applied sciences

IMPACT FACTOR 2.5

CITESCORE 5.5

Article

Enhancing Security of Error Correction in Quantum Key Distribution Using Tree Parity Machine Update Rule Randomization

Bartłomiej Gdowski, Miralem Mehic and Marcin Niemiec

Special Issue

Innovations in Artificial Neural Network Applications

Edited by

Dr. Iztok Peruš and Prof. Dr. Milan Terčelj

MDPI

https://doi.org/10.3390/app15147958

applied sciences

MDPI

Article

Enhancing Security of Error Correction in Quantum Key Distribution Using Tree Parity Machine Update Rule Randomization

Bartłomiej Gdowski [1], Miralem Mehic [2] and Marcin Niemiec [1,*]

[1] Faculty of Computer Science, Electronics and Telecommunications, AGH University of Krakow, Mickiewicza 30, 30-059 Krakow, Poland; bgdowski@agh.edu.pl
[2] Department of Telecommunications, Faculty of Electrical Engineering, University of Sarajevo, Zmaja od Bosne bb, 71000 Sarajevo, Bosnia and Herzegovina
* Correspondence: niemiec@agh.edu.pl; Tel.: +48-12-617-4803

Abstract

This paper presents a novel approach to enhancing the security of error correction in quantum key distribution by introducing randomization into the update rule of Tree Parity Machines. Two dynamic update algorithms—`dynamic_rows` and `dynamic_matrix`—are proposed and tested. These algorithms select the update rule quasi-randomly based on the input vector, reducing the effectiveness of synchronization-based attacks. A series of simulations were conducted to evaluate the security implications under various configurations, including different values of `K`, `N`, and `L` parameters of neural networks. The results demonstrate that the proposed dynamic algorithms can significantly reduce the attacker's synchronization success rate without requiring additional communication overhead. Both proposed solutions outperformed `hebbian`, an update rule-based synchronization method utilizing the percentage of attackers synchronization. It has also been shown that when the attacker chooses their update rule randomly, the `dynamic` approaches work better compared to `random walk` rule-based synchronization, and that in most cases it is more profitable to use `dynamic` update rules when an attacker is using `random walk`. This study contributes to improving QKD's robustness by introducing adaptive neural-based error correction mechanisms.

Keywords: quantum cryptography; artificial neural networks; tree parity machines

Academic Editors: Iztok Peruš and Milan Terčelj

Received: 11 June 2025
Revised: 5 July 2025
Accepted: 14 July 2025
Published: 17 July 2025

Citation: Gdowski, B.; Mehic, M.; Niemiec, M. Enhancing Security of Error Correction in Quantum Key Distribution Using Tree Parity Machine Update Rule Randomization. *Appl. Sci.* **2025**, *15*, 7958. https://doi.org/10.3390/app15147958

1. Introduction

Since the early days of communication, humans have developed various techniques to secure transmitted information. Starting with the Skytale cipher, through Enigma machines, to the wide application of modern public-key cryptography such as the Rivest–Shamir–Adleman (RSA) algorithm, each approach reflects the technological capabilities of its era; while historical methods relied on the secrecy of the algorithm or mechanical complexity, modern cryptographic systems derive their security from mathematical principles and the computational difficulty. These systems, though secure for their time, share one trait: their resistance is ultimately based on the computational difficulty of solving certain mathematical problems, such as prime factorization. As computing technology evolves, particularly with the advent of quantum computing, the limitations of classical cryptography become increasingly evident.

For nearly 10 years, the RSA key length recommended by the National Institute of Standards and Technology (NIST) has been 2048 bits [1]. Although breaking modern encryption remains theoretically impossible for classical machines, quantum algorithms such as the Shor algorithm pose a serious threat. According to a commonly held estimate, with today's computer resources, it would take 300 trillion years to break an RSA-2048 encryption key. According to the Center for Strategic and International Studies, a powerful quantum computer will break the same key in 10 s [2]. While such a quantum machine does not yet exist, the field is advancing rapidly. At the time of writing, the most powerful quantum computer has been created by Atom Computing–with a capacity of 1225 qubits and population of 1180 qubits [3]–and IBM plans to release a 2000-qubit machine in 2033 or later [4], so viable quantum decryption capabilities can be only a matter of time.

This approaching threat to our security mechanisms has driven the development of post-quantum cryptography and quantum-safe communication models [5]. Among the most promising approaches is Quantum Key Distribution (QKD), which leverages the fundamental principles of quantum mechanics to securely exchange encryption keys. However, QKD itself still depends on classical procedures, such as error correction and privacy amplification, which must also be resistant to quantum adversaries. One emerging solution is to combine quantum cryptography with adaptive mechanisms from machine learning. Artificial Neural Network (ANN) structures, and in particular Tree Parity Machine (TPM), offer a way to perform secure key synchronization and error correction without explicitly transmitting sensitive key information. However, as with every approach to security, neural-based cryptography is not invulnerable: if an attacker can predict the internal parameters of the communication, the synchronization becomes endangered.

In this context, our work proposes a novel enhancement to TPM-based synchronization: dynamic selection of the update rule, driven by the properties of the input vector. This randomized approach increases the attacker's uncertainty while preserving performance and compatibility with existing QKD infrastructures. The proposed solution represents a significant step toward improving the robustness of quantum cryptographic systems in the face of realistic adversaries.

2. Overview

2.1. Quantum Cryptography

The motivation for using cryptography can generally be divided into two key objectives. The first is to ensure the confidentiality, meaning that the content of the communication remains inaccessible to unauthorized parties. The second is to guarantee the authenticity, meaning that the communicating parties are able to verify that the message has been originated from a legitimate source. The simplest way to achieve this is to use a secret key known only to the sender and the receiver. However, with conventional communication methods, it is nearly impossible to guarantee that no third party has accessed any part of the key. Quantum cryptography addresses this problem by providing a more secure way to detect whether data has been eavesdropped [6,7].

Quantum cryptography relies on two types of communication channels [8]: a pre-authenticated quantum channel, used for the secure distribution of quantum bits used to build cryptographic keys, and a public channel (which also should be pre-authenticated by the communicating parties), used for error correction and the exchange of additional information. In general, quantum cryptography consists of two main phases: quantum key distribution based on communication in quantum channel and key distillation.

2.1.1. Quantum Key Distribution

Quantum Key Distribution (QKD) refers to the process of establishing cryptographic keys via a quantum channel. QKD protocols can be categorized based on the type of quantum particles they use. The first category–for example, the BB84 [9] and B92 [10] protocols–uses independent single qubits. The second category–for example, the E91 [11] protocol–operates on pairs of qubits with entangled states, which means that the measurement of one qubit instantaneously determines the state of the other [12].

The security of QKD relies on two fundamental principles of quantum mechanics: Heisenberg's uncertainty principle and the no-cloning theorem. The uncertainty principle [13] states that certain pairs of related physical properties cannot be simultaneously measured with arbitrary precision. In other words, this means that any attempt to measure a quantum state inevitably changes it, which may help detect that an adversary tries to intercept the communication [6,14]. The no-cloning theorem states that unknown quantum states cannot be copied [15]. This principle is crucial because it makes passive eavesdropping on the quantum key virtually impossible. Although an attacker can access the quantum channel, they cannot simply observe the qubits without altering their state. As a result, any eavesdropping attempt becomes detectable, limiting the attacker to actions such as slowing down or intercepting the communication rather than remaining undetected.

2.1.2. Key Distillation

Public channels are the second medium of communication between both communicating parties. Unlike the quantum channel, the public channel does not protect information from eavesdropping by default. Therefore, it is used only for transferring non-confidential data to perform the key distillation. There are three main stages of key distillation [14]: Quantum Bit Error Rate (QBER) estimation, error correction, and privacy amplification.

An attempt to eavesdrop quantum communication is not the only source of errors in the quantum channel. In fact, multiple factors can affect the transmission of qubits, such as disturbances in the channel, noise in the detectors, or environmental decoherence [16]. As a result, the key is never 100% accurate, and both parties must estimate the QBER value. This value is typically a few percent [17,18], and its estimation can be performed by comparing a portion of the raw key over the public channel [19]. If the QBER exceeds an agreed-upon threshold, it may indicate an eavesdropping attempt or excessive channel noise, rendering the key distribution unreliable. The portion of the key used for comparison should be removed from the final key to minimize the potential information gained by an attacker. The length of this compared segment must be chosen carefully: if too short, QBER estimation may be inaccurate; if too long, a substantial number of bits would be discarded, significantly reducing the final key length.

After QBER estimation, the next step is error correction. A popular solution is the parity check method, such as in BBBSS [20] or in the solution presented by Jennewein, et al. [21]. In this method, the key is divided into blocks and the parity of each block is compared through the public channel. Blocks with matching parities are kept in the key after discarding one bit (so the supposed attacker loses information about the parities of specific blocks). The length of the final key depends on the number of iterations–many iterations can significantly shorten the final key. To keep the final key length as close as possible to the initial received key length, a more efficient protocol such as Cascade [22] is necessary. Another error correction approach–first mentioned in [23]–uses ANNs with initial weights based on the bit strings transmitted and received over the quantum channel. This solution will be analyzed in this paper.

The final stage of key distillation is privacy amplification. The most widely used method involves applying a one-way hash function that converts a string of bits into

a shorter key [24,25]. Although the hash function is publicly known, its information is essentially useless to anyone other than the communicating parties, as it is extremely difficult (though not impossible [26]) to reverse a hash function. Other types of privacy amplification include privacy amplification by sampling [27] and privacy amplification by iteration [28], although these solutions are not widely adopted. The current research is increasingly focused on eliminating this phase. One example, mentioned previously, is key reconciliation using ANNs, which is secure enough to make privacy amplification unnecessary. However, even in this case, an additional phase can still be applied to enhance security.

