, June 2009.
[19] N. Mu, X. Liao, and T. Huang, "Approach to design neural cryptography: a generalized architecture and a heuristic rule," *Physical Review E*, vol. 87, no. 6, Article ID 062804, 2013.
[20] X. Lei, X. Liao, F. Chen, and T. Huang, "Two-layer tree-connected feed-forward neural network model for neural cryptography," *Physical Review E*, vol. 87, no. 3, Article ID 032811, 2013.
[21] T. Dong and T. Huang, "Neural cryptography based on complex-valued neural network," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 11, 2019.
[22] E.`. Salguero Dorokhin, W. Fuertes, and E. Lascano, "On the development of an optimal structure of tree parity machine for the establishment of a cryptographic key," *Security and Communication Networks*, vol. 2019, Article ID 8214681, 10 pages, 2019.
[23] W. Feller, *An Introduction to Probability Theory and Its Applications*, John Wiley & Sons, Hoboken, NJ, USA, 2008.
[24] S. Santhanalakshmi, K. Sangeeta, and G. K. Patra, "Design of group key agreement protocol using neural key synchronization," *Journal of Interdisciplinary Mathematics*, vol. 23, no. 2, pp. 435–451, 2020.
[25] A. M. Allam and H. M. Abbas, "Group key exchange using neural cryptography with binary trees," in *Proceedings of the 2011 24th Canadian Conference on Electrical and Computer Engineering (CCECE)*, IEEE, Ontario, Canada, April 2011.
[26] M. Volkmer, "Entity authentication and authenticated key exchange with tree parity machines," *IACR Cryptology ePrint Archive*, vol. 112, p. 2006, 2006.
[27] A. M. Allam, H. M. Abbas, and M. Watheq El-Kharashi, "Authenticated key exchange protocol using neural cryptography with secret boundaries," in *Proceedings of the The 2013 International Joint Conference on Neural Networks (IJCNN)*IEEE, Dallas, TX, USA, August 2013.
[28] T. Chen, D. Yan, and S. Bai, "TinyTPM: a novel lightweight key agreement scheme for sensor networks," *WRI International Conference on Communications and Mobile Computing*, vol. 3, 2009.
[29] M. Volkmer and S. Wallner, "Tree parity machine rekeying architectures," *IEEE Transactions on Computers*, vol. 54, no. 4, pp. 421–427, 2005.
[30] A. Kilmov, A. Mityagin, and A. Shamir, "Analysis of neural cryptography," *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, Berlin, Heidelberg, 2002.
[31] A. Ruttor, W. Kinzel, R. Naeh, and I. Kanter, "Genetic attack on neural cryptography," *Physical Review E*, vol. 73, no. 3, Article ID 036121, 2006.
[32] L. N. Shacham, E. Klein, R. Mislovaty, I. Kanter, and W. Kinzel, "Cooperating attackers in neural cryptography," *Physical Review E*, vol. 69, no. 6, Article ID 066137, 2004.
[33] A. Ruttor, "Neural synchronization and cryptography," 2007, https://arxiv.org/abs/0711.2411.
[34] E. J. Gumbel, "Les valeurs extrêmes des distributions statistiques," *Annales de l'institut Henri Poincaré*, vol. 5, no. 2, 1935.