Enhancing Mobile Communication System Security via Neural Cryptography Applications

Lela Mirtskhulava[1(✉)], Nana Gulua[2], and Khatuna Putkaradze[2]

[1] Ivane Javakhishvili Tbilisi State University, 0186 Tbilisi, Georgia
lela.mirtskhulava@tsu.ge
[2] Sokhumi State University, 0186 Tbilisi, Georgia
{ngulua,khatuna-putkaradze}@sou.edu.ge

Abstract. The given paper aims to understand how the emerging technologies of 5G and Beyond, and artificial intelligence could affect the security of mobile communication systems and to explore ways to mitigate any potential risks. We analyze the vulnerabilities of 5G and beyond systems to attacks and the potential risks associated with the use of AI in these systems. We optimise a security protocol and systems that can withstand the power of quantum computing and enhance the security of IoT and "5G and beyond" systems. We identify potential areas of regulatory intervention and ensure that AI is used responsibly and transparently in 5G and beyond systems. The proposed method is novel - implementing a type of synchronization mechanism between two Tree Parity Machines (TPMs) using feedback. A feedback mechanism helps to adjust the weights of one TPM based on the outputs of both TPMs on the same input. This is different from traditional methods of training neural networks, which usually involve minimizing a cost function or optimizing weights using some form of gradient descent. The use of TPMs instead of traditional neural networks is also somewhat novel as TPMs are a type of recurrent neural network that has been proposed for use in cognitive modeling. In the training phase of TPMs, the Hebbian learning rule is employed to adjust connection weights between perceptrons.

Keywords: 5G and beyond · TPMs · synchronization · security · neural cryptography

1 Introduction

In the evolving landscape of mobile communication systems, safeguarding sensitive information and ensuring secure data transmission has become paramount. The growing utilization of mobile devices continues unabated, and the need for robust security mechanisms becomes more critical than ever. We offer an innovative approach gaining prominence to address the security challenges is integrating neural cryptography applications into mobile communication systems [1].

Neural cryptography - a branch of artificial intelligence and cryptography, leverages the power of neural networks to secure communication channels against unauthorized access and cyber threats. This cutting-edge methodology goes beyond traditional

X. Li et al. (Eds.): SmartGift 2024, LNICST 600, pp. 145–156, 2025.
https://doi.org/10.1007/978-3-031-78806-2_9

encryption techniques providing a dynamic and adaptive security framework tailored to the unique challenges posed by mobile communications.

Neural cryptography uses adaptive encryption protocols capable of dynamically adjusting to the changing nature of communication patterns. Via leveraging neural networks, these protocols can continuously learn and optimize encryption strategies based on real-time data, securing them and making them more resilient against emerging threats. Neural cryptography can detect anomalies in communication systems by integrating behavioral analysis. Then the system can identify deviations from typical usage, triggering alerts and responses to potential security breaches [2].

Classical authentication methods often face challenges caused by sophisticated cyber attacks. Neural cryptography can enhance security by using self-learning authentication mechanisms that continuously adapt to evolving patterns and reduce the risk of unauthorized access. On the other hand, with advancing quantum computing capabilities, classical cryptographic methods are becoming susceptible to new vulnerabilities. Neural cryptography applications provide quantum-resistant security measures and ensure the long-term viability of mobile communication system security.

Neural cryptography applications provide robust key management systems, securing cryptographic keys against various attack vectors. By implementing neural networks, these systems enhance the randomness and complexity of generated keys, making them more resistant to brute-force attacks. The integration of neural cryptography enables mobile communication systems' real-time threat detection and response mechanisms enable the system to adapt its security measures dynamically. Neural cryptography applications ensure the development of end-to-end encryption mechanisms with minimal latency.

The intersection of quantum AI (Artificial Intelligence), 5 G (Fifth Generation), and beyond telecommunications is a rapidly evolving field that has the potential to revolutionize various industries. However, as with any new technology, it also poses significant security challenges. The novelty of using a detailed analysis of the impact of quantum AI on the security of IoT and 5 G lies in the fact that it addresses a critical gap in current research. Traditional cryptographic methods used in IoT and 5 G systems may not be sufficient to protect against attacks from quantum computers. Therefore, understanding the potential impact of quantum AI on the security of these systems is crucial to developing effective countermeasures [3].