2.2. Artificial Neural Networks

An Artificial Neural Network (ANN) is a system that simulates the operation of the brain, in terms of learning ability, to solve computational problems that typically require human intervention. The ANN consists of artificial neurons that–just as biological neurons–receive a signal, process it, and send the processed signal forward. The neural network model was first introduced by McCulloch and Pitts in 1943 [29], and since then, the concept has been widely used to perform estimations based on a large number of inputs. An ANN typically consists of three types of layers. The first layer, called the input layer, initiates the computing process by receiving an input from the user. The information is then transferred and processed to the second layer, known as the hidden layer. An ANN can have either zero [30] or multiple hidden layers. The connections between the input and hidden layers can follow different patterns, such as one-to-one, one-to-all, or any custom configuration defined by the user. The output layer is the final layer in the ANN–it returns the result(s) of the process [31].

2.2.1. Tree Parity Machine

A specific type of ANN, Tree Parity Machine (TPM), was first introduced in [32]. TPM specifically has only one hidden layer and one output, usually called τ *(tau)*. The total number of input neurons is equal to $\mathtt{N} \times \mathtt{K}$, where $\mathtt{N}$ is the number of neurons connected to each hidden layer neuron, and $\mathtt{K}$ is the number of neurons in the hidden layer. An example TPM with 3 hidden neurons and 3×4 input neurons is presented in Figure 1. Each connection between the input and hidden layers has a parameter called *weight*. The weight values are discrete and are chosen from the interval set by the $\mathtt{L}$ value: $[-\mathtt{L}, \mathtt{L}]$. The input values (denoted as x) are either -1 or 1 [33].

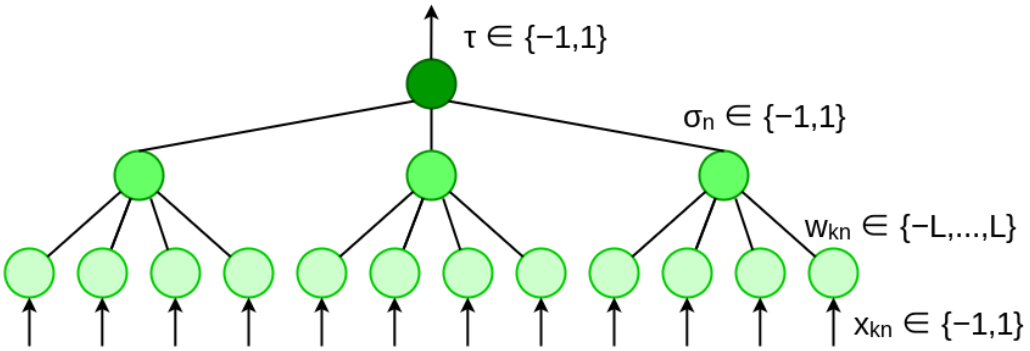


Figure 1. Tree parity machine with 3 hidden neurons and 3×4 input neurons.

The learning process of the TPM begins with computing the output values of the hidden layer neurons. The value for the k-th neuron is calculated using the formula presented in Equation (1) [23]:

$$\sigma_k = \text{sgn}\left(\sum_{n=1}^{N} x_{kn} * w_{kn}\right) \tag{1}$$

where $n \in \{1...N\}$, $k \in \{1...K\}$, x is the value of the n-th input neuron of the k-th hidden neuron, w is the weight value on the connection between these neurons, and the signum function is calculated as shown in Equation (2):

$$\text{sgn}(z) = \begin{cases} -1 & z \leq 0 \\ 1 & z > 0 \end{cases}. \tag{2}$$

The final output is then calculated based on the previously computed hidden layer outputs, as shown in Equation (3):

$$\tau = \prod_{k=1}^{K} \sigma_k \tag{3}$$

where $k \in \{1...K\}$. Based on Equations (1) and (3), it can be concluded that the final output value of the TPM is either -1 or 1.

2.2.2. TPMs in Cryptography

The characteristics of TPMs make them suitable for the implementation of the neural key exchange protocol proposed in [34]. Although the term "neuro-cryptography" has been introduced first in [35] in the context of using a model of perceptrons with back-propagation of the gradient, the term "neural cryptography" is most widely used for TPM applications in cryptography. This term for the neural key exchange protocol implemented with TPMs has been described in [36]. In this protocol, the communicating parties start with a random key, which is used as weights of the TPM. Using a channel that does not need to be secure, they begin to exchange input vectors (in the form of `K` random vectors of length `N`) for their machines. Depending on the outputs, they either update the weights (if the τ values are equal) or change the input vector (if the values differ). The process is repeated until both weight vectors are synchronized. An example formula for verifying synchronization has been provided in [37].

There are multiple learning rules used in neural cryptography. The most popular are [33]: the `hebbian` rule (see Equation (4)), the `anti-hebbian` rule (see Equation (5)) and the `random walk` rule (see Equation (6)).

$$w_{kn}^{\star} = w_{kn} + x_{kn} * \sigma_k * \Theta(\sigma_k, \tau), \tag{4}$$

$$w_{kn}^{\star} = w_{kn} - x_{kn} * \sigma_k * \Theta(\sigma_k, \tau), \tag{5}$$

$$w_{kn}^{\star} = w_{kn} + x_{kn} * \Theta(\sigma_k, \tau), \tag{6}$$

The variables have the same values as in Equation (1) and the Θ *(theta)* function is calculated as presented in Equation (7).

$$\Theta(\sigma_k, \tau)) = \begin{cases} 0 & \text{if } \sigma_k \neq \tau \\ 1 & \text{if } \sigma_k = \tau \end{cases}. \tag{7}$$

Furthermore, to ensure that the updated weights do not exceed the {−L;L} interval, invalid values can be corrected with a function similar to the one presented in Equation (8) [23].

$$v_L(w_{kn}) = \begin{cases} -L & \text{if } w_{kn} \leq -L \\ w_{kn} & \text{if } -L < w_{kn} < L \\ L & \text{if } w_{kn} \geq L \end{cases} . \quad (8)$$

The benefit of neural cryptography is that the secret key is never transferred, even in small parts, through the public channel. Additionally, if the attacker has no knowledge of the TPM parameters (K, N, L, and the update rule), there are no feasible methods of attack apart from setting up multiple TPMs. Unfortunately, if the attacker has sufficient resources and establishes multiple learning networks (for example, in a genetic attack [33]), they still have a reasonable chance of success if the parameter values are not large enough. On the other hand, both parties in the communication process must know the parameters of the TPM, and if these are transmitted through an open channel, the attacker can set up one or more machines with the same parameter values, increasing their chances of obtaining the secret key.

When considering an eavesdropping attack with a single TPM, even though all three machines may start from a random vector, the attacker's synchronization is always slower than the synchronization between the sender and receiver. This is because there are only three possible events during the synchronization process [23].

- $\tau_s \neq \tau_r$–in this case, τ_a does not matter, as the sender and receiver do not update their weights.
- $\tau_s = \tau_r \neq \tau_a$–in this case, the sender and receiver update their weights, but the attacker does not.
- $\tau_s = \tau_r = \tau_a$–this is the only case in which the attacker's weights are updated.

In other words, the attacker's output must be equal to both the sender's and receiver's outputs (which means that these two values must also be equal) for the attacker to perform an update of the weights. This implies that the attacker's machine synchronizes slower than the other two machines, as there is no possibility of the attacker's update without the sender and receiver updating at the same time. This attack may have a better chance of success if there are multiple attacking TPMs, but security can be improved by increasing the L value [33]. Although there are other types of attacks that perform synchronization even if the third event from the list above occurs (such as the genetic attack or flipping attack [38]), in these cases, the attacker's synchronization is also slower compared to the synchronization between the sender and receiver.

2.2.3. TPMs in Quantum Cryptography Error Correction

As mentioned in the previous section, the security of neural cryptography is highly dependent on the attacker's knowledge of the TPM parameters. The effectiveness of an attack can be reduced if the sender and receiver synchronize from initially similar keys. However, how can pre-synchronization be done without sending the key through a public channel? Here, quantum cryptography fits perfectly. It was mentioned in Section 2.1.2 that the QBER value is usually a few percent. Considering the fact that the attacker cannot gain significant knowledge of the key from the quantum channel without the sender and receiver knowing, TPMs can be used for error correction. In this way, the networks can be initialized with a raw key translated into weights, and synchronization would be much faster, as the weights would already be pre-synchronized. At the same time, the attacker needs to start the synchronization from a random vector, so their probability of success is lower compared to neural cryptography.

Quantum cryptography, when implemented with ANNs, also benefits in terms of the final key length, as error correction by TPMs does not require removing bits. An algorithm for error correction based on TPM was proposed in [23], and the author states that although privacy amplification is not compulsory, it is still recommended in the future for new attack methods on TPMs in the future. ANN-based synchronization of quantum keys as a solution stands out in terms of the speed of synchronization and the general security of communication compared to either quantum or neural cryptography alone. However, it is important to identify how the values of the `K`, `N`, and `L` parameters affect the overall security of synchronization. Based on the value of `L`, the maximum number of bits that the attacker can obtain after the i-th iteration can be calculated as in Equation (9).

$$Z = \log_{(2L+1)} 2^i \tag{9}$$

The impact of `K` and `N` is somewhat related, as both values affect the total number of input neurons. In general, the longer the key, the more chances the attacker will have to synchronize their key. However, at the same time, it becomes harder for the attacker to synchronize larger numbers of weights because of starting from a random vector.

