Received 14 March 2023, accepted 1 May 2023, date of publication 4 May 2023, date of current version 12 May 2023.
Digital Object Identifier 10.1109/ACCESS.2023.3273011

RESEARCH ARTICLE

Improvement of the Efficiency of Neural Cryptography-Based Secret Key Exchange Algorithm

JUYOUNG KIM[1], SOOYONG JEONG[2], DOWON HONG[3], AND NAM-SU JHO[1]
[1]Cyber Security Research Division, Electronics and the Telecommunications Research Institute, Daejeon 34129, Republic of Korea
[2]Department of Convergence Science, Kongju National University, Gongju 32588, Republic of Korea
[3]Department of Mathematics, Kongju National University, Gongju 32588, Republic of Korea

Corresponding author: Nam-Su Jho (nsjho@etri.re.kr)

This work was supported by the Electronics and Telecommunications Research Institute (ETRI) Grant funded by Korean Government (Core Technology Research on Trust Data Connectome) under Grant 20ZR1300.

ABSTRACT Owing to new security threats and relevant changes in network computing environments, key exchange methods that replace conventional public key exchange algorithms are being researched. The neural cryptography-based key exchange algorithm proposed recently uses neural synchronization as an alternative to public-key methods. However, the learning-based synchronization method uses considerable communication resources to generate output values and share the results. The efficiency of this method depends on the type of algorithm and is affected by both the communication rounds for exchanging output values and the number of weight learning rounds. To improve its efficiency, this paper proposes 1-h random walk and batch scheme methods and verifies their efficiency and security.

INDEX TERMS Random walk, batch scheme methods, key exchange methods, neural cryptography-based key exchange algorithm.

I. INTRODUCTION

The Rivest–Shamir–Adelman (RSA) and Diffie–Hellman (DH) algorithms are the most widely used public key exchange algorithms that use mathematical complexity, such as the prime factorization of very large numbers or discrete logarithm problems, to secure data transmission [1]. With recent developments in computing performance, the time required for prime factorization has decreased. In 2009, Kleinjung et al. succeeded in solving the prime factorization of an RSA-786 key [2]. Hence, RSA-2048 and higher numbers have been recommended. Unfortunately, this has increased the resources required for public key exchange. Likewise, larger prime numbers are now required to ensure the security of the discrete logarithm problem, which is the basis of the DH algorithm. However, this also increases computing resource requirements while reducing efficiency.

The associate editor coordinating the review of this manuscript and approving it for publication was Junggab Son.

Furthermore, with the imminent commercialization of quantum computers, the vulnerabilities of conventional key exchange algorithms based on prime factorization will become insurmountable. Conventional prime factorization takes exponential time, but the possibility of prime factorization within polynomial time using Shor's algorithm in a quantum-computer environment has been verified [3]. To respond to quantum computing, the DH algorithm has been improved via the introduction of new models, such as the supersingular isogeny Diffie-Hellman (SIDH) method [3]. However, this also requires vast improvements to contend with quantum capabilities.

The neural cryptography-based key exchange algorithm [4] has been researched extensively as it provides an alternative to conventional public key exchange algorithms [5], [6], [7]. It generates keys by sharing and synchronizing the same neural network. The working principle of the neural cryptography-based key exchange algorithm includes the generation of a random weight by two users. Next, the users generate an output value using the same input value randomly generated and their respective weights. Then, the weight is changed via machine learning according to the shared output values. This is repeated until the weights of the two neural networks are synchronized. Finally, a key is generated using the synchronized weight.

The neural network used in this study is a tree parity machine (TPM), which has a neural network structure with k hidden layers and n inputs per hidden layer, and it generates binary output values. The neural-network learning methods of TPM include Hebbian, anti-Hebbian, and Random-walk [8]. Among these, the random walk learning method is known to have the highest security [9], [10]. Because the neural cryptography-based key exchange algorithm does not depend on the difficulty of a specific mathematical problem, it has the advantage of maintaining the security of the exchanged key, even when the difficulty of a specific problem is neutralized by a new attack method or a quantum computer.

However, the learning-based synchronization method uses considerable communication resources to generate output values and share the results, and the communication rounds for exchanging output values as well as the number of weight learning rounds has a significant effect on the efficiency of the proposed algorithm. Therefore, weight learning and communication rounds must be reduced to increase the efficiency of the neural cryptography-based key exchange algorithm.

In particular, conventional learning, wherein the change in weight is fixed at ± 1, requires a number of learning iterations that are too large to be practical. Therefore, this paper proposes a 1-h random-walk learning method that sets the change of weight differently depending on conditions improve the efficiency of the learning rules. The 1-h random-walk learning method enables key exchange efficiency at a level that is applicable to practical communication environments. Moreover, shifting and changing random number (SCRN) and error correction code (ECC) methods are proposed to improve the efficiency of the batch scheme and its algorithm, which collects and transmits learning results to achieve higher efficiency.

Through experiments, this study demonstrates that the proposed methods have higher efficiency while maintaining the same levels of security as the baseline neural network encryption-based key exchange algorithm.

The remainder of this paper is organized as follows. Section II describes the neural cryptography-based key exchange algorithm. Section III describes the proposed 1-h random-walk learning and batch schemes. Section IV describes the implementation of the proposed TPM model, including SCRN and ECC. Section V analyzes the security and performance of the proposed model. Finally, conclusions are presented in Section VI.

II. BACKGROUND

TPM, which is used for neural cryptography-based key exchange, comprises a weight ($w_{i,j}$), a hidden unit (σ_i), and an output value (τ), as shown in Figure 1, where three hidden units ($K = 3$) and a TPM structure with 1,000 weights ($N = 1000$) per hidden unit are shown. The total number of weights is $K \times N = 3,000$. The input value used in learning, $x_{i,j}$ is a random value newly generated whenever learning is performed.

TABLE 1. Definition of Symbols.

Symbol	Description
K	Number of hidden units
L	Maximum weight
N	Number of weights
MAX	Maximum number of continuous learning rounds
w	Weight
τ	Output value
σ	Hidden unit
x	Input value
hop	Weight change
ECC_K	Total message length of ECC block
ECC_N	Original message length of ECC block
ECC_T	Maximum number of correctable errors of ECC block
B	Batch value
p	Participant identification number: 0 or 1

Each TPM initializes the random weight first and shares the input value at every learning round. The output value, τ, is generated using the input value and weight and is shared with other TPMs. The weight is learned if τ matches; otherwise, learning is not performed. This process is repeated until the weights are synchronized. See Table 1 for definitions of symbols.

