Enhancing IoT Security using Multi-Layer Feedforward Neural Network with Tree Parity Machine Elements

Erekle Shishniashvili, Lizi Mamisashvili, Lela Mirtskhulava

Department of Computer Science, Iv. Javakhishvili Tbilisi State University/San Diego State University Georgia, Tbilisi, Georgia.

eshishniashvili3@gmail.com; mamisashvili.lizi@gmail.com; lela.mirtskhulava@tsu.ge

***Abstract* - To enhance IoT security level of Tree Parity Machines, TPMs, many researchers recently tried to increase the number of neurons in a single hidden layer. In this paper we propose a novel solution of building a feedforward neural network with multiple hidden layers using elements of Tree Parity Machines. As the number of permutations of weights in these layers rises exponentially, it is almost impossible for an attacker to generate the same key, which is also proven by us using simulations, where 10,000 attacker machines try to imitate the key. While prioritizing security level and complex structures, the algorithm is time efficient as well for real time usage. We run our simulations on an Intel Core processor where key generation takes less than 1 second. After long usage of asymmetric cryptographic algorithms, the modern era requires to explore and test many new ways of generating public as well as private keys.**

Keywords - IoT security, multiple layer feed-forward neural network, tree parity machine, multiple hidden layers, generating secure keys

I. INTRODUCTION

In the era of the Internet of Things every device is connected to each other. More and more we see smart new machines taking steps in our everyday lives. Smart homes, smart cities are buzzwords that we hear daily. According to Forbes, they predicted that spending on IoT will rise up to 1.2 Trillion by 2022 [1]. This means that many families will have devices at home that tell you which products you have left in the fridge, what is your average room temperature, phone calls that you have made and etc. This accumulates into huge data and information that is collected on each person. And to make sure this information will be secure advanced cryptographic methods are required.

Among the huge database of public key algorithms, the RSA is the most widely known public key in cryptosystem. But the algorithm fails to provide the big length of keys that are crucial for today's systems. As RSA algorithms work by factoring two prime numbers. One can easily solve this problem by increasing prime numbers, but it requires a lot of computational cost, which is an undesirable side effect of increasing security [2].

NTRU (Nth Degree Truncated Polynomial Ring Units) - the first lattice-based public key cryptosystem was proposed for securing IoT. NTRU's cryptosystem is much faster than the algorithms like RSA or ECC. NTRU use one-of keys that makes possible changing keys in a few seconds of use. An intruder needs to obtain thousands of keys rather than just one for cracking the encryption on a video files [3]. Key exchange protocol was proposed in securing an information exchanging through IoT network using Wolfram Key Exchange Application giving an opportunity to use a key exchange protocol through the synchronization of two neural networks for encrypting communication using the Hebbian learning rule [4].

Even though this algorithm has been around for many generations, increasing usage of Artificial Intelligence gives hope and raises a lot of questions about whether we can use it in cryptography as well. Successful AI that can master how to encrypt and decrypt data in a way that it will be only decryptable by authenticated users - this concept is very promising and likeable for computer scientists, especially when Machine Learning (ML) and Deep Learning (DL) algorithms have become very powerful tools against many tasks that seemed impossible to master for classic algorithms. For example, Microsoft's deep learning algorithms can outperform humans in image recognition tasks [5], which again outlines the power of AI and how useful it might be if used correctly on cryptosystem tasks.

This was the motivation that made us study the concept of Neural Key exchange. Which means using AI algorithms to create secret keys for authenticated users [6]. To achieve this, the elements of tree parity machines are used. In this paper, we will explain how tree parity machines work and all the adaptations that we came up with, that make this algorithm and structure more robust to hacker attacks, while maintaining desired computational cost and speed.

The remainder of this document is structured as follows. Section 2 mentions the related work. Section 3 describes tree parity machines, since it is used in our algorithm. The proposed algorithm is shown in Section 4. Section 5 provides the results and the analysis of them. Finally, Section 6 includes conclusions and in the Section 7 future work is discussed.

II. RELATED WORK

Tree parity machines have been around for quite some time now and many works have been conducted relating to using AI logic with TPM for key generation.