3. Materials and Methods

3.1. Methodology

As previously mentioned, the more data related to the TPMs used in the synchronization that the attacker is able to obtain, the higher their chances of success. Due to this, the authors propose a novel approach: instead of relying on a single synchronization rule, the rule is selected quasi-randomly, based on the number of ones present in the input vector. This method introduces additional uncertainty for a potential attacker who is then faced with a choice: either to select a rule at random or to consistently apply a fixed update rule. In scenarios where the communicating parties alternate between different update algorithms, an attacker adhering to a static rule risks desynchronizing their machine, potentially decreasing rather than increasing their machine's synchronization.

In this paper, the following two new approaches are introduced:

- `dynamic_rows`,
- `dynamic_matrix`.

Both update algorithms utilize the values generated as the input vector (which consists of -1's or 1's). This strategy leverages data already known to both communicating parties, without requiring the transmission of any additional information over the public channel. Each algorithm selects either the `hebbian` or `random walk` rule for a specified vector. Although these algorithms operate in a similar manner, they differ in the resulting weight distribution, which affects the number of updates required for synchronization [39,40].

The generalized pseudocode for both methods is as follows:

```
Initiate function with vector V.
If number of 1's in V > floor of (V length/2):
    Update part of the key corresponding to V using hebbian rule.
Else:
    Update part of the key corresponding to V using random walk rule.
```

where: `V` is the chosen vector (which may be a part or the entirety of the input vector). and `floor` function rounds the value to the largest integer that is less than or equal to that value.

The key difference between `dynamic_rows` and `dynamic_matrix` algorithms lies in the granularity of their application. In case of the latter algorithm, the entire input vector is used as `V`, resulting in a single update rule applied per iteration of synchronization. In the

first algorithm, each subtree corresponding to a hidden layer neuron is treated as a separate `V`. Therefore, in that case, the algorithm is chosen `K` times per iteration of synchronization, each time based on a segment of the input vector of length `N`.

3.2. Testbed

The simulations included in this research paper have been conducted with a publicly available tool, which was modified according to predefined requirements. All test results presented within this research have been obtained using a personal computer with an Intel Core i5-1335U CPU, 16 GB of RAM memory, and Windows 11 Pro 64-bit operating system. All scripts have been executed using the Python programming language, version 3.11.

To perform the tests, the authors used a *NeuralKey* tool [41]. This allows the synchronization process between two TPMs with specified values of `K`, `N`, and `L` to be simulated. The update rule (referred to as the update algorithm from now on) can also be selected–all of the algorithms covered in Section 2.2.2 (`hebbian`, `anti-hebbian`, or `random walk`) are available. During synchronization, an attack attempt is carried out–the attacker's machine tries to synchronize with both the sender's and receiver's machines (referred to as victims from now on). In other words, this program allows simulation of the neural cryptography process. The output of *NeuralKey* consists of the following:

- The number of updates between synchronizing TPMs,
- The percentage synchronization value between the attacker's and victims' machines,
- The number of attacker's TPM updates.

To simulate quantum cryptography, the tool was modified. The modified version (called *QuantumKey* from now on) allows setting a specific QBER for the victims' machines, as well as allows one to choose new, aforementioned update algorithms. Additionally, the modifications enable to perform multiple iterations at once, ensuring that the results are not merely anomalies. *QuantumKey* is executed with parameters corresponding to the following variables:

- `K` value,
- `N` value,
- `L` value,
- `QBER` value in %,
- Chosen update algorithm,
- Attacker's update algorithm (optional, used in case of new, dynamic update algorithms).

3.3. Parameters

The simulations conducted for this research can be split into two parts: in the first part of tests, the `K` value is gradually increasing (`{5,7,9,11}`, and in the second part, the `N` value is increasing (`{5,13,34,89}`). All tests are executed with `L` value from the range: `{2,4,6}` and `QBER` value from the range `{3,5,7}`, which corresponds to the following percentage of synchronization of victims' machines at the start of the synchronization: `{97%,95%,93%}`. The `QBER` values that have been chosen were selected to reflect a realistic synchronization process in quantum cryptography, where QBER is typically a few percent (as mentioned in Section 2.1.2).

The tests for each set of parameters have been repeated with the following update algorithms:

- All parties: `hebbian`;
- Victims: `dynamic_rows`, attacker: `hebbian`;
- Victims: `dynamic_matrix`, attacker: `hebbian`;
- All parties: `random walk`;

- Victims: `dynamic_rows`, attacker: `random walk`;
- Victims: `dynamic_matrix`, attacker: `random walk`;
- Victims: `dynamic_rows`, attacker: `random rule`;
- Victims: `dynamic_matrix`, attacker: `random rule`.

Each test has been repeated 500 times. Part of the results is presented in the paper, while all of the results, as well as the source code, are available in a dedicated GitHub repository [42]. The values used for analysis were the total number of updates between the victims and attackers' synchronization percentage (as well as the 68% confidence intervals of these values). Throughout the paper, the leading zeros of the values have been omitted.

Although only some of the test results will be presented in this work, an analysis of all of the results has been conducted based on several questions.

- In how many cases did the `dynamic` algorithms result in the attacker retrieving less information when the attacker chose the `hebbian` rule?
- In how many cases did the `dynamic` algorithms result in the attacker retrieving less information when the attacker chose the `random walk` rule?
- In how many cases did the `dynamic` algorithms result in the attacker retrieving less information when the attacker chose a random update rule at each iteration?

The algorithm-specific analysis includes a direct comparison to the baseline scenarios in which all parties utilized the same update algorithm. The final category of analysis, involving random rule selection by the attacker, has been further divided to include comparisons against both `hebbian`-only and `random walk`-only reference cases.

During the analysis, the upper limits of the confidence intervals for the `dynamic` algorithms' results were compared to the lower boundaries of the confidence intervals for the corresponding standard algorithms. For example, if the percentage of synchronization for a `dynamic` algorithm was 63.52 ± 0.3, and for a standard algorithm it was 65.09 ± 0.31, the comparison was made between 63.82 (i.e., $63.52 + 0.3$) and 64.78 (i.e., $65.09 - 0.31$). Additionally, for convenience in presentation, a one-hundredth value has been added to the largest partial result to ensure that the entire pie chart sums to exactly 100%.

4. Results

4.1. Scenario 1

The first part of the tests has been conducted with the following parameter values:

- `K`: `{5,7,9,11}`,
- `N`: `{5}`,
- `L`: `{2,4,6}`,
- `QBER`: `{3,5,7}`

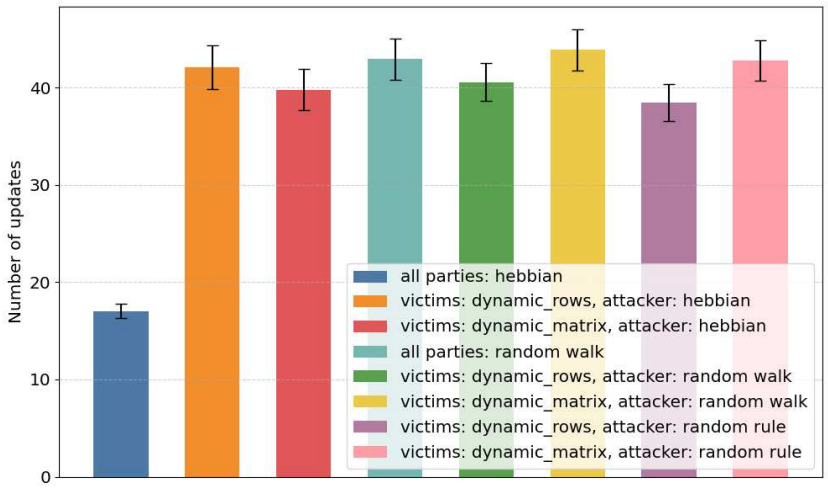
Which resulted in 36 groups of tests with combinations of update algorithms presented in Section 3.3.

4.1.1. First Example Test

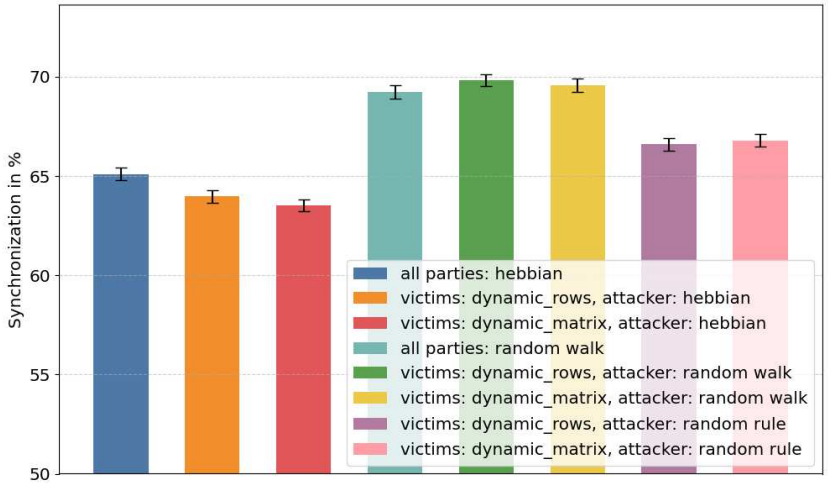
The results of one of the tests of Scenario 1 (with parameters: $K = 5, N = 4, L = 2$, $QBER = 3$), are presented in Table 1 and Figure 2. The number of victims' updates in this test–presented in the second column of Table 1 and in Figure 2a–is comparable in seven of the eight update rule combinations (≈41.5 on average), while for all parties using `hebbian`, it is more than two times lower (≈17). Although prolonged synchronization has been shown to increase the likelihood of a successful attack by the attacker [38,43,44], this effect is not observed in the current scenario. Specifically, when the attacker is using the `hebbian` update rule, the level of synchronization achieved–values are presented in the third column of Table 1 and in Figure 2b–is lower if the victims utilize either the

`dynamic_rows` or `dynamic_matrix` algorithm (≈64% and ≈63.5%, respectively), compared to the significantly shorter synchronization process when all parties use the `hebbian` rule (≈65%).