Long-term usage of public-key algorithms showed the necessity to explore new methods of public and private key generations. GSMA organization announced that the security of wireless networks (5 G/LTE/3 G/2 G) and user equipment (UE), is essential to providing secure services since 5 G evolving brings new security threats. 5 G wireless networks are still in the process of deploying across the world where the transition from 4 G to 5 G addresses many challenges such as addressing the threats faced in 2 G/3 G/4 G networks. In 5 G and beyond, the implementation of new technologies introduces new potential threats in the industry [4].

5G and beyond standard security techniques are based on classical cryptography and unfortunately do not take into account the threats or issues that can be potentially caused by quantum computing. The development of quantum computers (QC) is going rapidly and promising to solve computing problems that cannot be solved by traditional

computers. Moreover, they can generate new threats at a very high speed. They can break traditional encryption algorithms like the RSA and ECC in seconds using quantum algorithms such as Shor's algorithm. It is necessary to shift to the world of quantum cryptography [5, 6].

AI is expected to be widely used to mitigate previous undetected and AI-driven attacks in 5 G networks in real time and benefit security. Leveraging Machine Learning AI techniques will automate threat detection. AI is particularly relevant to be used in generating the volumes of data in 5 G networks. The wide employment of AI in 5 G networks will enhance 5 G security.

The 3rd Generation Partnership Project (3 GPP) offered a standard for 5 G networks. It contains the identity protection scheme, which addresses the important privacy problem of permanent subscriber-identity disclosure. This offer contains two stages: the identification stage, which provides the security context between service providers and mobile subscribers using the authenticated key agreement with the symmetric key. 3 GPP offers to protect the identification stage using a public-key scheme. They offer to use the Elliptic Curve Integrated Encryption Scheme (ECIES). The offered scheme is not secure against the attacks of quantum computers. It is important to integrate the quantum-resistant scheme into 5 G networks and beyond.

This motivation drove our exploration into the Neural Key Exchange concept, involving the utilization of AI algorithms to generate secret keys for authenticated users. To accomplish this goal, we employ the components of tree parity machines. This paper delves into the workings of tree parity machines and outlines the adaptations we have devised to enhance the algorithm's resilience against quantum attacks. Our methodology aims to fortify both the algorithm and its structure, ensuring increased robustness while preserving the desired balance between computational cost and speed.

The implemented code incorporates a synchronization mechanism between two Temporal Product Machines (TPMs) through feedback. This approach stands out due to its novelty, employing a feedback mechanism to fine-tune the weights of one TPM based on the outputs of both TPMs for the same input. This diverges from conventional neural network training methods, which typically center around minimizing a cost function or optimizing weights through gradient descent. Furthermore, the use of TPMs, instead of traditional neural networks, adds to the novelty, given that TPMs are a subtype of recurrent neural networks suggested for cognitive modeling.

2 Related Work

Tree parity machines (TPMs) have a longstanding history, with numerous studies exploring the integration of AI logic for key generation. Revankar, P [7], for instance, employed TPMs with a single hidden layer to generate public keys and demonstrated mathematically that breaking the key through brute force is nearly impossible given today's computational capabilities. Kinzel and Kanter [8, 9] established that increasing the synaptic depth of the network makes it progressively more challenging for attackers to compromise the key, resulting in an exponential rise in computational cost.

In a similar vein, Dorokhin and Fuertes [10] conducted experiments revealing that not only does the depth influence the security of a TPM network, but also the use of a large number of neurons in hidden layers enhances its security. Dolecki and Kozera [11] explored the impact of weight initialization in the network, comparing random allocation with Gaussian distribution. Their findings suggested that Gaussian distribution outperforms random initialization, facilitating faster synchronization even with an increasing depth. However, caution is advised, as using Gaussian distribution may inadvertently provide more information to potential attackers attempting to synchronize their weights with authorized hosts, potentially compromising security.

Mirtskhulava et al. [12] proposed a systematic approach is essential for monitoring and mitigating all potential threats to IoT security. Encryption stands out as a primary requirement to establish secure communication within the IoT framework. Key exchange is pivotal in securing information exchange across the IoT network. Neural networks present an effective strategy through the synchronization process, employing the Hebbian learning rule to balance weights. This synchronization in neural networks provides a cryptographic key-exchange protocol. A significant advantage of this process is the extended time required for an attacker to guess the generated key.