Revankar, P [7] uses tree parity machines with only one hidden layer to generate the public keys and mathematically proves that it is almost impossible to break the key using brute force in today's computational capabilities. Kinzel and Kanter [8] showed that as synaptic depth of network increases, it becomes harder and harder for attackers to break the key and the computational cost increases exponentially. Also, Dorokhin and Fuertes [9] demonstrated through experimentation that not only the depth, but the level of security of TPM network rises when using large number of neurons in hidden layers.

Not only the architecture of the TPM network, but Dolecki and Kozera [9] experimented with initializing weights in the network. By comparing random allocation with Gaussian distribution, they concluded that random initialization is far outperformed by Gaussian distribution. This means achieving synchronization faster, even though the depth is increasing. But using Gaussian distribution, might be giving more information to the attacker that is trying to synchronize its weights along with the authorized hosts. So, initializing distributions with special functions might decrease the security level. Different from other methods, we will introduce three-layer TPM, with random initialization and we prove that while not losing efficiency we have a higher security level.

III. TREE PARITY MACHINES

A. Basic Operation

Tree Parity Machine initiated by Kanter et al introduces a novel approach in symmetric key exchange. Two machines can be trained by means of mutual learning and as a result, generate a common secret key. The workflow comes in a following way - machines receive the same set of random inputs and start to feed a network. Each of them has one output and when their outputs are the same, weights in a network are updated.

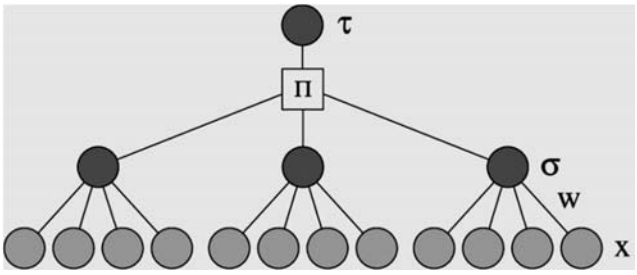

Fig. 1. Tree Parity Machine

Fig. 1 shows an architecture of the TPM when the number of nodes (k) in a hidden layer is 3 and the number of input neurons for each hidden node (n) is 4. Hence, in total, 12 random input values should be generated on each iteration. It should be noted that weights are always discrete, and they vary in a predefined range from -L to L.

$$W_{i,j}^{A/B} \in \{-L, -L+1, \dots, L-1, L\}; \in \{-1,1\}$$

Before calculating final output, activation function $\sigma$is applied to the values of the hidden nodes. Eq. 1 and Eq.2 show this function. The final output defined with the letter $\tau$is simply a product of the values of hidden neurons (Eq. 3) [10].

Equation 1. Result of signum function

$$\sigma_i = sgn(\sum_{j}^{N} w_{ij} x_{ij}$$

Equation 2. Signum function

$$\operatorname{sgn}(x) = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$

Equation 3. Output of TPM

$$\tau = \prod_{j=1}^{K} \sigma_i$$

B. Learning

As mentioned, TPMs use mutual learning to synchronize and it is different from what a general learning problem is in neural networks. Here, a model does not try to approach a static point through the training. TPMs serve both as trainer and trainee and therefore, use each other's results to adjust the weights in a way that eventually, their outputs synchronize [11]. There are different learning rules used for updating weights, such as Hebbian rule, Anti-Hebbian rule or Random Walk. Here, we will examine TPM that uses the Random Walk algorithm since our approach is also connected to it. The Random Walk rule that is used for updating weights (Eq. 4) is combined with the functions $\theta$and g resulting in AND logical operation and clipping of the weights in a range of [-L, L] respectively [12].