While this is not the case for the `random walk`-oriented tests, it is worth noting that the attacker's synchronization percentage in those tests (approximately 69.5% on average) was higher compared to the tests where the attacker used a random update rule at each iteration (approximately 67% on average). This indicates that the attacker's uncertainty regarding which algorithm is being used during the synchronization process may benefit the communicating parties. In particular, this advantage is evident when the attacker either fails to recognize that the update rule is being dynamically chosen or does not deploy multiple machines to attempt synchronization.



(**a**) Average number of victims' updates.



(**b**) Attacker's average synchronization in percent.

Figure 2. Scenario 1–$K = 5, N = 4, L = 2, QBER = 3$.

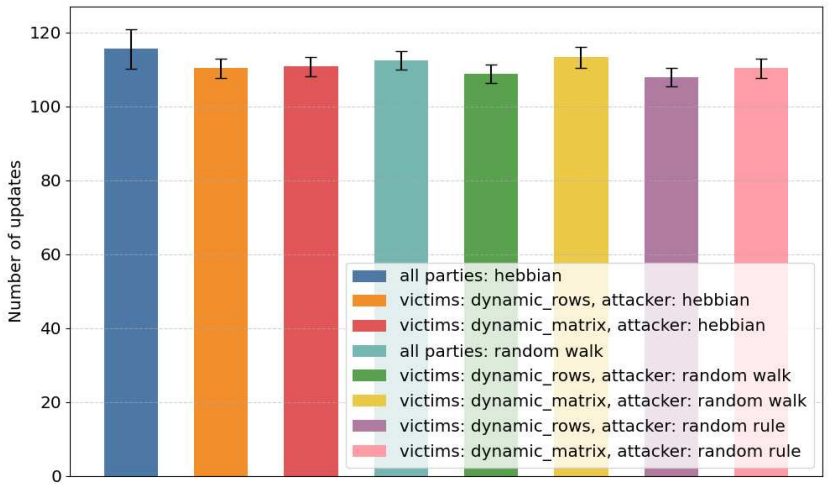
Table 1. Scenario 1–$K = 5, N = 4, L = 2, QBER = 3$.

Update Rules	Victims' Updates	Attacker's Synchronization
all parties: `hebbian`	17.02 ± 0.72	65.09 ± 0.31
victims: `dynamic_rows`, attacker: `hebbian`	42.10 ± 2.25	63.98 ± 0.31
victims: `dynamic_matrix`, attacker: `hebbian`	39.8 ± 2.15	63.52 ± 0.3
all parties: `random walk`	42.94 ± 2.11	69.24 ± 0.32
victims: `dynamic_rows`, attacker: `random walk`	40.56 ± 1.94	69.82 ± 0.31
victims: `dynamic_matrix`, attacker: `random walk`	43.91 ± 2.12	69.58 ± 0.32
victims: `dynamic_rows`, attacker: random rule	38.46 ± 1.91	66.6 ± 0.31
victims: `dynamic_matrix`, attacker: random rule	42.79 ± 2.09	66.79 ± 0.32

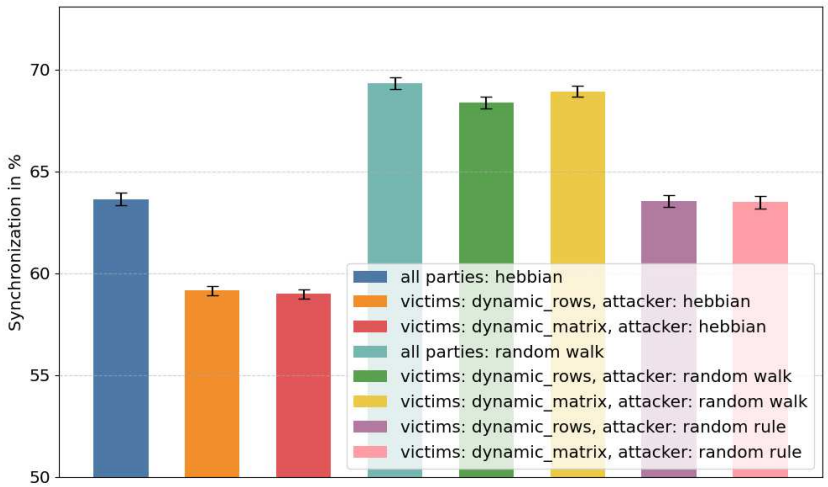
4.1.2. Second Example Test

The results of a different test from Scenario 1 (with parameters $K = 9, N = 4, L = 4, QBER = 7$) are presented in Table 2 and Figure 3. In the case of this test, compared to the previous one, the number of updates between victims–presented in the second column of Table 2 and in Figure 3a–is comparable across all combinations of update rules (≈111 on average). However, many of the previously described observations related to the synchronization percentage remain valid. Based on the data presented in the third column of Table 2 and in Figure 3b, we can conclude several statements.

- In `hebbian`-oriented tests, the attacker was able to obtain less information when the victims were using `dynamic_rows` or `dynamic_matrix` (≈59% in both cases), compared to the scenario in which all parties used the `hebbian` algorithm.
- The attacker's synchronization percentage is lower when they choose the update rule randomly, while the victims choose `dynamic_rows` or `dynamic_matrix` (≈63.5% in both cases), compared to any combination in which the attacker uses the `random walk` algorithm (≈69% on average). The values are also comparable with the attacker's synchronization when all parties use the `hebbian` algorithm (≈63.5%), although the confidence intervals are not narrow enough to definitely conclude which of these combinations is more beneficial for the victims.
- Additionally, compared to previously presented tests, in this case, the mean value of the attacker's synchronization was lower when the attacker used the `random walk` rule, and the victims used either `dynamic_rows` or `dynamic_matrix` (≈68% and ≈69%, respectively), compared to all parties using `random walk` (≈69%). However, in the case of `dynamic_matrix`, this observation is not conclusive, due to the width of the confidence intervals.



(**a**) Average number of victims' updates.



(**b**) Attacker's average synchronization in percent.

Figure 3. Scenario 1–$K = 9, N = 4, L = 4, QBER = 7$.

Table 2. Scenario 1–$K = 9, N = 4, L = 4, QBER = 7$.

Update Rules	Victims' Updates	Attacker's Synchronization
all parties: `hebbian`	115.62 ± 5.27	63.62 ± 0.1
victims: `dynamic_rows`, attacker: `hebbian`	110.32 ± 2.56	59.13 ± 0.25
victims: `dynamic_matrix`, attacker: `hebbian`	110.78 ± 2.60	58.97 ± 0.24
all parties: `random walk`	112.5 ± 2.56	69.31 ± 0.29
victims: `dynamic_rows`, attacker: `random walk`	108.8 ± 2.52	68.35 ± 0.29
victims: `dynamic_matrix`, attacker: `random walk`	113.28 ± 2.76	68.92 ± 0.27
victims: `dynamic_rows`, attacker: random rule	107.93 ± 2.6	63.55 ± 0.29
victims: `dynamic_matrix`, attacker: random rule	110.35 ± 2.56	63.48 ± 0.31

4.1.3. Comparative Analysis

To provide a broader perspective on the impact of proposed solutions on the security of synchronization, a comparative analysis was conducted. Both scenarios in which victims employ the proposed `dynamic` algorithms and scenarios in which all parties use the same standard update rule were considered. This comparison highlights the effectiveness of the `dynamic` approaches in limiting the attacker's ability to synchronize with the communicating parties. The results are presented in Figure 4.