Mirtskhulava et al. [13] proposed an analysis of NTRU (Nth Degree Truncated Polynomial Ring Units), which stands out as the first lattice-based public key cryptosystem, primarily due to its notable advantages, such as the absence of an established install base and resilience against long-term cryptanalysis. NTRU's cryptosystem demonstrates significantly higher speed compared to algorithms like RSA or ECC. Employing one-off keys, NTRU allows for key changes within seconds of use.

This characteristic imposes a formidable challenge for potential attackers, as they would need to obtain hundreds or thousands of keys instead of just one to crack the encryption on video files. The Internet of Things (IoT) represents a vast network comprising thousands of interconnected smart devices, including cars, smart medical equipment, industrial control systems, smart grids, and more. The escalating use of connected vulnerable devices poses serious security threats, leading to an increased risk of security breaches.

Through these vulnerable devices, attackers can potentially gain access to sensitive information, compromising enterprise networks. The proliferation of malwares, botnets, and other security threats exacerbates the challenges associated with the security of IoT networks.

Shishniashvili, Mamisaishvili, and Mirtskhulava [14] introduced an innovative approach to constructing a feedforward neural network with multiple hidden layers by incorporating elements from Tree Parity Machines. With the exponential increase in the permutations of weights within these layers, it becomes highly challenging for an attacker to replicate the key. Simulations, involving 10,000 attacker machines attempting to imitate the key, substantiate this claim. Balancing security considerations and the complexity of structures, the proposed algorithm maintains time efficiency for real-time applications. Simulations executed on an Intel Core processor demonstrate that key generation takes less than 1 s. Given the evolving landscape of asymmetric cryptographic algorithms, exploring and testing novel methods for generating both public and private keys is imperative in the modern era.

Mirtskhulava et al. [15] explored the lattice-based open-source NTRU cryptosystem designed for public-key encryption, specifically applicable in the context of the Internet of Things (IoT). Given that IoT encompasses individuals' sensitive data and various aspects of personal lives, it necessitates a specialized level of protection. The security of IoT involves safeguarding both the network and the connected devices. Cryptographic technologies emerge as optimal solutions to counter communication attacks.

Mirtskhulava et al. [16] proposed an approach for securing the Mobile Internet of Things (MIoT) ecosystem in the era of rapidly advancing mobile internet (5G and 6G) involves the adoption of post-quantum digital signature-based blockchain. This method is deemed suitable as the MIoT ecosystem gains significance due to the expanding accessibility and higher speeds of mobile internet.

In contrast to existing approaches, we propose a three-layer TPM with random initialization and provide evidence that, without sacrificing efficiency, it offers a heightened level of security.

3 Tree Parity Machine Architecture

3.1 TPM with Three-Layered Perceptron

We offer a Tree Parity Machine (TPM) with a three-layered perceptron architecture (Fig. 1) consisting of three layers of perceptrons organised in a tree-like structure. In the given architecture, the first layer of the perceptron is connected to the input vector, where the second layer is connected to the first layer, and the third layer represents the output layer.

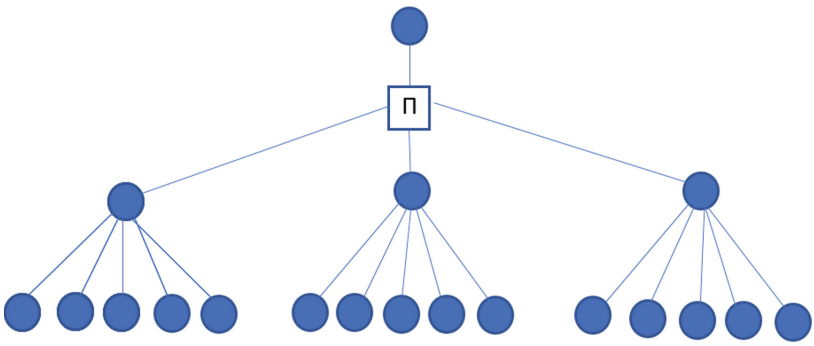

Fig. 1. The structure of the Tree Parity machine.

Each perceptron has binary weights in the network, where the weights are randomly initialized. The perceptrons have the same number of inputs in each layer, which is dimensionality equal to the input vector. The outputs of the first-layer perceptrons are connected to the inputs of the second-layer perceptrons, and the outputs of the second-layer perceptrons are connected to the inputs of the third-layer perceptrons [17].