The learning-based neural cryptography-based key exchange algorithm is described in detail as follows. Two participants in the key exchange set up TPM(0) and TPM(1), respectively, using the initially shared parameters. They generate random weights $w^{(0)}{}_{i,j}$ and $w^{(1)}{}_{i,j}$ using a pseudo-random function, and the component element range is $-L \leq w^{(p)}{}_{i,j} \leq L$. Additionally, the two TPMs share a seed for generating the same input value, $x_{i,j}$, to generate $x_{i,j}$ using the shared seed at each learning round. However, $x_{i,j}$ is 1 or -1, and the two TPMs use the same $x_{i,j}$ during the same learning round.

Next, for $p = 0$ or 1, TPM(p) consecutively adds $x_{i,j}$ multiplied by $w^{(p)}{}_{i,j}$ to obtain $\sigma^{(p)}{}_i$. If $\sigma^{(p)}{}_i$ is larger than zero, it is replaced with one; if it is smaller than zero, it is replaced with -1. This can be expressed as follows:

$$\sigma_i^{(p)} = sgn(\sum_{j=1}^{N} w_{i,j}^{(p)} x_{i,j}) \tag{1}$$

The generated $\sigma^{(p)}{}_1\sigma^{(p)}{}_2 \ldots \sigma^{(p)}{}_K$ is multiplied to obtain $\tau^{(p)}$. TPM$^{(0)}$ and TPM$^{(1)}$ update the weight by learning if the values generated by sharing $\tau^{(0)}$ and $\tau^{(1)}$ match. For the weight-learning method, Hebbian, anti-Hebbian, and random-walk learning methods are widely used. Among them, random-walk is known to be more practical and secure than Hebbian and anti-Hebbian methods [9]. With random-walk learning, $x_{i,j}$ is added to the $w_{i,j}^{(p)}$ of the hidden unit, where $\tau^{(p)}$ and $\sigma^{(p)}{}_i$ match. The function $g((w)$ replaces the learned $w^{(p)}{}_{i,j}$ with $-L$ if it is smaller than $-L$ or with L if it is larger than L to keep $w_{i,j}^{(p)}$ in the range of $-L \leq w_{i,j}^{(p)} \leq L$.

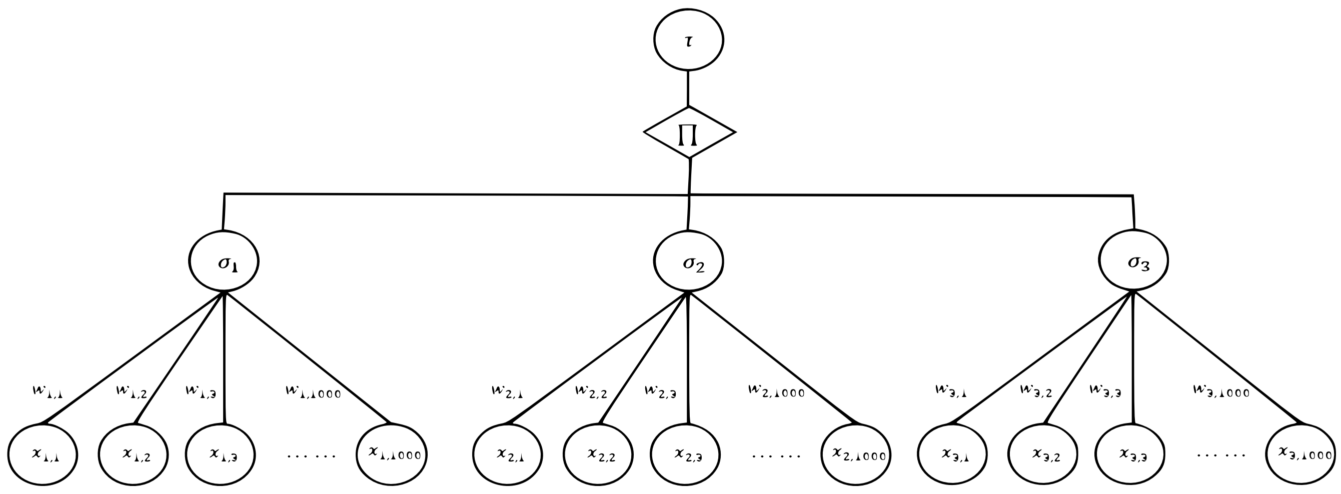

FIGURE 1. **TPM structure with $K = 3$ and $N = 1,000$.**

The function, $\Theta(x)$, replaces $\sigma^{(p)}{}_i$ with one if it is larger than zero, and with 0 if it is smaller or equal than 0. This can be expressed as follows:

$$w_{i,j}^{+} = g(w_{i,j}^{(p)} + x_{i,j}\Theta(\sigma_i^{(p)}\tau^{(p)})\Theta(\tau^{(0)}\tau^{(1)}))$$
$$g(w) = \begin{cases} sgn(w) \times L & \text{for } |w| > L \\ w & \text{otherwise} \end{cases}$$
$$\Theta(x) = \begin{cases} 0 & x < 0 \\ 1 & \text{otherwise} \end{cases} \tag{2}$$

To determine whether synchronization is completed, the two TPMs agree in advance on the parameter MAX for ending learning. Initially, the end counter is set to zero and is increased in each learning round if $\tau^{(0)}$ and $\tau^{(1)}$ match, and learning is performed. If the shared $\tau^{(0)}$ and $\tau^{(1)}$ do not match, the end counter is reset to zero. When the end counter reaches MAX, learning is completed, considering that weight synchronization has been completed. Finally, TPM(p) generates the secret key using the synchronized $w^{(p)}$. In random-walk learning, parameters L and N are important elements that influence the security and efficiency of the TPM. If L and N are large, security increases, but more computer resources are required because the number of learning rounds required for synchronization and the number of communications used to replace τ increase. If L and N are small, security problems can occur because attackers can easily infer the weight of each TPM. Various attack methods for neural synchronization-based key exchange algorithms have also been studied [4], [10]. Among them, the most effective attack method for random-walk learning is the majority attack, which uses the geometric properties of the input value and weight vectors. The attacker sets up M neural networks according to the initially set M value and attempts to synchronize with one TPM using the input value and the message between legitimate participants [7]. A previous study verified that the success probability of a majority attack can be decreased to 10-4, but it was not practical because the parameters used in the experiment were $K = 3, N = 1,000$, and $L = 57$, and the average number of learning rounds required was 1.6×10^5 [9], [10]. This suggests that synchronizing the weight while maintaining a certain level of security has practical problems owing to the use of excessive resources. Therefore, the algorithm must be improved to reduce the number of learning and communication rounds while maintaining security. To address this issue, we describe an improved neural cryptography-based key exchange algorithm in the next section.