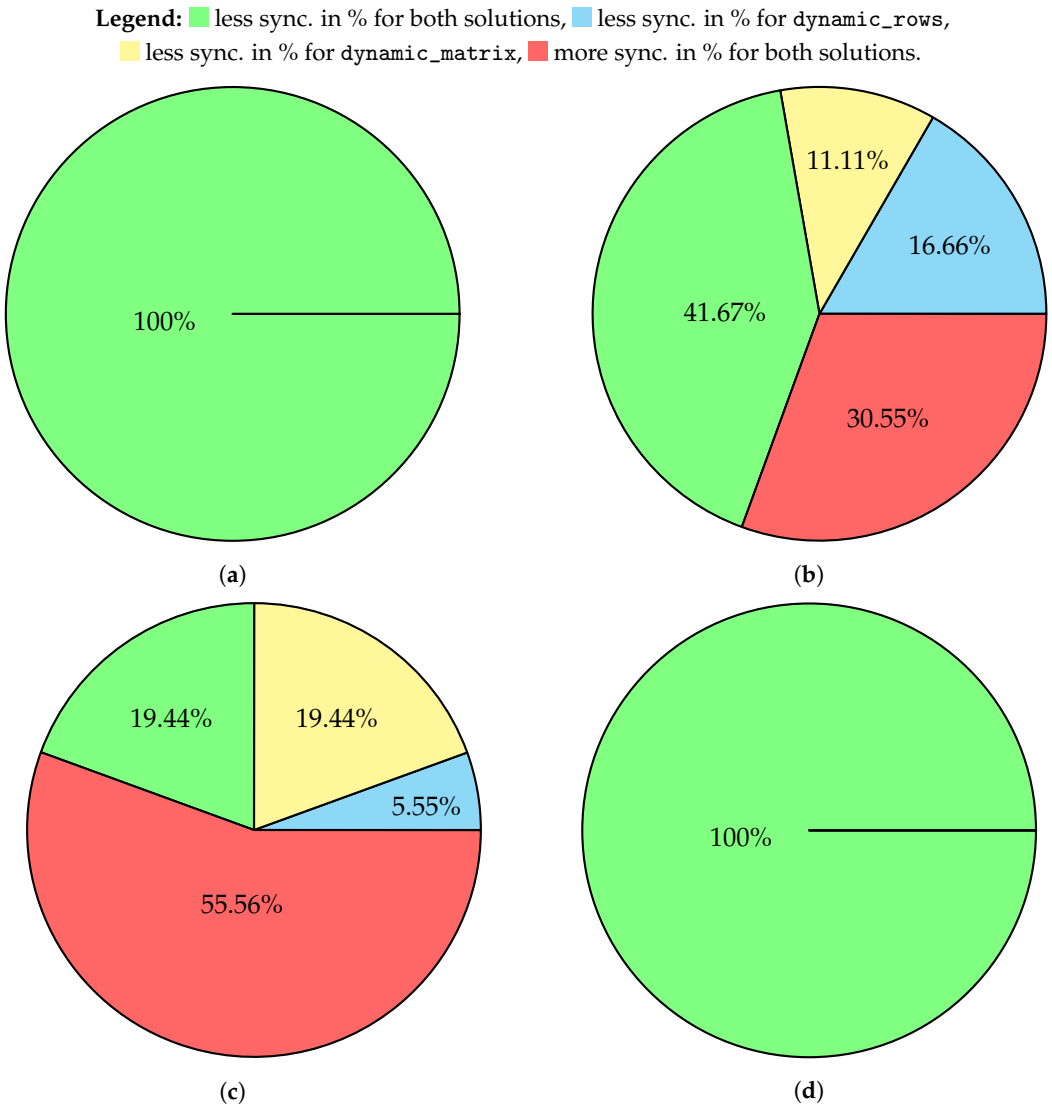


Figure 4. Attacker's synchronization percent using different update rules with victims using proposed solutions compared to all parties using the same update rule–Scenario 1. (**a**) Only attacker using `hebbian`versus all parties using `hebbian`; (**b**) Only attacker using `random walk` versus all parties using `random walk`; (**c**) Attacker using random update rule versus all parties using `hebbian`; (**d**) Attacker using random update rule versus all parties using `random walk`.

The analysis consists of four parts, each corresponding to a different subset of tests. The first part, presented in Figure 4a shows that in all of the tests conducted within this scenario, the attacker achieved a lower percentage of synchronization when victims were using either `dynamic_rows` or `dynamic_matrix`, compared to the scenario where all parties were using the `hebbian` update rule. This observation is consistent with the results from the previously discussed tests. This leads to a conclusion that while `hebbian`-only synchronization may require fewer iterations or less time, the use of any of the proposed

dynamic algorithms provides improved communication security for the victims. Therefore, if the communicating parties suspect that the attacker might employ the hebbian update rule, it is always more secure to adopt a dynamic algorithm–regardless of the combination of the parameters described in Scenario 1.

A similar analysis on the random walk update rule is presented in Figure 4b. In this case, both dynamic algorithms outperformed all parties using the random walk scenario in only 41.67% of the tests. In 11.11% of the cases, only dynamic_matrix was more secure, while dynamic_matrix provides higher security in 16.66% of the cases. These results suggest that in most cases, employing either of the proposed dynamic algorithms yields better protection against an attacker utilizing the random walk rule. As such, even in less consistent scenarios, the use of dynamic algorithms remains a reasonable and often advantageous choice for enhancing communication security.

If we compare scenarios in which victims use dynamic approaches, perform significantly worse in the means of synchronization security, compared to all parties using the hebbian update rule. The results for this part of the analysis is presented in Figure 4c. Both dynamic approaches performed better in 19.44% of the cases, in the same percentage of scenarios only dynamic_matrix was better, while dynamic_rows outperformed other algorithms only in 2.77% of cases. As the mean synchronization of the attacker was lower when all parties used the hebbian update rule in 55.56% of the cases, it may be more advantageous to rely on that algorithm when there is a reasonable suspicion that the attacker may know about dynamic rule selection and they also dynamically select their update rule. It should also be noted that the observed decrease in performance of the dynamic solutions can be attributed to the fact that the hebbian rule is the fastest learning algorithm among the most widely adopted ones, as discussed in Section 2.2.2 [45].

In the scenario where the attacker randomly selects the update rule, they consistently achieved a lower percentage of synchronization when the victims used either of the dynamic approaches, compared to the scenario in which all parties used the random walk update rule. As illustrated in Figure 4d, both dynamic_rows and dynamic_matrix outperformed the uniform random walk synchronization in 100% of the evaluated cases. This observation supports the conclusion that, in situations where the attacker is suspected of dynamically selecting the update rule, the use of dynamic synchronization strategies by the victims provides a more secure synchronization process. This is an important observation, as the results indicate that regardless of whether the attacker is using a random update rule or the random walk algorithm–provided they are not utilizing the hebbian rule–the security of the synchronization process for the victims does not deteriorate. In such cases, the synchronization security remains at a standard level when victims also use the random walk rule, or it can be improved if either of the dynamic algorithms is used.

4.2. Scenario 2

The second part of the tests has been conducted with the following parameter values:

- K: {5},
- N: {5,13,34,89},
- L: {2,4,6},
- QBER: {3,5,7}

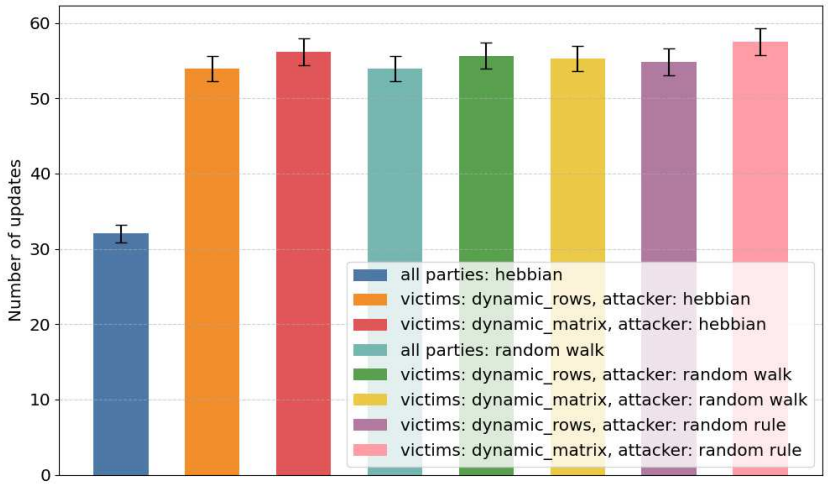
Which resulted in 36 groups of tests with combinations of update algorithms presented in Section 3.3.

4.2.1. First Example Test

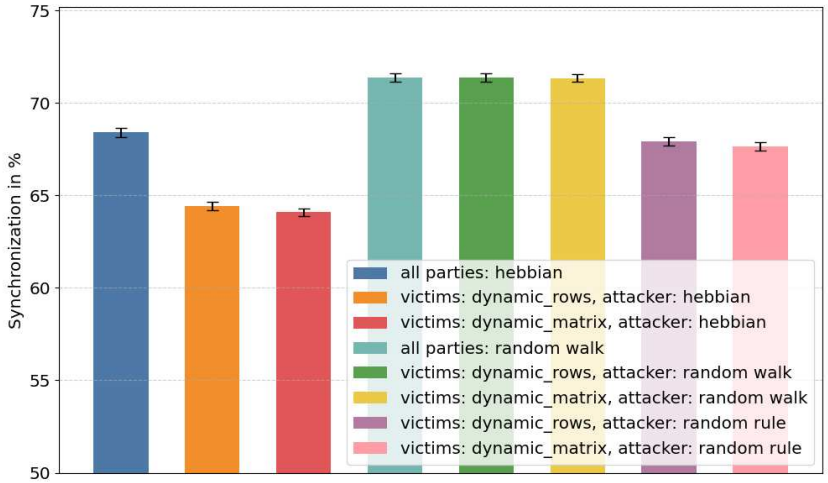
The results of one of the tests in Scenario 2 (with parameters: $K = 5, N = 13, L = 2, QBER = 3$) are presented in Table 3 and Figure 5. The number of victims' updates in

this test, presented in the second column of Table 3 and in Figure 5a, is–as in the first test of Scenario 1–comparable in seven of the eight combinations of the update rules (≈55 on average), while for all parties using `hebbian` its nearly two times lower (≈33). The level of synchronization of the attacker (in %) is presented in the third column of Table 3 and in Figure 5b. Similarly to the observation from the first test of Scenario 1, again, the lower number of updates did not result in a more secure synchronization process. In this case, the difference in the attacker's synchronization percentage between tests in which the victims utilize either the `dynamic_rows` or `dynamic_matrix` algorithm and the test where all parties use the `hebbian` rule is even larger (≈64% for both proposed solutions, versus ≈68%).

In case of `random walk`-oriented tests, the attacker's synchronization percentage was nearly identical in all three cases–71.39%, 71.38% and 71.35%. These values are much higher than the results achieved by the attacker when they use random update rule, and the victims use any of the proposed textttdynamic solutions–≈68%. As stated in the summary of the first test of Scenario 1, in this case, the communicating parties also benefit from attacker not knowing which update rule is used at every iteration.



(**a**) Average number of victims' updates.