Mathematically, the output of a TPM with a three-layered perceptron architecture can be represented as follows:

$$y = sign(W_3 \cdot g(W_2 \cdot g(W_1 \cdot x))) \tag{1}$$

where:

- x is the input vector
- W_1 is the weight matrix between the input layer and the first hidden layer
- W_2 is the weight matrix between the first and second hidden layers
- W_3 is the weight vector between the second hidden layer and the output layer
- g() is the activation function
- sign() is the sign function mapping positive values to $+1$ and negative values to -1

In the TPM, the weights of the perceptrons are updated through a learning process called training. During training, the TPM is presented with a set of input patterns and their corresponding labels. The weights are adjusted based on the difference between the actual output and the desired output, using a learning rule such as the perceptron learning rule or the backpropagation algorithm.

The goal of training a TPM is to learn a set of weights that can correctly classify input patterns into their respective categories. Once the TPM is trained, it can be used to classify new input patterns by computing the output of the network and comparing it to a threshold value. If the output is above the threshold, the input is classified as belonging to one category, and if it is below the threshold, the input is classified as belonging to another category.

3.2 TPM Key Agreement Protocol

The TPM key agreement protocol is a method of exchanging a secure key between two parties based on the output of a TPM. The protocol comprises the following three steps: initialization, key generation, and key agreement.

1st step -Initialization: both parties agree on the number of input bits, the number of layers, and perceptrons in the TPM. They also agree on a shared seed value, which is used to initialize the weights of the TPM.

2nd step – Key Generation: both parties generate random input bit vectors and use them as inputs to their respective TPMs. The TPMs produce output vectors that are sent over a secure channel to the other party. The parties then compute the dot product of their output vector with the received output vector, and if the result is positive, they set the corresponding bit in a shared secret key to 1. Otherwise, they set it to 0. This process is repeated for a predetermined number of iterations to generate a shared secret key.

3rd step – Key Agreement: both parties compare their generated secret keys to ensure that they are identical. If the keys match, they can use them as shared secret keys for symmetric encryption or other cryptographic applications.

The security of the TPM key agreement protocol is based on the difficulty of computing the dot product of the output vectors of the TPMs. The weights of the TPMs are initialized randomly and kept secret, and the output vectors are highly sensitive to changes in the input vectors. Therefore, an attacker would need to know the weights of the TPMs to compute the dot product, which is computationally infeasible in most cases.

The train_tpm function applies this rule to each perceptron's output for every input pattern, facilitating weight updates. Widely utilized in various neural network architectures such as Hopfield networks, Boltzmann machines, and self-organizing maps, the Hebbian learning rule proves to be a straightforward yet effective mechanism for learning.

4 Security of TPMs

The security of TPMs lies in their ability to perform non-linear classification and their resistance to certain types of attacks. However, like any cryptographic algorithm, TPMs are not completely secure, and their security depends on various factors such as the size of the network, the quality of the seed used to initialize the weights, and the number of training iterations.

One of the primary advantages of TPMs is their ability to perform non-linear classification, which makes them resistant to linear attacks. In contrast to linear classifiers, TPMs can learn complex decision boundaries between input patterns, which makes them more difficult to attack. Additionally, TPMs are highly sensitive to changes in the input, and small changes in the input can cause large changes in the output, which makes them resistant to certain types of attacks such as gradient-based attacks.

However, TPMs are vulnerable to certain types of attacks such as brute force attacks and adversarial attacks. Brute force attacks involve guessing the weights of the TPM through trial and error, and the security of the TPM depends on the size of the network and the quality of the seed used to initialize the weights. Adversarial attacks involve modifying the input to the TPM to cause it to misclassify the input. While TPMs are resistant to certain types of adversarial attacks, they are not completely immune, and recent research has shown that it is possible to construct adversarial examples that can fool the TPM.

In addition to the above, TPMs can also be vulnerable to side-channel attacks, where an attacker gains information about the network by analyzing its behavior or the physical environment in which it is located. For example, an attacker might be able to learn the weights of the TPM by analyzing the electromagnetic radiation emitted by the network.

It is imperative to address the susceptibility of TPMs to side-channel attacks, which pose a significant threat to the security of mobile communication systems. Side-channel attacks exploit unintended information leakage from the system's physical implementation rather than directly targeting cryptographic algorithms. When it comes to electromagnetic radiation analysis, an attacker can intercept and analyze the emitted radiation from the TPM network to gain crucial information, such as the weights of the TPMs. Moreover, attackers can potentially compromise the security of the communication system by discerning these weights, undermining the confidentiality and integrity of transmitted data.