III. IMPROVED TREE PARITY MACHINE MODEL

A. 1-H RANDOM-WALK LEARNING

To maintain the security and efficacy of random-walk learning, it is necessary to choose an appropriate L value [11]. A larger L increases efficacy, but it requires many learning rounds. In particular, according to the random-walk learning rule, the change in weight is fixed at ± 1 . Hence, if L is large, a large amount of learning is required to reduce the difference in the early learning process, in which the difference in weight between two users is large. Therefore, if the amount of weight changes, learning can be performed more efficiently than with the conventional random-walk method. However, if the same weight change is set for each round, an effect similar to reducing L in proportion to the weight change appears, and this can interfere with synchronization at a stage where synchronization has progressed significantly. Therefore, we propose a 1-h random-walk learning algorithm to improve efficiency while maintaining security by variably setting the weight change depending on the situation. The weight change of the conventional random-walk learning algorithm is fixed at ± 1 , but the weight change of the 1-h random-walk learning algorithm alternates between ± 1 and $\pm hop$. Thus, if $\pm hop$ is changed, the weight is changed as

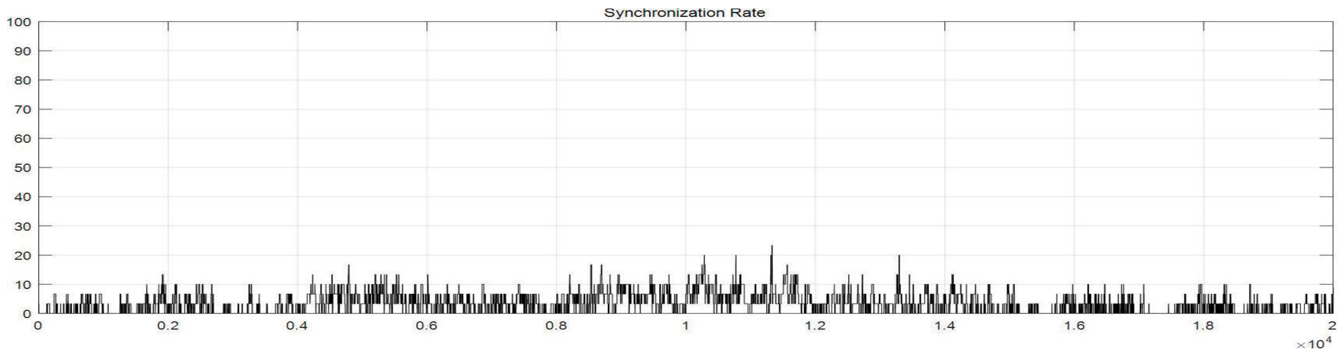

FIGURE 2. **Random-walks learning ($K = 3, N = 10, L = 57, MAX = 3, 500$).**

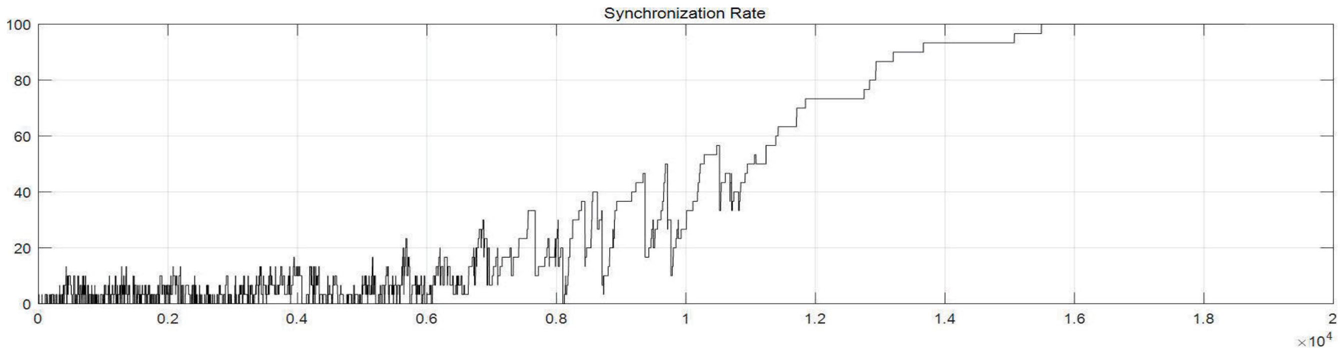

FIGURE 3. **1–3 Random-walks learning ($hop = 3, N = 10, L = 57, MAX = 3, 500$).**

follows:

$$
\begin{aligned}
w_{i,j}^{+} &= g(w_{i,j}^{(p)} + hop \times x_{i,j}\Theta(\sigma_i^{(p)}\tau^{(p)})\Theta(\tau^{(0)}\tau^{(1)})) \\
g(w) &= \begin{cases} sgn(w) \times L & \text{for } |w| > L \\ w & \text{otherwise} \end{cases} \\
\Theta(x) &= \begin{cases} 0 & x < 0 \\ 1 & \text{otherwise} \end{cases}
\end{aligned} \tag{3}
$$

To verify the efficiency of the 1-h random-walk learning, we compare the number of learning rounds required for weight synchronization with conventional random-walk learning. To adjust the weight change, the weight is set to ± 3 for even numbers and to ± 1 for odd numbers. Figure 2 and figure 3 shows the weight match rate for each learning stage of conventional and 1-h random-walk learning methods. The parameters for this experiment are set to $K = 3, N = 10, L = 57$, and $MAX = 3, 500$. In conventional random-walk learning, the weight synchronization rate is approximately 10%, even when the number of learning rounds is 20,000. However, in 1–3 random-walk learning, the weight synchronization rate becomes 100% when the number of learning rounds reaches 15,490. This experiment shows that the 1-h random-walk learning method is more efficient in weight synchronization than the conventional random-walk learning method. Analysis according to hop value is described in Section V-A.

B. BATCH SCHEME

The neural cryptography-based key exchange algorithm continuously exchanges the output value. Hence, the communication time required for output value exchange can become longer than the learning time, depending on the communication environment in the real-world environment. In particular, owing to the nature of the neural cryptography-based key exchange algorithm, which performs learning by receiving the output value of the other party, network traffic is frequently generated because it must wait until the other party transmits the output value and determines whether to update the output value by continuously sharing the output value.

This increases the time required for random-walk learning depending on the communication environment. Although the number of learning rounds is decreased by the 1-h random walk algorithm, more than 18,000 communications are required at $K = 3, L = 57, N = 10$, and $hop = 3$. Therefore, if the total number of communication rounds is reduced, the efficiency can be improved by reducing the time required for the neural cryptography-based key exchange algorithm.

To this end, we design a batch scheme, in which the conventional algorithm shares the generated τ values during each learning round and performs weight learning depending on whether they match. However, the batch scheme generates τ values as many as the previously defined parameters (B) in advance, shares them with each other, and performs learning only when they match.