Equation 4. Random Walk

$$w_i^+ = g(w_i + x_i \theta(\sigma_i \tau) \theta(\tau^A \tau^B))$$

While considering the mutual learning of the two networks for generating a common secret key, the presence of the third network - attacker - should be taken into account. If the attacker has the same machine, she can also try to synchronize the network and therefore, access the

secret key that is not apparently generated only for two points. As mentioned, the weights of two networks that should share a key are being updated when the outputs of their machines are the same. There can arise three cases:

1. $\tau_A \neq \tau_B$,
2. $\tau_A = \tau_B \neq \tau_C$,
3. $\tau_A = \tau_B = \tau_C$

In the third case, the attacker also updates the weights and therefore, the chance of her machine's synchronization within the other two increases (See Algorithm 1 for implementation of attacking simulations). To decrease the chance of synchronizing the attacker's machine, it is necessary to increase the synaptic depth of the weights meaning that we should increase the range in which weights vary (L). In this case, the cost for synchronizing the attacker grows exponentially. The higher synaptic depth of the weights synchronization of two intended parties as well. However, the cost of their synchronization grows polynomially and therefore, it is still an efficient approach to use for security. In the next section, the novel approach of changing TPM structure for better protection will be described.

IV. AN ENHANCED IOT SECURITY METHOD

A. *Multi-layer Feedforward Neural Network with TPM Elements - How It Works*

We introduce a novel approach of combining TPM workflow with the classic multi-layer feedforward neural network. To explain, mutual learning will be used here as well. The set of random inputs will be fed to the network and the single output will be produced. The main difference between this model and the classic TPM is that we add hidden layers in our architecture (more details in the next subsection).

B. *Structure*

Fig. 2 shows the architecture of our network which consists of input/output and 3 hidden layers. The model is initialized with three hyper parameters. One of them defines the number of nodes in the last layer of the network (k). The range for weights is also predefined and for each hidden layer, weights vary from -L to L. There is a third hyper parameter (n) that connects the number of inputs to the number of nodes in the last hidden layer (k). Specifically, there are as many input values as k multiplied by n. To start from the last layer, the number of nodes on the next hidden layer is 3 times bigger. However, on the first hidden layer, this number is adjusted in a way that the number of input values divide on the number of nodes in this layer without a remainder. Here, it is notable that in our architecture, the input neurons and the values of the hidden layers do not affect the final output in the same way. To explain, for each node on the first hidden layer - a_1, a weights vector w_1 consists of as many entries as it is defined for that specific node, while on the next layers, each a_i is densely connected to the ones in the previous layer (Fig. 2) depicts this behavior). The activation function is applied on each layer (signum function remains from the classic TPM) (Eq. 1 and Eq. 2) and the output is a product of the values on the last layer (Eq. 3).

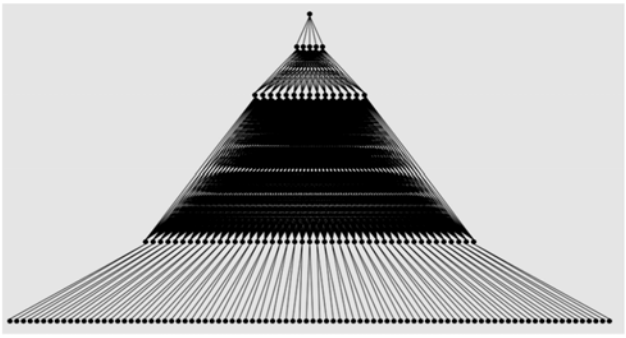

Fig. 2. Multi-layer feedforward neural network with TPM elements.

C. *Learning*

Mutual learning or the chaotic moves of the networks eventually result in a successful synchronization of them and as mentioned, the higher synaptic depth of the weights significantly lower the chance of successful attack. In this case, the learning process becomes more complex, since 3 hidden layers affect the final output of the machine instead of one. The weights on all layers are updated using the rule of Random Walk (Eq. 4). If the attacker had to adjust the weights of the machine in [-L, L] and achieve synchronization, now, the problem expends significantly, since the attacker's machine should adjust the weights matrix consisting of 3 times more rows for each layer. The procedure of chaotic synchronization becomes more complex, but the results are promising for the sake of more security (Section 5 describes this in details).

V. RESULTS AND DISCUSSION

To measure the performance of our proposed algorithm, we have created two networks (A and B) intended for generating a common symmetric key and additionally, added 10000 machines that are being used by the attacker throughout the synchronization of A and B (see Algorithm 1). Moreover, to better evaluate the results in terms of both - time and security - we will show how classic TPM deals with the same case. For testing the proposed algorithm as well as classical TPM, hyper parameters will be the same - k = 5, n = 18 and l = 8

While we prioritize security over speed of the key generation, we still get the time that is desirable for usage of this algorithm. With the proposed algorithm, A and B

synchronize on an Intel Core processor in less than 1 second. The two networks need 87 iterations and, in this time, from the 1000 attackers, 2 of them are able to synchronize the machine (average results from multiple tries). Figure 3 shows how synchronization score changes throughout iterations until the two machines are finally synchronized.