The above-mentioned vulnerability underscores the importance of enhancing the resilience of synchronized TPMs against side-channel attacks. It requires the implementation of robust countermeasures, such as noise injection to mitigate information leakage and prevent adversaries from exploiting these vulnerabilities. Comprehensive security protocols will play a crusial role to detect and respond to potential side-channel attacks effectively, thereby enhancing the level of security of mobile communication systems employing synchronized TPMs.

Addressing these vulnerabilities and implementing effective countermeasures are essential steps towards strengthening the security of mobile communication systems against sophisticated adversaries aiming to exploit side-channel weaknesses in synchronized TPMs. Mitigating these risks, we can ensure the integrity, confidentiality, and reliability of communication channels in real-world deployment processes.

5 Synchronization Time of TPMs with Differentiated Input Vectors

The synchronization time of Tree Parity Machines (TPMs) with differentiated input vectors refers to the amount of time it takes for two or more TPMs to synchronize their output vectors given different input vectors. The synchronization time is an important metric for assessing the performance of TPMs in tasks such as key agreement, where two parties need to synchronize their TPMs to generate a shared secret key.

The synchronization time of TPMs with differentiated input vectors depends on several factors such as the size of the network, the number of layers and perceptrons, and the quality of the seed used to initialize the weights. Generally, the synchronization time is longer for larger networks and networks with more layers and perceptrons, as there are more parameters to synchronize. Additionally, the synchronization time can be affected by the quality of the seed used to initialize the weights, as poor initialization can lead to longer convergence times.

One way to reduce the synchronization time of TPMs with differentiated input vectors is to use a pre-synchronization phase, where the TPMs are trained with the same input vectors for a few iterations before being used with differentiated input vectors. This pre-synchronization phase can help to align the output vectors of the TPMs, which can reduce the amount of time needed for synchronization when different input vectors are used.

Another way to reduce the synchronization time of TPMs is to use a smaller number of layers and perceptrons, which reduces the number of parameters that need to be synchronized. However, reducing the number of layers and perceptrons can also reduce the complexity of the decision boundary learned by the TPM, which can affect its ability to perform non-linear classification.

6 Hebbian Learning Rule

The Hebbian learning rule is a simple unsupervised learning rule that reinforces connections between neurons that are active at the same time. The rule can be expressed mathematically as:

$$\Delta w_{ij} = \eta \cdot x_i \cdot x_j \tag{2}$$

where Δw_{ij} is the change in weight between neuron i and neuron j, η is the learning rate, x_i and x_j are the binary states of neuron i and neuron j, respectively. This means that the weight between two neurons is increased when both neurons are active ($x_i = 1$, $x_j = 1$), and is decreased when both neurons are inactive ($x_i = -1$, $x_j = -1$). If only one neuron is active, the weight remains unchanged.

The Hebbian learning rule is used to update the weights of the connections between the perceptrons in the TPMs during the training phase. The train_tpm function applies the Hebbian learning rule to the output of each perceptron in the TPM for each input pattern and updates the weights accordingly. The Hebbian learning rule is a simple but powerful mechanism for learning in neural networks and has been used in many different types of neural networks, including Hopfield networks, Boltzmann machines, and self-organizing maps.

7 Simulation Results and Discussions

In this section, we discuss the results of the simulation by generating the code in Python. This code defines two TPMs, tpm1 and tpm2, each with num_layers layers of size layer_size and num_bits input bits. The initial weights for the TPMs are randomly generated using the *np.random.randint* function with values of -1, 0, and 1. The feedback function takes the two TPMs and an input vector as arguments, and returns the output vectors of the TPMs after processing the input, as well as updates the weights of the second TPM if there is a difference in output with the first TPM.

The synchronization loop runs for 100 iterations, where in each iteration, a new input vector is generated randomly using the *np.random.randint* function. The feedback function is called with the two TPMs and the input vector, which produces the output vectors of the TPMs and updates the weights of the second TPM if needed (See Table 1).

The outputs of the TPMs and the updated weights of the second TPM are printed after each iteration. However, the way the function updates the weights of tpm2 based on the diff variable and input vector is incorrect and will not produce the desired results.

The above code does not use a traditional training algorithm like backpropagation to adjust the weights of the TPMs. Instead, it uses a feedback mechanism to synchronize the weights of two separate TPMs.