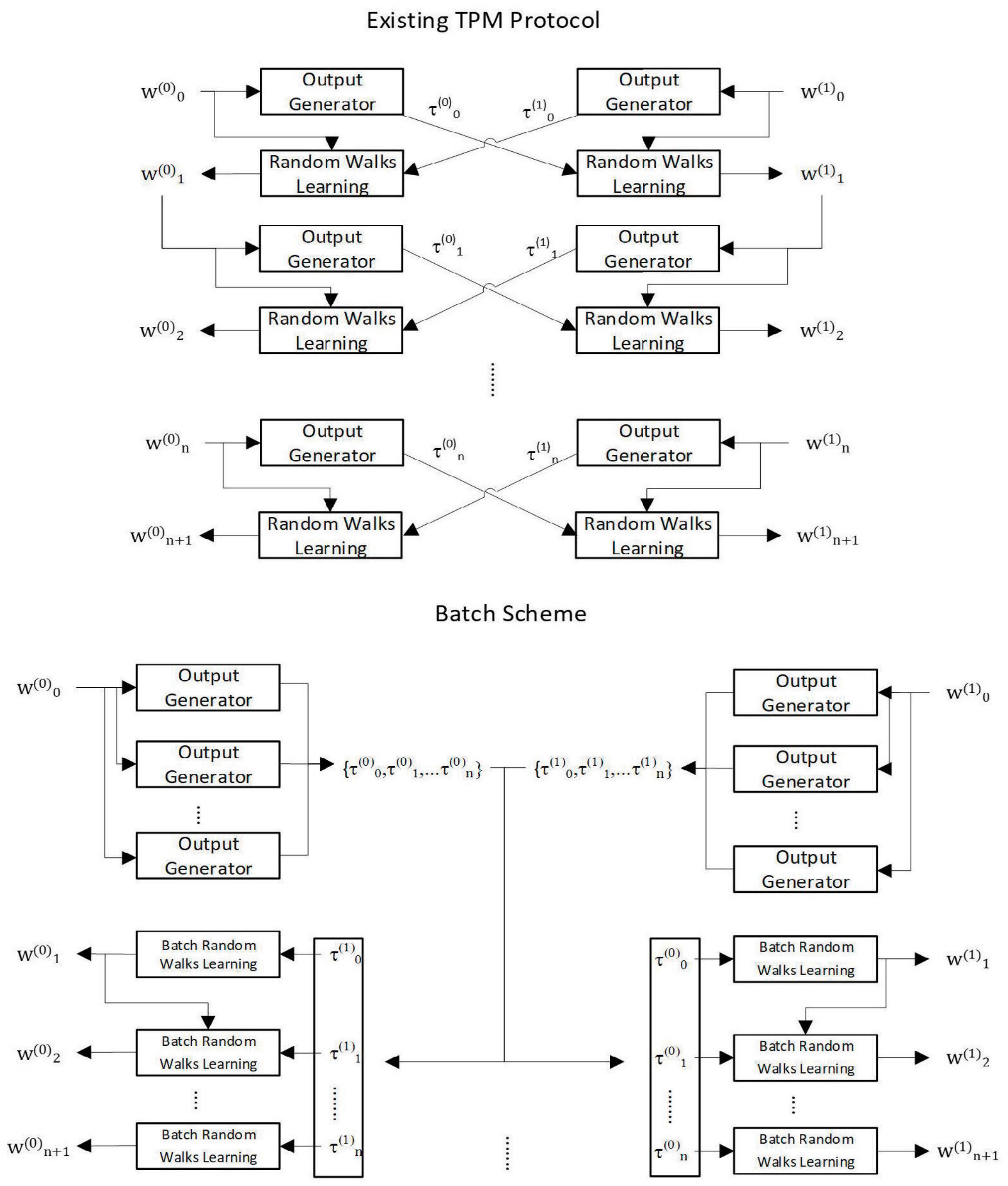

FIGURE 4. Comparison of batch scheme and existing TPM protocol.

For example, if the batch value is $B = 10$, $\tau_1 \ldots \tau_{10}$ are generated and are shared with each other. Then, learning is performed in the same manner as in the conventional learning method by sequentially comparing τ_i.

The conventional random-walk learning algorithm increases the end counter by one if the exchanged τ values match and terminates learning if the preset end counter value is reached. If the τ values do not match, the end counter is reset to zero. Because the batch scheme sends and compares a set of τ, the end counter is increased by one only when the set of τ matches, and it is reset to zero if any one element of the set does not match. In other words, the conventional algorithm sends only a single τ, but the batch scheme sends a set.

Figure 4 shows the difference between the conventional TPM algorithm and that with the batch scheme. The conventional TPM algorithm exchanges τ one by one during the learning stage, but the batch scheme exchanges τ as many times as the set batch number.

The following figures 5 and figures 6 compare the weight match rate in each communication round of the conventional

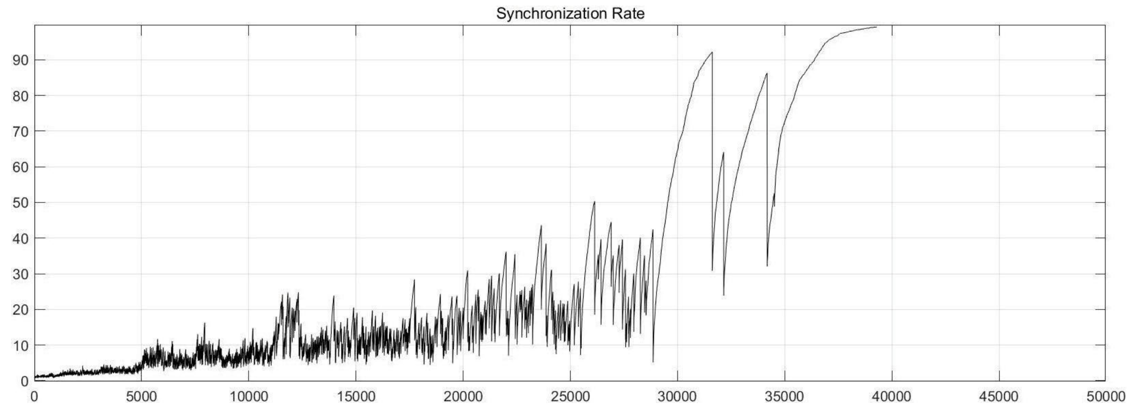

FIGURE 5. 1-h random-walks learning($K = 3, N = 1,000, L = 57, MAX = 200$).