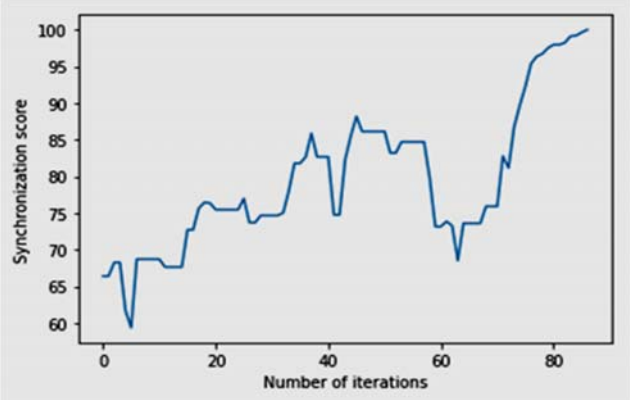

Fig. 3. Synchronization history with the proposed algorithm

```
1.  k = 18
2.  n = 5
3.  l = 8
4.
5.  Alice = Machine(k,n,l)
6.  Bob = Machine(k,n,l)
7.  NUM_OF_ATTACKER_MACHINES = 10000
8.
9.  attackerMachines ← empty array
10. attackerIters ← empty array
11.
12. For i := 0 to NUM_OF_ATTACKER_MACHINES do
13.         add Machine(k,n,l) in attackerMachines
14.
15. while (not synced) do
16.         X ← Generate random number in a range [-1, 1]
17.         tauAlice = Alice.predict(A)
18.         tauBob = Bob.predict(x)
19.
20.         Alice.updateWeights(x, tauBob) //mutual learning
21.         Bob.updateWeights(x, tauAlice)
22.
23.         score = 100 * sync_score(Alice, Bob)
24.
25.         For i, machine := 0 in enumerate(attackerMachines) do
26.             tauE = machine.predict(X)
27.
28.             if (tauAlice == tauBob == tauE) then
29.                 machine.updateWeights(X, tauA)
30.                 attackerIters[i] += 1
31.
32.         if syncScore == 100 then
33.             synced = True
```

Algorithm 1. Pseudocode of simulating Alice, Bob and 10000 attacker machines

On the other hand, the classic TPM needed 201 iterations to synchronize A and B. The time needed is less than 1 second here as well. From the 1000 attackers, 8 have synchronized (the results are average of multiple tries here as well). Figure 4 shows how synchronization score changes throughout iterations until the TPMs are finally synchronized.

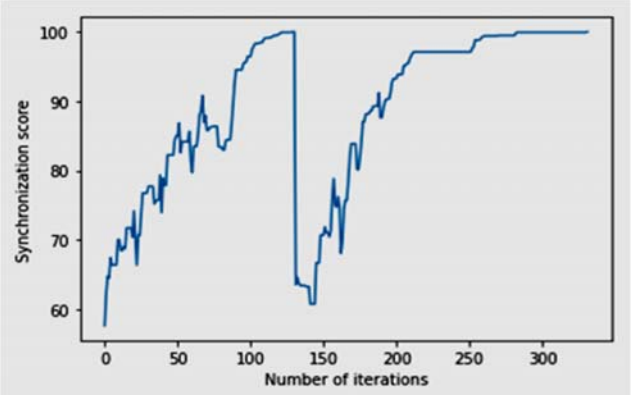

Fig. 4. Synchronization history with the classic TPM

VI. CONCLUSION ANDF FUTURE WORK

In conclusion, in this paper we described the need of more advanced cryptographic measures in today's society as the growing number of IoT devices gathers data from every part of our daily life, so all the systems should be secured in a way that data leakage is minimized. To cope with that task, we introduced a new structure of Tree Parity Machines, Artificial Intelligence algorithm that is very promising in generating public keys for authorized users, making it almost impossible for unauthorized ones to crack it. All the recent techniques involved using one hidden layer, while in this paper we showed that using 3 hidden layers with specific configuration makes the algorithm more robust, while not drastically increasing computational time of key generation. After performing an attack simulation on our model, we show that success of syncing unauthorized machines with authorized ones is 0.01% using the random walk algorithm, which just randomly updates weight and does not depend on the output. While we believe using other methods will prove more useful.