During each iteration of the loop, the code generates a random input vector and applies it to both TPMs using the feedback function. The feedback function calculates the output vectors of both TPMs for the input vector and compares them. If the output vectors differ, it calculates the difference and adjusts the weights of the second TPM accordingly. The loop continues until the output vectors of both TPMs are identical.

Table 1. TP1 and TPM 2 outputs for some iterations

Iteration 57	**Iteration 58**	**Iteration 59**	**Iteration 82**	**Iteration 99**
TPM 1 output: [1 1 1]	TPM 1 output: [-1 -1 -1]	TPM 1 output: [1 1 1]	TPM 1 output: [0 0 0]	TPM 1 output: [1 0 -1]
TPM 2 output: [1 1 1]	TPM 2 output: [-1 -1 -1]	TPM 2 output: [1 1 1]	TPM 2 output: [0 0 0]	TPM 2 output: [1 1 -1]
TPM 2 weights after feedback: [[[3 3 -2] [3 4 -3] [3 3 -2] [4 5 -2] [3 4 -1]] [[0 1 -3] [2 2 -3] [2 1 -4] [0 1 -3] [2 2 -5]] [[4 -2 0] [2 -2 -1] [4 -1 0] [3 -3 -1] [4 -1 1]]]	TPM 2 weights after feedback: [[[3 3 -2] [3 4 -3] [3 3 -2] [4 5 -2] [3 4 -1]] [[0 1 -3] [2 2 -3] [2 1 -4] [0 1 -3] [2 2 -5]] [[4 -2 0] [2 -2 -1] [4 -1 0] [3 -3 -1] [4 -1 1]]]	TPM 2 weights after feedback: [[[3 3 -2] [3 4 -3] [3 3 -2] [4 5 -2] [3 4 -1]] [[0 1 -3] [2 2 -3] [2 1 -4] [0 1 -3] [2 2 -5]] [[4 -2 0] [2 -2 -1] [4 -1 0] [3 -3 -1] [4 -1 1]]]	TPM 2 weights after feedback: [[[2 3 -3] [2 4 -4] [2 3 -3] [3 5 -3] [2 4 -2]] [[0 1 -3] [2 2 -3] [2 1 -4] [0 1 -3] [2 2 -5]] [[4 -2 1] [2 -2 0] [4 -1 1] [3 -3 0] [4 -1 2]]]	TPM 2 weights after feedback: [[[2 3 -3] [2 4 -4] [2 3 -3] [3 5 -3] [2 4 -2]] [[1 0 -3] [3 1 -3] [3 0 -4] [1 0 -3] [3 1 -5]] [[4 -2 0] [2 -2 -1] [4 -1 0] [3 -3 -1] [4 -1 1]]]

This method of synchronization is based on the principle that the weights of two TPMs that produce the same output for the same input must be identical. By applying the same input to both TPMs and adjusting the weights of the second TPM to match the output of the first TPM, the two TPMs become synchronized and have identical weights.

This method of synchronization can be thought of as a form of unsupervised learning, where the TPMs learn to produce the same output for the same input without being explicitly told what the correct output should be. It is important to note, however, that this method of synchronization may not work in all cases and may require careful tuning of the parameters to achieve good results.

```
# Initialize TPMs with random weights
num_layers = 3
layer_size = 5
num_bits = 3
tpm1 = random_weights(num_layers, layer_size, num_bits)
tpm2 = random_weights(num_layers, layer_size, num_bits)

# Feedback function
def feedback(tpm1, tpm2, input):
    output1 = compute_output(tpm1, input)
    output2 = compute_output(tpm2, input)
    error = compute_error(output1, output2)

    if has_error(error):
        diff = compute_difference(output1, output2)
        adjust_weights(tpm2, input, diff)

    return output1, output2

# Synchronize TPMs using feedback
for i in range(100):
    input = random_input(num_bits)
    output1, output2 = feedback(tpm1, tpm2, input)

    print("Iteration", i)
    print("TPM 1 output:", output1)
    print("TPM 2 output:", output2)
    print("TPM 2 weights after feedback:", tpm2)

# Display results
display_results()
```

Algorithm: Pseudocode of implementing a key exchange mechanism using TPMs

8 Conclusion

We provided the code in Python that diverged from conventional training algorithms such as backpropagation for adjusting the weights of the Tree Parity Machines (TPMs). Instead, it employs a feedback mechanism for the synchronization of weights between two distinct TPMs. In each iteration of the loop, the code generates a random input vector and utilizes the feedback function to apply it to both TPMs. This feedback function computes the output vectors of both TPMs for the given input and compares them. If discrepancies arise in the output vectors, the function calculates the disparity and adjusts the weights of the second TPM accordingly. The loop runs until the output vectors of both TPMs achieve identical states.