(**b**) Attacker's average synchronization in percent.

Figure 5. Scenario 2–$K = 5, N = 13, L = 2, QBER = 3$.

Table 3. Scenario 2–$K = 5, N = 13, L = 2, QBER = 3$.

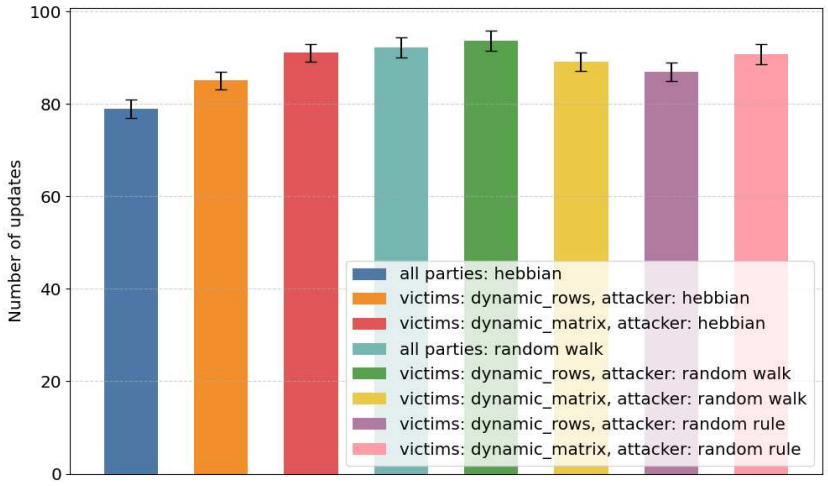
Update Rules	Victims' Updates	Attacker's Synchronization
all parties: `hebbian`	33.06 ± 1.19	68.41 ± 0.24
victims: `dynamic_rows`, attacker: `hebbian`	53.95 ± 1.69	64.42 ± 0.21
victims: `dynamic_matrix`, attacker: `hebbian`	56.16 ± 1.84	64.08 ± 0.21
all parties: `random walk`	53.95 ± 1.64	71.39 ± 0.23
victims: `dynamic_rows`, attacker: `random walk`	55.65 ± 1.71	71.38 ± 0.22
victims: `dynamic_matrix`, attacker: `random walk`	55.26 ± 1.69	71.35 ± 0.21
victims: `dynamic_rows`, attacker: random rule	54.88 ± 1.79	67.92 ± 0.26
victims: `dynamic_matrix`, attacker: random rule	57.5 ± 1.86	67.64 ± 0.23

4.2.2. Second Example Test

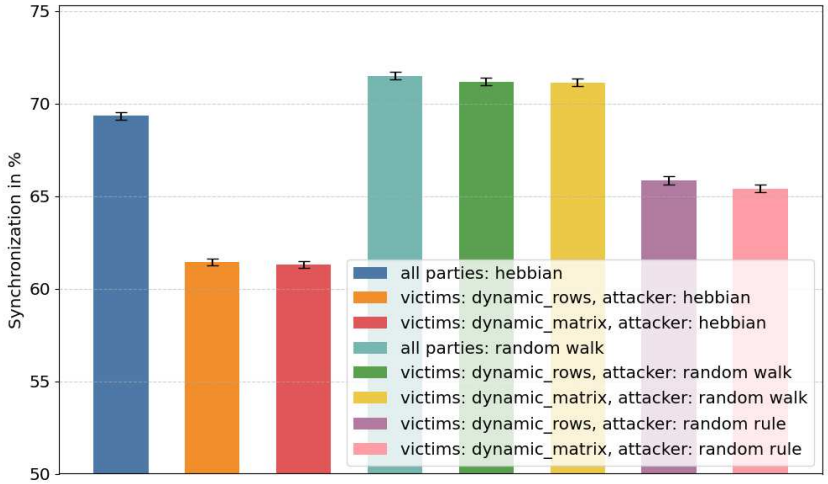
The results of a different test from Scenario 2 (with parameters: $K = 5, N = 34, L = 4, QBER = 7$) are presented in Table 4 and Figure 6. The results related to the average number of victims' updates in the second test from Scenario 2 are presented in second column of Table 4 and in Figure 6a. In this test, the trend continues, as the number of updates between the victims is much more comparable comparable across all rule combinations (≈88.5 on average). At the same time, the observations related to the attacker's average synchronization percentage–presented in third column of Table 4 and in Figure 6b–remain consistent: `dynamic` approaches provide a higher level of security compared to `hebbian`-only synchronization (≈61% compared to ≈69% of attacker's synchronization) and the attacker's synchronization percentage in all of the `random walk`-oriented tests is higher than in tests where victims were using `dynamic_rows` or `dynamic_matrix` while the attacker chose the update algorithm randomly (≈71.5% compared to ≈65%).

Table 4. Scenario 2–$K = 5, N = 34, L = 4, QBER = 7$.

Update Rules	Victims' Updates	Attacker's Synchronization
all parties: `hebbian`	78.94 ± 2.06	69.32 ± 0.22
victims: `dynamic_rows`, attacker: `hebbian`	85.05 ± 1.89	61.42 ± 0.18
victims: `dynamic_matrix`, attacker: `hebbian`	91.08 ± 1.95	61.28 ± 0.18
all parties: `random walk`	92.15 ± 2.17	71.5 ± 0.2
victims: `dynamic_rows`, attacker: `random walk`	93.71 ± 2.23	71.17 ± 0.2
victims: `dynamic_matrix`, attacker: `random walk`	89.12 ± 2.06	71.13 ± 0.21
victims: `dynamic_rows`, attacker: random rule	87.02 ± 2	65.85 ± 0.21
victims: `dynamic_matrix`, attacker: random rule	90.76 ± 2.17	65.41 ± 0.22



(**a**) Average number of victims' updates.



(**b**) Attacker's average synchronization in percent.

Figure 6. Scenario 2–$K = 5, N = 34, L = 4, QBER = 7$.

4.2.3. Comparative Analysis

A broader analysis of the impact of the proposed solutions on the security of synchronization, following the same methodology as in the previous section, is presented in Figure 7. The analysis is again split into four parts:

1. Comparison of scenarios in which all parties are using `hebbian` versus scenarios in which victims change their update algorithms to proposed solutions.
2. Comparison of scenarios in which all parties are using `random walk` versus scenarios in which victims change their update algorithms to proposed solutions.
3. Comparison of scenarios in which all parties are using `hebbian` versus scenarios in which victims change their update algorithms to proposed solutions and the attacker chooses their update rule randomly.
4. Comparison of scenarios in which all parties are using `random walk` versus scenarios in which victims change their update algorithms to proposed solutions and the attacker chooses their update rule randomly.

Most of the results are almost the same as in the analysis of Scenario 1. In fact, in the first part (presented in Figure 7a) they are the same–usage of either of the `dynamic` solutions resulted in attacker achieving a lower percentage of synchronization, compared to all parties using `hebbian` in all cases. The same applies to the fourth part of the analysis

(presented in Figure 7d)–when the attacker was choosing their update rule randomly (and the victims were using `dynamic` update rules), in all of the tests in this scenario, they were unable to synchronize their machines as much as in case where everyone was using `random walk`.

In the case of the second part of the analysis (presented in Figure 7b) the results were not identical, but very close to the results from Scenario 1. In Scenario 2, the percentage of tests in which both proposed solutions outperformed `random walk` synchronization has increased (50% of tests, compared to 41.67% in Scenario 1), percentage of tests in which only `dynamic_rows` was better and both proposed solutions were worse have decreased (from 16.66% to 11.11% and from 30.55% to 27.78% compared to Scenario 1, respectively), while percentage of tests in which only `dynamic_matrix` was better when it remained unchanged (11.11%).