Overall, we can say that our method is successful at increasing the level of security of Tree Parity Machines and introduces more combinations that are more robust against brute force as well as passive attacks.

Future Work:

As for future work, we are interested in performing research around the value of L. And how it can vary between different hidden layers. For example, we can make a range [-L, L] for the first layer while doubling it and making it [-2L, 2L], or opposite shrinking it. We think that

there might be some connection between these values and hidden layers. Another thing that we want to try out in future, is to use Hebbian or Anti-Hebbian rule to update our weights, which is proven to be even more secure. We also want to test this approach combined with generative deep learning models that are used heavily and perform very well. We wish to combine synchronized TPM weights with encoder and decoder algorithms, so that generative models will be able to successfully encode and decode if the set of weights that generate public keys are matching.

ACKNOWLEDGEMENT

This research has been supported by the Shota Rustaveli National Science Foundation of Georgia and Turkish Scientific and Technological Research Council joint grant 04/03.

REFERENCES

[1] Columbus, L. (2020). 2018 Roundup of Internet of Things Forecasts And Market Estimates. Retrieved 10 May 2020, from https://www.forbes.com/sites/louiscolumbus/2018/12/13/2018-roundup-of-internet-of-things-forecasts-and-market-estimates/#6634be0d7d83

[2] Zhou, X. and Tang, X., 2011. Research And Implementation Of RSA Algorithm For Encryption And Decryption - IEEE Conference Publication. [online] Ieeexplore.ieee.org. Available at: <https://ieeexplore.ieee.org/abstract/document/6021216>

[3] L. Mirtskhulava, N. Gulua, N. Meshveliani. Iot Security Analysis Using Neural Key Exchange. GESJ: Computer Science and Telecommunications 2019|No.2(57)

[4] L. Mirtskhulava, N. Gulua, N. Meshveliani. Ntru Cryptosystem Analysis For Securing IoT. GESJ: Computer Science and Telecommunications 2019|No.1(56)

[5] Thomsen, M., 2020. Microsoft's Deep Learning Project Outperforms Humans In Image Recognition. [online] Forbes. Available at: https://www.forbes.com/sites/michaelthomsen/2015/02/19/microsofts-deep-learning-project-outperforms-humans-in-image-recognition/#1eb4accc740b

[6] Chakraborty, S., Dalal, J., Sarkar, B. and Mukherjee, D., 2015. Neural Synchronization Based Secret Key Exchange Over Public Channels: A Survey. [online] Arxiv.org. Available at: https://arxiv.org/ftp/arxiv/papers/1502/1502.05153.pdf

[7] Revankar, P., Gandhare, W. and Rathod, D., 2010. Private Inputs To Tree Parity Machine. [online] Pdfs.semanticscholar.org. Available at: <https://pdfs.semanticscholar.org/dfbc/7fde64f2b55d8767daa50e9ea2130bf6db69.pdf>

[8] Kinzel, W. and Kanter, I., n.d. Interacting Neural Networks and Cryptography. Advances in Solid State Physics, pp.383-391.

[9] Salguero Dorokhin, É., Fuertes, W. and Lascano, E., 2019. On the Development of an Optimal Structure of Tree Parity Machine for the Establishment of a Cryptographic Key. Security and Communication Networks, 2019, pp.1-10.

[10] Dolecki, M. and Kozera, R., 2015. The Impact of the TPM Weights Distribution on Network Synchronization Time. Computer Information Systems and Industrial Management, pp.451-460.

[11] W. Kinzel and I. Kanter, "Neural cryptography," Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP '02., Singapore, 2002, pp. 1351-1354 vol.3, doi: 10.1109/ICONIP.2002.1202841.

[12] A. Klimov, A. Mityagin and A. Shamir, "Analysis of Neural Cryptography", Springer.com, 2002. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-36178-2_18.