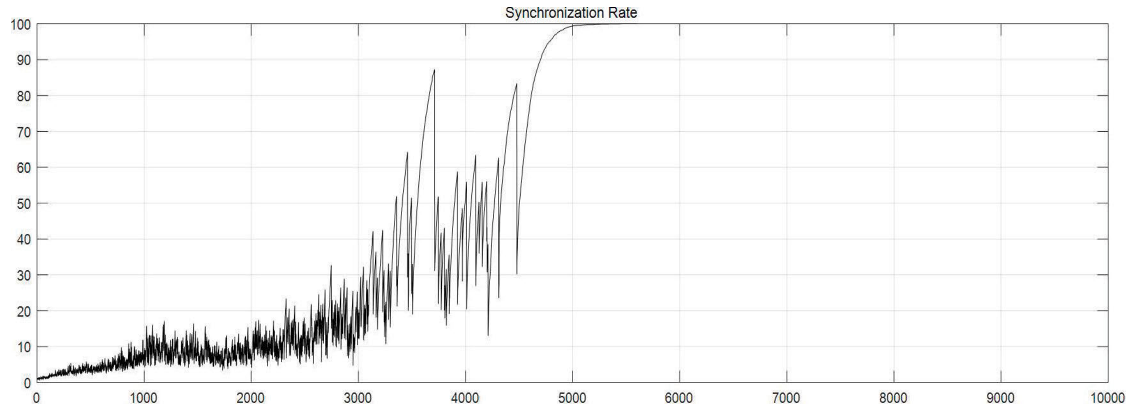

FIGURE 6. Batch 1-h random-walks learning ($K = 3, N = 1,000, L = 57, hop = 3, MAX = 200, B = 10$).

1-h random-walk algorithm and the batch-scheme 1-h random walk algorithm. The parameters used in this experiment are $K = 3$, $N = 1,000$, $L = 57$, $MAX = 200$, and $hop = 3$. Efficacy increases significantly compared with the previous experiment. As a result, the conventional 1-h random-walk algorithm shows a weight synchronization ratio of 99.1667% in 39,292 communication rounds. By contrast, the batch scheme 1-h random-walk algorithm shows a 100 % synchronization rate in a total of 5,450 communication rounds; the number of communication rounds after synchronization is 153. This experiment confirms that the difference in synchronization rates between the conventional 1-h random walk and the batch-scheme 1-h random-walk is insignificant. Rather, the batch-scheme 1-h random-walk algorithm reduces the number of communication rounds.

IV. FURTHER IMPROVEMENTS AND IMPLEMENTATION

A. SHIFTING AND CHANGING RANDOM NUMBER(SCRN)

The TPM must generate a random input value for each round. If τ does not match, the generated random input value becomes unnecessary. Although it varies according to the parameter settings, the 1-h random-walk algorithm does not use more than 30% of the total generated input values, owing

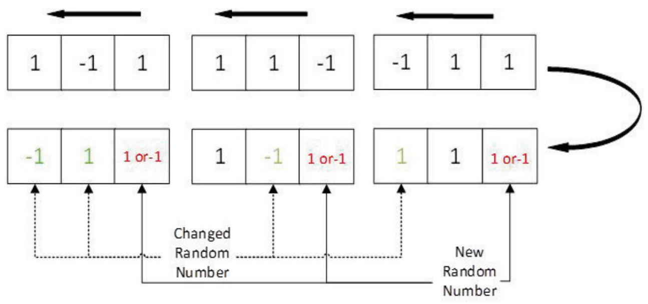

FIGURE 7. Shifting and changing random number (SCRN).

to the mismatch of τ. In particular, the batch scheme must generate as many random input values as the set batch value, but this causes performance degradation in the early stage because the pseudo-random number function is frequently called.

To solve this problem, we designed an SCRN(Shifting and Changing Random Number) as shown in Figure 7, whose working principle is described as follows. During the first round, the input value set is generated at once using a pseudo-random function. In the next round, the input value corresponding to each hidden unit is moved to the right by one bit, and the newly generated bit is inserted at the end. Using this method, we do not need to generate the entire random input value set in each round. We only need K random numbers, where K is the number of TPMs. Therefore, randomness can be maintained while reducing the call frequency of the pseudo-random function. The security of the SCRN is verified through an experiment described in Section V.

B. AMPLIFICATION OF WEIGHT SYNCHRONIZATION RULE

ECC can be used to reduce the number of learning rounds. The neural cryptography-based key exchange algorithm cannot check the mutual synchronization rate because it does not share the state of the weight. Therefore, it performs unnecessary and excessive learning to ensure synchronization. Moreover, the 1-h random-walk algorithm continues to generate and exchange τ to satisfy the condition of $MAX = 3,500$, even though weight synchronization was completed in 15,490 learning rounds, as shown in Figure 3. In the 1-h random-walk experiment, a total of 18,622 learning rounds was performed. Hence, 3,132 more communication rounds were performed after the weights were fully synchronized. This shows that approximately 17% of the total number of communication rounds occurred after the weights were fully synchronized. Therefore, the number of learning and communication rounds can be greatly reduced if the mutual weights are corrected using the parity of the ECC after the weights are synchronized above a certain rate.

The ECC application method proposed in this paper is illustrated in Figure 8. TPM_A encodes ECC in the first half of the weights and only sends the generated parities to TPM_B. TPM_B encodes the second half of the weights and sends the generated parities to TPM_A. This makes weight synchronization difficult, even if the attacker obtains the parities. The two TPMs perform weight synchronization by decoding the received parities.

There are many ECC algorithms such as the Bose—Chaudhuri–Hocquenghem (BCH) code, convolutional code, and Reed–Solomon error correction [12]; however, this study uses the BCH code, which has characteristics of original data and parity being easily separated. Furthermore, it is difficult to infer the original data only by the parity value. BCH encoding is performed in the block unit. For example, if the parameters of the BCH code are set to the message length of $ECC_N = 511$ and the original message length of $ECC_K = 229$, the number of error corrections becomes $ECC_T = 38$. Hence, 38 errors in 229 bits can be corrected whenever encoding is performed once. This method can improve efficiency to a great extent because the same level of synchronization can be achieved, although a small MAX value is used in the synchronization process.

C. IMPLEMENTATION

The proposed TPM learning-based key exchange algorithm with 1-h random-walk learning, batch scheme, SCRN, and ECC works as shown in the pseudocode of Algorithm 1.

The inputs of the key exchange algorithm are parameters for the TPM learning, i.e., the threshold of the weight L, the number of hidden units K, the number of input values for each hidden unit N, the size of batch B, the maximum round of learning MAX, the weight change hop, the parameters of ECC, and the random seed. Note that, since the batch scheme is included, output $\tau^{(0)}$ and hidden unit $\sigma^{(0)}$ are the vector values.

that, since the batch scheme is included, output $\tau^{(0)}$ and hidden unit $\sigma^{(0)}$ are the vector values. First, the TPM_A initializes own weight to a random number from $-L$ to L, and hidden units, round, count to 0. Then, input values are randomly generated and TPM_A repeatedly calculates hidden units and output value for B times by using $Out[utGenerator$ function which shown in Algorithm 2. As mentioned in Section II, TPM consecutively adds input multiplied by weight $w^{(0)}$ to obtain hidden units $\sigma^{(0)}$ and replace the hidden units as -1 or 1. The generated hidden units are multiplied to obtain output $\tau^{(0)}$. To update the weights, TPM_A and TPM_B exchange the output value and update the weights. In order to update the weights, RandomWalks function is also performed B times as shown in Algorithm 3. The 1-h random-walk learning is performed with $hop = 1$ in even rounds and with the hop value set in advance in the odd rounds. When the RandomWalks function is completed, $\tau^{(0)}$ and $\tau^(1)$ are checked to see if they match. If so, the count is increased; otherwise, the count is reset to zero. This process is repeated until the count become MAX.