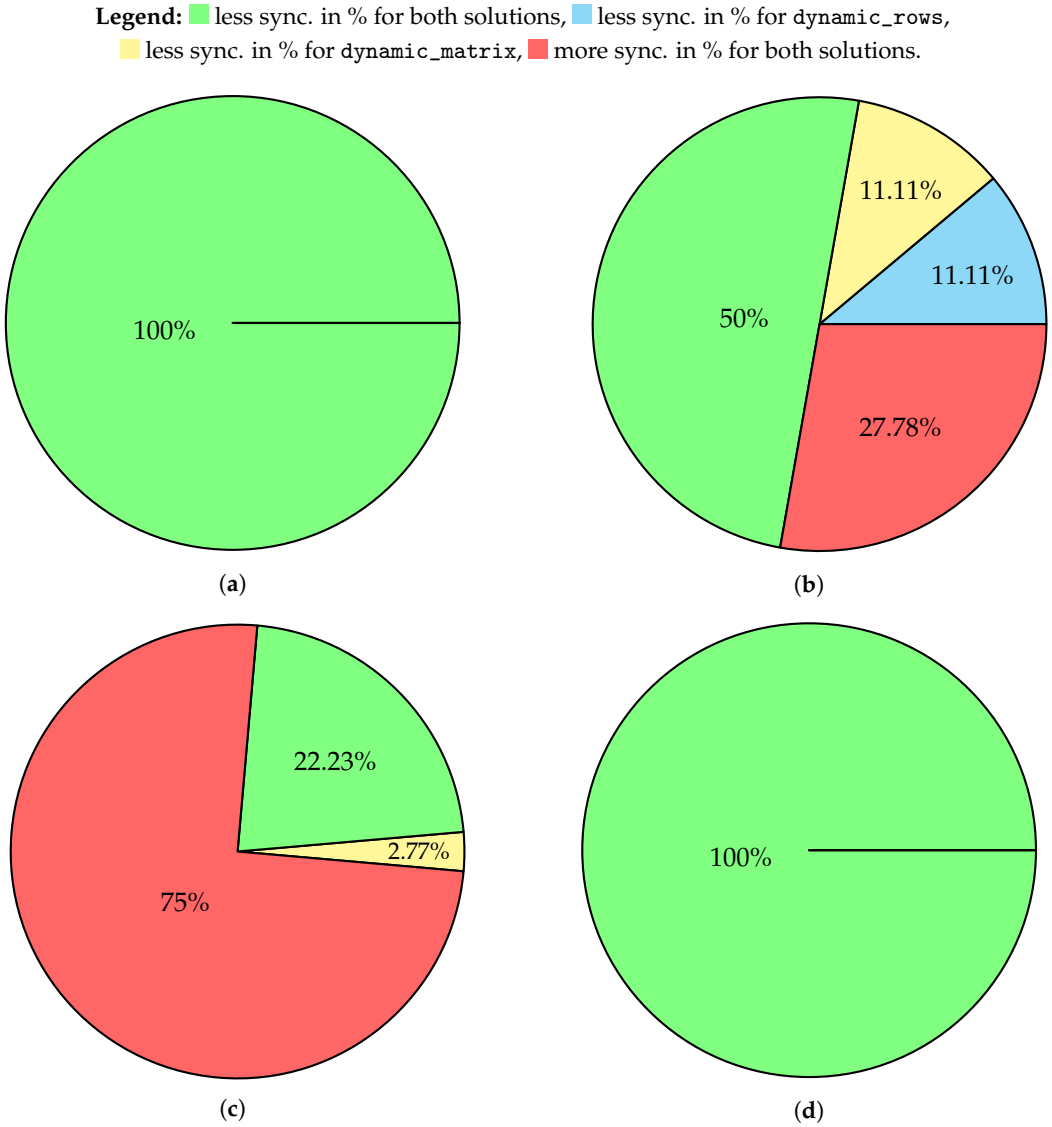


Figure 7. Attacker's synchronization percent using different update rules with victims using proposed solutions compared to all parties using the same update rule–Scenario 2. (**a**) Only attacker using `hebbian` versus all parties using `hebbian`; (**b**) Only attacker using `random walk` versus all parties using `random walk`; (**c**) Attacker using random update rule versus all parties using `hebbian`; (**d**) Attacker using random update rule versus all parties using `random walk`.

In Scenario 2, the proposed solutions performed significantly worse in the third part of the analysis (shown in Figure 7c). The percentage of cases in which both `dynamic`

approaches performed worse than the hebbian-only synchronization increased from 55.56% to 75%, while the percentage of cases in which both dynamic algorithms performed better increased slightly (from 19.44% to 22.23% compared to Scenario 1), the other metrics have notably decreased: from 19.44% to 2.77% for dynamic_matrix and from 5.55% to 0% for dynamic_rows.

In general, the outcomes for Scenario 2 are the same as for Scenario 1:

- If the victims have suspicion that the attacker is using hebbian, it is always better to use any of the dynamic approaches;
- If the victims have suspicion that the attacker is using random walk, in general dynamic_matrix or dynamic_rows are better (about 3/4 of all cases);
- If the attacker is using random update rule, dynamic algorithms perform worse compared to hebbian-only synchronization;
- If the attacker is using random update rule, it is always better to use dynamic update rule compared to all parties synchronizing with random walk.

5. Discussion

The results clearly indicate that introducing dynamic update rules in TPM-based synchronization improves resistance against various forms of attacks. In particular, when the attacker is unaware of the update rule strategy or incorrectly assumes a static rule (e.g., always hebbian or random walk), their ability to synchronize with the victims is significantly diminished.

To evaluate the robustness of the proposed approach under different conditions, a structured set of simulation scenarios was designed. In each scenario, different TPM parameter is increased, to both test out what is the influence of the key length and the hidden to input neuron ratio on the security achieved by dynamic_rows and dynamic_matrix algorithms, compared to standard update rules. To analyze how the length and structure of the cryptographic key, derived from TPM parameters, affect the system's security, two scenarios have been prepared. Scenario 1 focused on varying the number of hidden neurons (K) while keeping the number of input neurons per hidden neuron (N) constant. This setup tests how increasing the depth of the neural structure (i.e., the number of subtrees involved in synchronization) affects the resilience to attacks. In contrast, Scenario 2 kept the number of hidden neurons constant and varied the number of input neurons per hidden neuron, simulating the effect of extending each hidden neuron's input space, and in result increasing the entropy and granularity of the key. This dual approach allows to compare how different architectural dimensions of the TPM influence synchronization behavior and the effectiveness of dynamic update strategies against potential attackers.

While dynamic algorithms do not always outperform standard rules in all configurations, they consistently provide a better security-to-performance trade-off in scenarios involving attacker uncertainty. There are two main approaches–either the victims suspect that the attacker is using a single rule or that the attacker is choosing their rule randomly. In the first case, the dynamic update rules are either better in 100%–if the attacker is using hebbian–or in $\approx 3/4\%$ of the cases–if the attacker is using random walk. These results reinforce that hebbian–while efficient in terms of learning speed–leads to increased vulnerability when the attack model is predictable. Alternatively, when random walk is used by the attacker, the dynamic methods show clear statistical advantages.

In case the attacker uses the random update rule, the dynamic_rows and dynamic_matrix in most cases do not provide more security compared to all parties using hebbian, but they outperform random walk-based synchronization in all of the cases. This trade-off indicates that dynamic update rules may not always be optimal in absolute terms, but can introduce substantial confusion and desynchronization in adversarial conditions where the attacker's

strategy is uncertain or varies between the attacks. This robustness to unpredictability makes the dynamic rules particularly appealing in practical QKD systems where full knowledge of the attack model cannot be assumed.

Based on all of this information, we can conclude two main outcomes (for synchronization using machines with parameters as described in Section 3.3).

- In the vast majority of cases, it is more secure to use either `dynamic_rows` or `dynamic_matrix` if we are sure that the attacker will use a static update rule (in case they are using `hebbian` rule, it is always more profitable).
- If the communicating parties would want to use `random walk` as the update rule, in the vast majority of cases it is more secure to use `dynamic` approaches, no matter what the attacker's approach is (in case they are using random update rule, it is always more profitable).

It is quite important that these gains are achieved without adding any modifications to the TPM architecture or increasing the amount of exchanged data (not counting the results from the less complicated machines, like those presented in Sections 4.1.1 and 4.2.1), preserving compatibility with existing QKD protocols. This compatibility, combined with enhanced security under realistic adversarial conditions, suggests that `dynamic` update rule selection is a viable enhancement to existing QKD error correction protocols.

Future studies on the `dynamic` approaches to TPM synchronization in the context of cryptography applications should include a broader data set (e.g., more populated or expanded sets of parameters' values) or should be based on some specific list of TPM parameters–either already used in the QKD process, or some recommended values (like those proposed in [46]). The additional expansion of the research could explore integration into live QKD networks and further optimization of selection criteria to balance security and synchronization speed (e.g., integrating multiple approaches to leverage their respective strenghts while simultaneously mitigating the limitations and risks associated with using individual soltions in isolation), comparison of `dynamic_rows` and `dynamic_matrix` with other, less popular update rules (such as selective update rule based on entropy, error-corrective update rule which updates weights only when the outputs differ, weight-decaying update rule, or standard rule with additional decaying factor) or even testing the proposed solutions with novel approaches, such as synchronization with non-binary input vectors [47]. Additionally, we do recognize that the attacker may somehow obtain the information on how the victims choose their update rule on each iteration, so it would be worth comparing the performance of synchronization (in the means of number of updates required to synchronize the key, number of attacker updates or attacker's synchronization percentage) in the scenario where all parties use 'dynamic' approaches, compared to standard update rules scenarios.

Author Contributions: Conceptualization: B.G. and M.N.; methodology: B.G. and M.N.; software: B.G.; validation: B.G.; formal analysis: B.G.; investigation: B.G. and M.N.; resources: B.G.; data curation: B.G.; writing—original draft preparation: B.G. and M.N.; writing—review and editing: B.G., M.M., and M.N.; visualization: B.G.; supervision: M.M. and M.N.; project administration: M.N.; funding acquisition: M.N. All authors have read and agreed to the published version of the manuscript.

Funding: This research project was partly supported by the program "Excellence initiative—research university" for the AGH University of Krakow. This research was also supported by the National Research Institute, grant number POIR.04.02.00-00-D008/20-01 on "National Laboratory for Advanced 5G Research" (acronym PL-5G) as part of the Measure 4.2 Development of modern research infrastructure of the science sector 2014-2020 financed by the European Regional Development Fund and by the EU Horizon Europe Framework Program under grant agreement no. 101119547 (PQ-

REACT). This work was also supported by the Federal Ministry of Education and Science of Bosnia and Herzegovina within the project "Virtual Quantum Cryptographic Networks".

Institutional Review Board Statement: Not applicable.

Data Availability Statement: The results, as well as the source codes for the software used during the research, are available in a GitHub repository, under following link: https://github.com/bgdowski-agh/dynamic_TPM_update (accessed on 4 July 2025).