This synchronization approach operates on the principle that two TPMs yielding the same output for identical inputs must possess identical weights. By subjecting both TPMs to the same input and adjusting the weights of the second TPM to align with the output of the first TPM, synchronization occurs, resulting in identical weights. This

synchronization method can be conceptualized as a form of unsupervised learning, where the TPMs learn to produce identical outputs for the same inputs without explicit guidance on the correct output.

The TPM-based key agreement protocol is an efficient method for secure key exchange between two parties. However, it is important to ensure that the TPMs are initialized with sufficiently large weights and that the number of iterations is appropriate to ensure the security of the generated key.

References

1. https://www.gsma.com/security/network-equipment-security-assurance-scheme/
2. Chakraborty, S., Dalal, J., Sarkar, B. and Mukherjee, D.: Neural Synchronization based secret key exchange over public channels: a survey. Arxiv.org. (2015). https://arxiv.org/ftp/arxiv/papers/1502/1502.05153.pdf
3. https://www.gsma.com/security/securing-the-5g-era/
4. https://www.gsma.com/security/5g-cybersecurity-knowledge-base/
5. Rivest, R.: The RC4 encryption algorithm, RSA Data Security (1992)
6. Zhou, X., Tang, X.: Research and Implementation of RSA Algorithm For Encryption And Decryption - IEEE Conference Publication. Ieeexplore.ieee.org. (2011). https://ieeexplore.ieee.org/abstract/document/6021216
7. Revankar, P., Gandhare, W., Rathod, D.: Private inputs to tree parity machine. Pdfs.semanticscholar.org. (2010). https://pdfs.semanticscholar.org/dfbc/7fde64f2b55d8767daa50e9ea2130bf6db69.pdf
8. Kinzel, W., Kanter, I.: Neural cryptography. In: Proceedings of the 9th International Conference on Neural Information Processing. ICONIP 2002, vol. 3, pp. 1351–1354. Singapore (2002). https://doi.org/10.1109/ICONIP.2002.1202841
9. Kinzel, W., Kanter, I.: Interacting neural networks and cryptography. Adv. Solid State Phys. 383–391 (2003)
10. Salguero Dorokhin, É., Fuertes, W., Lascano, E.: On the development of an optimal structure of tree parity machine for the establishment of a cryptographic key. Secur. Commun. Networks 1–10 (2019)
11. Dolecki, M., Kozera, R.: The impact of the TPM weights distribution on network synchronization time. Comput. Inform. Syst. Indust. Manage. 451–460 (2015)
12. L. Mirtskhulava, N., Gulua, N., Meshveliani: Iot security analysis using neural key exchange. GESJ: Computer Science and Telecommunications |No.2 (57) (2019)
13. L. Mirtskhulava, N., Gulua, N., Meshveliani.: Ntru cryptosystem analysis for securing IoT. GESJ: Computer Science and Telecommunications |No.1 (56) (2019)
14. Shishniashvili, E., Mamisashvili, L., Mirtskhulava, L.: Enhancing IoT security using multilayer feedforward neural network with tree parity machine elements. Int. J. Simul. Syst. Sci. Technol. **21**(2), 371–383 (2020). https://doi.org/10.5013/ijssst.a.21.02.37
15. Mirtskhulava L.N. Meshveliani, N., Gulua, Globa, L.: Cryptanalysis of Internet of Things (IoT) wireless technology. In: IEEE International Conference on Information and Telecommunication Technologies and Radio Electronics – UkrMicO. Odessa, Ukraine (2019)
16. Mirtskhulava, L., GlobaL, L., Gulua, N., Meshveliani N.: Complex approach in cryptanalysis of Internet of Things (IoT) using blockchain technology and lattice-based cryptosystem. In: Ilchenko, M., Uryvsky, L., Globa, L. (eds.) Advances in Information and Communication Technology and Systems. MCT 2019. NNS, vol. 152. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-58359-0_4
17. Klimov, A., Mityagin A., Shamir: Analysis of Neural Cryptography, Springer.com (2002). https://doi.org/10.1007/3-540-36178-2_18