When the weight update is complete, the weight is corrected using an ECC. TPM_A extracts the parity code by encoding $wfront^{(0)}$ which is the first half of the weights. The extracted code is sent to TPM_B, which extracts the parity code by encoding $wback^{(1)}$ which is the second half of the weights and sends it to TPM_A. Each TPM completes weight synchronization by correcting the weight using the received parity code.

V. ANALYSIS

A. SECURITY OF 1-H RANDOM-WALK

Attack methods for learning models include simple attacks [4], geometric attacks [13], genetic attacks [13], [14], and majority attacks [15]. Among them, majority attacks are known to be powerful attacks against learning models [10]. In this section, the security of the 1-h random-walk learning method is verified. Majority attacks have the characteristic that the attack probabilities become lower as the L value grows [10]. The number of neural networks available to majority attackers is set to M. Table 2 shows the probability of majority attacks according to hop value. The experiment is considered to be successful if the weights of the attacker are synchronized with 98 % or more of the sender and receiver weights.

The experimental results showed that at $L = 40$ and $hop = 3$, the attack probability of the 1-h random-walk learning is not significantly different from that of the conventional random-walk learning. Therefore, 1-h random-walk

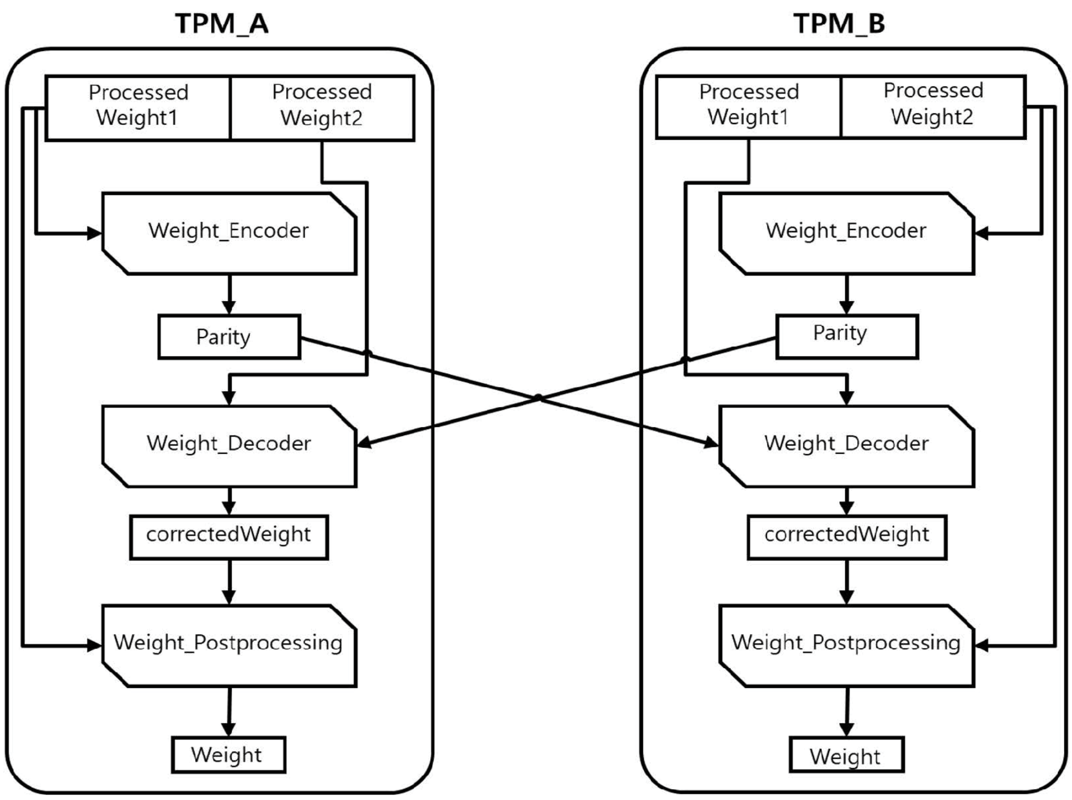

FIGURE 8. Weight synchronization using Error Correction Code (ECC).

TABLE 2. Comparison of the majority attack probability (%) according to the *HOP* value, $M = 100$.

	$L = 30$	$L = 35$	$L = 40$
$hop = 1$(legacy algorithm)	0.0001	0.0001	0.0001
$hop = 2$	0.0001	0.0004	0.0001
$hop = 3$	0.0027	0.0009	0.0001
$hop = 4$	0.0286	0.0103	0.0038

TABLE 3. Comparison of majority attack probability according to SCRN ($K = 3, N = 1,000, M = 100, 10,000$ repetitions).

	$L = 25$	$L = 50$
1-3 random walk	0.0159	0.0001
$SCRN(1bit)$	0.41	0.0000(0/10000)

TABLE 4. Comparison of the majority attack probability of SCRN according to the number of bits ($K = 3, N = 1000, L = 50, M = 100, 10,000$ repetitions).

	1 bit	2 bit	3 bit	4 bit
P_E	0.0000	0.0003	0.0007	0.0208

learning can be used as an alternative to conventional methods because it can maintain a similar security level while decreasing the number of updates compared with the conventional random-walk at $hop = 3$ and $L = 40$. However, at $hop = 4$, the attack probability becomes higher than that at $hop = 3$. Thus, it is recommended to use a hop value of three or lower.

B. SECURITY OF SCRN

To verify the security of SCRN, an attack experiment was conducted for the SCRN method while changing the L value. In this experiment, majority attacks were performed for a TPM to which the SCRN was applied to the 1-h random-walk learning method with $hop = 3$ and for a TPM to which the SCRN was not applied. The results are summarized in Table 3.

It can be seen that at $L = 25$, the attack success rate when the SCRN was applied with an increase of 25 % or more. When L was sufficiently large, the difference between the two methods was insignificant. The security of the SCRN according to the number of moving bits was also verified. The parameters used in this experiment were $K = 3, L = 50$, and $N = 1,000$. The same parameters as in the previous experiment were used, and the results are shown in Table 4.

The experimental results show that the SCRN is the most secure at 1 bit, and the algorithm is more vulnerable against majority attackers when the number of moving bits is larger. Therefore, this experiment confirms that SCRN can be used for the efficiency of using random numbers at $L = 50$.