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Barker, E.; Dang, Q. *Recommendation for Key Management, Part 3: Application-Specific Key Management Guidance*; Technical Report; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2015.
2. Wood, G. Encryption Security for a Post Quantum World. Available online: https://csis.org/blogs/strategic-technologies-blog/encryption-security-post-quantum-world (accessed on 4 July 2025).
3. AtomC. Quantum Startup Atom Computing First to Exceed 1000 Qubits). Available online: https://atom-computing.com/quantum-startup-atom-computing-first-to-exceed-1000-qubits/ (accessed on 4 July 2025).
4. IBM. Expanding the IBM Quantum Roadmap to Anticipate the Future of Quantum-Centric Supercomputing (Updated 2024). Available online: https://research.ibm.com/blog/ibm-quantum-roadmap-2025 (accessed on 4 July 2025).
5. Opiłka, F.; Niemiec, M.; Gagliardi, M.; Kourtis, M.A. Performance Analysis of Post-Quantum Cryptography Algorithms for Digital Signature. *Appl. Sci.* **2024**, *14*, 4994. [CrossRef]
6. Hughes, R.J.; Alde, D.M.; Dyer, P.; Luther, G.G.; Morgan, G.L.; Schauer, M. Quantum cryptography. *Contemp. Phys.* **1995**, *36*, 149–163. [CrossRef]
7. Mehic, M.; Niemiec, M.; Rass, S.; Ma, J.; Peev, M.; Aguado, A.; Martin, V.; Schauer, S.; Poppe, A.; Pacher, C.; et al. Quantum key distribution: A networking perspective. *ACM Comput. Surv.* **2020**, *53*, 96. [CrossRef]
8. Ekert, A.; Gisin, N.; Huttner, B.; Inamori, H.; Weinfurter, H. Quantum Cryptography. In *The Physics of Quantum Information*; Springer: Berlin/Heidelberg, Germany, 2000.
9. Bennett, C.; Brasard, G. Quantum cryptography: Public key distribution and coin tossing. In Proceedings of the IEEE International Conference on Computers, Systems, and Signal Processing, Bangalore, India, 10–19 December 1984.
10. Bennett, C. Quantum cryptography using any two nonorthogonal states. *Phys. Rev. Lett.* **1992**, *68*, 3121–3124. [CrossRef] [PubMed]
11. Ekert, A. Quantum cryptography based on Bell's theorem. *Phys. Rev. Lett.* **1991**, *67*, 661–663. [CrossRef] [PubMed]
12. Dušek, M.; Lütkenhaus, N.; Hendrych, M. Quantum cryptography. *Prog. Opt.* **2006**, *49*, 381–454.
13. Heisenberg, W. Über den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik. *Z. Phys.* **1927**, *43*, 172–198. [CrossRef]
14. Gisin, N.; Ribordy, G.; Tittel, W.; Zbinden, H. Quantum cryptography. *Rev. Mod. Phys.* **2002**, *74*, 145–195. [CrossRef]
15. Wootters, W.K.; Zurek, W.H. A single quantum cannot be cloned. *Nature* **1982**, *299*, 802–803. [CrossRef]
16. Devitt, S.; Munro, W.; Nemoto, K. Quantum error correction for beginners. *Rep. Prog. Phys.* **2013**, *7*, 076001. [CrossRef] [PubMed]
17. Liao, S.-K.; Cai, W.-Q.; Liu, W.-Y.; Zhang, L.; Li, Y.; Ren, J.-G.; Yin, J.; Shen, Q.; Cao, Y.; Li, Z.-P.; et al. Satellite-to-ground quantum key distribution. *Nature* **2017**, *549*, 43–47. [CrossRef] [PubMed]
18. Dixon, A.R.; Dynes, J.F.; Lucamarini, M.; Fröhlich, B.; Sharpe, A.W.; Plews, A.; Tam, S.; Yuan, Z.L.; Tanizawa, Y.; Sato, H.; et al. 77 day field trial of high speed quantum key distribution with implementation security. In Proceedings of the 6th International Conference on Quantum Cryptography (QCrypt), Washington, DC, USA, 12–16 September 2016.
19. Niemiec, M.; Pach, A. The measure of security in quantum cryptography. In Proceedings of the 2012 IEEE Global Communications Conference (GLOBECOM), Anaheim, CA, USA, 3–7 December 2012.
20. Bennett, C.H.; Bessette, F.; Brassard, G.; Salvail, L.; Smolin, J. Experimental quantum cryptography. *J. Cryptol.* **1992**, *5*, 3–28. [CrossRef]
21. Jennewein, T.; Simon, C.; Weihs, G.; Weinfurter, H.; Zeilinger, A. Quantum Cryptography with Entangled Photons. *Phys. Rev. Lett.* **2000**, *84*, 4729–4732. [CrossRef] [PubMed]
22. Brassard, G.; Salvail, L. Secret-Key Reconciliation by Public Discussion. In *Advances in Cryptology—EUROCRYPT '93*; Springer: Berlin/Heidelberg, Germany, 1994.
23. Niemiec, M. Error correction in quantum cryptography based on artificial neural networks. *Quantum Information Processing* **2019**, *18*, 174. [CrossRef]

24. Bennett, C.H.; Brassard, G.; Crepeau, C.; Maurer, U.M. Generalized privacy amplification. *IEEE Trans. Inf. Theory* **1995**, *41*, 1915–1923. [CrossRef]
25. Mehic, M.; Rass, S.; Fazio, P.; Voznak, M. *Quantum Key Distribution Networks: A Quality of Service Perspective*; Springer: Cham, Switzerland, 2022.
26. Sobti, R.; Geetha, G. Cryptographic hash functions: A review. *Int. J. Comput. Sci. Issues* **2012**, *9*, 461–479.
27. Kasiviswanathan, S.P.; Lee, H.K.; Nissim, K.; Raskhodnikova, S.; Smith, A. What Can We Learn Privately? In Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science, Philadelphia, PA, USA, 25–28 October 2008.
28. Feldman, V.; Mironov, I.; Talwar, K.; Thakurta, A. Privacy Amplification by Iteration. In Proceedings of the 2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS), Paris, France, 7–9 October 2018.
29. McCulloch, W.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* **1943**, *5*, 115–133. [CrossRef]
30. Hansel, D.; Sompolinsky, H. Learning from examples in a single-layer neural network. *EPL (Europhys. Lett.)* **1990**, *11*, 687. [CrossRef]
31. Jain, A.; Mao, J.; Mohiuddin, K. Artificial neural networks: A tutorial. *Computer* **1996**, *29*, 31–44. [CrossRef]
32. Barkai, E.; Hansel, D.; Kanter, I. Statistical mechanics of a multilayered neural network. *Phys. Rev. Lett.* **1990**, *65*, 2312–2315. [CrossRef] [PubMed]
33. Ruttor, A.; Kinzel, W.; Naeh, R.; Kanter, I. Genetic attack on neural cryptography. *Phys. Rev. E* **2006**, *73*, 036121. [CrossRef] [PubMed]
34. Kanter, I.; Kinzel, W.; Kanter, E. Secure exchange of information by synchronization of neural networks. *Europhys. Lett. (EPL)* **2002**, *57*, 141–147. [CrossRef]
35. Dourlens, S. Neuro-Cryptographie Appliquée et Neuro-Cryptanalyse du DES. Ph.D. Thesis, Paris 8 University, Paris, France, 1995.
36. Kinzel, W.; Kanter, I. Neural cryptography. In Proceedings of the 9th International Conference on Neural Information Processing (ICONIP), Singapore, 18–22 November 2002.
37. Javurek, M.; Turčanik, M. Synchronization of two tree parity machines. In Proceedings of the 2016 New Trends in Signal Processing (NTSP), Demanovska dolina, Slovakia, 12–14 October 2016.
38. Klein, E.; Mislovaty, R.; Kanter, I.; Ruttor, A.; Kinzel, W. Synchronization of neural networks by mutual learning and its application to cryptography. In Proceedings of the NIPS, Vancouver, BC, Canada, 13–18 December 2004.
39. Aleksandrov, M.; Bashkov, Y. Factors Affecting Synchronization Time of Tree Parity Machines in Cryptography. In Proceedings of the 2020 IEEE 2nd International Conference on Advanced Trends in Information Theory (ATIT), Kyiv, Ukraine, 25–27 November 2020.
40. Dolecki, M.; Kozera, R. Distribution of the Tree Parity Machine Synchronization Time. *Adv. Sci. Technol. Res. J.* **2013**, *7*, 20–27. [CrossRef]
41. farizrahman4u. GitHub–NeuralKey. Available online: https://github.com/farizrahman4u/neuralkey (accessed on 6 June 2025).
42. bgdowski_agh. GitHub–Dynamic TPM Update. Available online: https://github.com/bgdowski-agh/dynamic_TPM_update (accessed on 4 July 2025).
43. Ruttor, A. Neural Synchronization and Cryptography. Ph.D. Thesis, Julius Maximilian University of Würzburg, Würzburg, Germany, 2009.
44. Dolecki, M.; Kozera, R. The Impact of the TPM Weights Distribution on Network Synchronization Time. In *Computer Information Systems and Industrial Management. CISIM 2015*; Springer: Cham, Switzerland, 2015.
45. Teodoro, A.; Gomes, O.S.M.; Saadi, M.; Silva, B.A.; Rosa, R.L.; Rodríguez, D.Z. An FPGA-Based Performance Evaluation of Artificial Neural Network Architecture Algorithm for IoT. *Wirel. Pers. Commun.* **2022**, *127*, 1085–1116. [CrossRef]
46. Niemiec, M.; Widlarz, T.; Mehic, M. Secure Synchronization of Artificial Neural Networks Used to Correct Errors in Quantum Cryptography. In Proceedings of the ICC 2023—IEEE International Conference on Communications, Rome, Italy, 28 May–1 June 2023.
47. Stypiński, M.; Niemiec, M. Synchronization of Tree Parity Machines Using Nonbinary Input Vectors. *IEEE Trans. Neural Netw. Learn. Syst.* **2024**, *35*, 1423–1429. [CrossRef] [PubMed]