C. PERFORMANCE

Because the initial input value and weight of the TPM are random values, the communication rounds or execution times are changed whenever the TPM is executed. Therefore, to verify

Algorithm 1 TPM Learning-Based Key Exchange

```
1:  Input $L, K, N, B, MAX, hop, ECC\_N, ECC\_K, seed$
2:  function NN:
3:      $w^{(0)}$ ← Generate random numbers from $-L$ to $L$. Array size is $K * B$
4:      $\sigma^{(0)}$ ← Initialize to 0. Array size is $K * B$
5:      $count, round$ ← Initialize to 0
6:      $input$ ← Generate random numbers from $-1$ or 1 using $seed$. Array size is $K * B$
7:      while $count < MAX$ do
8:          for $i \leftarrow 1$ to $B$ do
9:              $\tau^{(0)}, \sigma^{(0)} \leftarrow$
10:             $OutputGenerator(w^{(0)}, \sigma^{(0)}, input)$
11:         end for
12:
13:         send $\tau^{(0)}$ to $TPM\_B$
14:         receive $\tau^{(1)}$ from $TPM\_B$
15:
16:         for $i \leftarrow 1$ to $B$ do
17:             if $round\%2 == 0$ then
18:                 if $\tau^{(0)}[i] = \tau^{(1)}[i]$ then
19:                     $RandomWalk(w^{(0)}, 1, \tau^{(0)}[i], \sigma^{(0)},$
20:                                     $input, 1)$
21:                 end if
22:             end if
23:             if $round\%2 == 1$ then
24:                 if $\tau^{(0)}[i] = \tau^{(1)}[i]$ then
25:                     $RandomWalk(w^{(0)}, hop, \tau^{(0)}[i],$
26:                                     $\sigma^{(0)}, input, hop)$
27:                 end if
28:             end if
29:             if $\tau^{(0)} == \tau^{(1)}$ then
30:                 $count{+}{+}$
31:             else
32:                 $count \leftarrow 0$
33:             end if
34:             $round{+}{+}$
35:         end for
36:         $input$ ← Using SCRN
37:     end while
38:
39:     $parity^{(0)}$ ← ECC encoding $w^{(0)}_{front}$ and extracting parity from encoded $w^{(0)}_{front}$
40:     send $parity^{(0)}$ to $TPM\_B$
41:     receive $parity^{(1)}$ from $TPM\_B$
42:     $w^{(0)}_{back}$ ← ECC decoding ($w^{(0)}_{back}$ concat $parity^{(1)}$)
43:     $w^{(0)} \leftarrow w^{(0)}_{front}$ concat $w^{(0)}_{back}$
44: end function
```

Algorithm 2 Output Generator

```
1:  function OutputGenerator($w, \sigma, input$):
2:      for $i \leftarrow 1$ to $K$ do
3:          for $j \leftarrow 1$ to $N$ do
4:              $\sigma[i] \leftarrow \sigma[i] + (w[N * K * (i-1) + j] *$
5:                          $input[N * K * (i-1) + j)$
6:          end for
7:      end for
8:
9:      for $i \leftarrow 1$ to $K$ do
10:         if $\sigma[i] > 0$ then
11:             $\sigma[i] \leftarrow 1$
12:         else
13:             $\sigma[i] \leftarrow -1$
14:         end if
15:     end for
16:
17:     $\tau \leftarrow 1$
18:     for $i \leftarrow 1$ to $K$ do
19:         $\tau \leftarrow \sigma[i] * \tau$
20:     end for
21:
22:     return $\tau, \sigma$
23: end function
```

Algorithm 3 1-h Random Walk

```
1:  function RandomWalk($w, \sigma, \tau, input, hop$):
2:      for $i \leftarrow 1$ to $K$ do
3:          if $\sigma[i] = \tau$ then
4:              for $j \leftarrow 1$ to $N$ do
5:                  $w[N * (i-1) + j] \leftarrow w[N * (i-1)$
6:                  $+j] + hop * input[N * (i-1) + j]$
7:                  if $w[N * (i-1) + j] > L$ then
8:                      $w[N * (i-1) + j] \leftarrow L$
9:                  elseif $w[N * (i-1) + j] < -L$
10:                     $w[N * (i-1) + j] \leftarrow -L$
11:                 end if
12:             end for
13:         end if
14:     end for
15: end function
```

TABLE 5. Number of updates until weight synchronization ac-cording to the hop value ($K = 3, N = 1,000, MAX = 3,500$).

	$L = 40$	$L = 50$	$L = 57$
$hop = 1$(legacy algorithm)	835,864	-	-
$hop = 2$	53,515	134,619	1,272,959
$hop = 3$	20,547	34,356	116,134
$hop = 4$	12,210	18,005	36,053

the efficiency of 1-h random-walk learning by comparing it with the conventional random-walk learning, we performed the synchronization process 100 times and compared the average values by the number of weight updates. The results are summarized in Table 5.

The experimental results showed that at $L = 40$, the number of updates was approximately 830,000 for the random-walk learning, and it could be reduced to approximately 20,000 times for $hop = 3$ and to approximately 12,000 times for $hop = 4$.

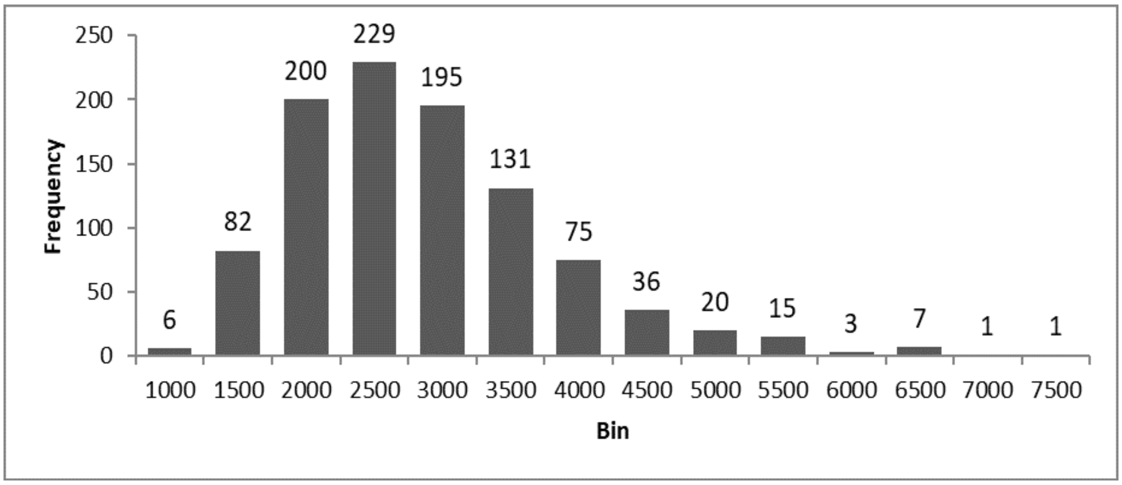

FIGURE 9. Communication round histogram.

TABLE 6. Result of key generation performance experiment of the proposed TPM model ($K = 3, N = 1,000, MAX = 3,500$).

	Round	Key Generation Time (ms)	Match Rate
MIN	786	102	88.9
MAX	7,016	786	99.9
AVG	2,619	274	98.01

Based on the security and performance verified, the time required for key generation was verified by the TPM simulator by applying the 1-h random walk, the batch scheme, the SCRN, and the ECC, as proposed. The performance was verified using the parameter values of $K = 3, L = 57$, and $N = 1,000$, which are practically difficult because of the large number of updates in the conventional random-walk method. With $hop = 3$ and $B = 10$, the simulation finishes if the τ set matches 10 times in a row. The ECC parameters were set to $ECC\_N = 511$, $ECC\_K = 229$, and $ECC\_T = 38$, and the experiment was repeated 1,000 times. The experimental environment and its results are summarized in Table 6.

The results show that the average number of communication rounds was 2,619, the minimum was 786, and the maximum was 7,016. The required time was 786 ms maximum, 102 ms minimum, and 274 ms on average. The average synchronization rate before the ECC application was 98%. In the experiment where only 1–3 random-walk was applied, 116,134 communication rounds were generated on average. This result shows the effect of applying the batch scheme and ECC on the reduction in the average number of communication rounds. Furthermore, in Figure 9, the distribution of the number of communication rounds shows that the synchronization rate was the highest in the 2000–3,000 sections.

VI. CONCLUSION

The conventional TPM model is secure under majority attacks as L becomes larger, but it is practically difficult to use, owing to the high number of learning rounds. Therefore, to maximize practicality while maintaining security, this paper proposed a 1-h random-walk learning algorithm and batch scheme, and the application of SCRN and ECC was used to reduce the execution time.

The experimental results showed that 1-h random-walk learning greatly decreased the number of updates compared with the conventional method while maintaining a similar level of security. In particular, when $L = 40, hop = 1$, and $hop = 3$, the number of updates was reduced 40 times while maintaining a security level similar to that of conventional random-walk learning. Furthermore, the communication rounds were reduced by using the batch scheme and ECC compared with using the 1-h random-walk learning method alone. Additionally, efficiency was improved by using SCRN to reduce the input value generation time while maintaining security.

However, additional studies are required to ensure the efficient use of TPMs. In particular, it is difficult to implement a parallel TPM model based on random-walk learning because a change in weight affects the next round. Furthermore, it is necessary to verify how an active change in the *hop* value influences the number of up-date rounds and corresponding security. Therefore, further research should focus on parallel processing and TPM efficiency by further optimizing the 1-h random-walk method.

REFERENCES

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

[2] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thome, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, and H. J. Te Riele, "Factorization of a 768-bit RSA modulus," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science), vol. 6223. Berlin, Germany: Springer, 2010, pp. 333–350.

[3] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. 35th Annu. Symp. Found. Comput. Sci.*, Santa Fe, NM, USA, 1994, pp. 124–134.

[4] I. Kanter, W. Kinzel, and E. Kanter, "Secure exchange of information by synchronization of neural networks," *Europhys. Lett. (EPL)*, vol. 57, no. 1, pp. 141–147, Jan. 2002.

[5] S. Pattanayak and S. A. Ludwig, "Encryption based on neural cryptography," in *Hybrid Intelligent Systems*, vol. 734, A. Abraham, P. K. Muhuri, A. K. Muda, and N. Gandhi, Eds. Cham, Switzerland: Springer, 2018, pp. 321–330.

[6] M. Abadi and D. G. Andersen, "Learning to protect communications with adversarial neural cryptography," 2016, *arXiv:1610.06918*.

[7] P. Xie, M. Bilenko, T. Finley, R. Gilad-Bachrach, K. Lauter, and M. Naehrig, "Crypto-Nets: Neural networks over encrypted data," 2014, *arXiv:1412.6181*.

[8] A. Engel and C. Van Den Broeck, *Statistical Mechanics of Learning*. New York, NY, USA: Cambridge Univ. Press, 2001.

[9] X. Lei, X. Liao, F. Chen, and T. Huang, "Two-layer tree-connected feed-forward neural network model for neural cryptography," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 87, no. 3, Mar. 2013, Art. no. 032811.

[10] A. Ruttor, "Neural synchronization and cryptography," 2007, *arXiv:0711.2411*.

[11] R. Mislovaty, Y. Perchenok, I. Kanter, and W. Kinzel, "Secure key-exchange protocol with an absence of injective functions," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 66, no. 6, Dec. 2002, Art. no. 066102.

[12] O. Gazi and A. O. Yilmaz, "Turbo product codes based on convolutional codes," *ETRI J.*, vol. 28, no. 4, pp. 453–460, Aug. 2006.

[13] A. Klimov, A. Mityagin, and A. Shamir, "Analysis of neural cryptography," in *Advances in Cryptology* (Lecture Notes in Computer Science), vol. 2501, Y. Zheng, Ed. Berlin, Germany: Springer, Dec. 2002, pp. 288–298.

[14] A. Ruttor, W. Kinzel, R. Naeh, and I. Kanter, "Genetic attack on neural cryptography," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 73, no. 3, Mar. 2006, Art. no. 036121.

[15] L. N. Shacham, E. Klein, R. Mislovaty, I. Kanter, and W. Kinzel, "Cooperating attackers in neural cryptography," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 69, no. 6, Jun. 2004, Art. no. 066137.

JUYOUNG KIM received the M.S. and Ph.D. degrees from Pukyong National University, Busan, South Korea, in 2011 and 2019, respectively. Since 2012, he has been a Researcher with kt ds, Seoul, South Korea. He is currently a Senior Researcher with the Information Security Research Division, Electronics and Telecommunications Research Institute (ETRI). His research interests include open source, secure coding, biometrics, deep learning security, and mobile and wireless network security.

SOOYONG JEONG received the B.S. degree in applied mathematics and the M.S. degree in information security from Kongju National University, South Korea, in 2018 and 2020, respectively, where he is currently pursuing the Ph.D. degree in information security. His research interests include AI-based cryptography, artificial intelligence, and deep learning.

DOWON HONG received the B.S., M.S., and Ph.D. degrees in mathematics from Korea University, Seoul, South Korea, in 1994, 1996, and 2000, respectively. He was a Principal Member of Engineering Staff with the Electronics and Telecommunications Research Institute, South Korea, from 2000 to 2012. In 2012, he joined the Department of Applied Mathematics, Kongju National University, South Korea, where he has been a Full Professor, since 2015. His research interests include cryptography and information security, data privacy, and digital forensics.

NAM-SU JHO received the B.S. degree in mathematics from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea, in 1999, and the Ph.D. degree in mathematics from Seoul National University, South Korea, in 2007. Since 2007, he has been with the Electronics and Telecommunications Research Institute (ETRI), Daejeon, as a Principal Researcher. His research interests include cryptography and information theory.