Error correction in quantum cryptography based on artificial neural networks

Marcin Niemiec[1] (ORCID)

Abstract

Intensive work on quantum computing has increased interest in quantum cryptography in recent years. Although this technique is characterized by a very high level of security, there are still challenges that limit the widespread use of quantum key distribution. One of the most important problems remains secure and effective mechanisms for the key distillation process. This article presents a new idea for a key reconciliation method in quantum cryptography. This proposal assumes the use of mutual synchronization of artificial neural networks to correct errors occurring during transmission in the quantum channel. Users can build neural networks based on their own string of bits. The typical value of the quantum bit error rate does not exceed a few percent; therefore, the strings are similar and also users' neural networks are very similar at the beginning of the learning process. It has been shown that the synchronization process in the new solution is much faster than in the analogous scenario used in neural cryptography. This feature significantly increases the level of security because a potential eavesdropper cannot effectively synchronize their own artificial neural networks in order to obtain information about the key. Therefore, the key reconciliation based on the new idea can be a secure and efficient solution.

Keywords Quantum cryptography · Artificial neural networks · Error correction · Machine learning

1 Introduction

Quantum cryptography is a technique which can ensure a very high level of data security. Thanks to principles of quantum mechanics, secret keys can be established between entities/users—usually called Alice and Bob. At the same time, an eavesdropper (called Eve) can attempt to gain information about the key. However, measurement

✉ Marcin Niemiec
niemiec@kt.agh.edu.pl

1 AGH University of Science and Technology, Mickiewicza 30, 30-059 Kraków, Poland

modifies the state of the transmitted information and even passive eavesdropping can be discovered by Alice and Bob.

After quantum key distribution in the quantum channel, the users must perform a key distillation process (consisting of quantum bit error estimation, error correction and privacy amplification) in order to establish the secure final key. This process directly influences the performance of key distribution and also the security and length of final cryptographic key. Therefore, it is desirable to use secure and efficient methods in practical implementations. These features are inherent in the solution presented in this article—synchronization of the artificial neural network to correct errors occurring in quantum channel during quantum key distribution process.

The rest of the article proceeds as follows. An introduction to quantum cryptography—especially a description of the key distillation process—is presented in Sect. 2. An introduction and presentation of the artificial neural networks used in neural cryptography follow in Sect. 3. The new solution based on machine learning in error correction process is presented in Sect. 4. In Sect. 5, experimental results are discussed and compared with typical scenarios used in neural cryptography. Finally, Sect. 6 concludes the article.

2 Quantum cryptography

Quantum cryptography ensures secure key distribution by means of the laws of quantum mechanics [1]. First of all, the rules of quantum mechanics ensure that measurement modifies the state of the transmitted qubit (quantum bit). This modification can be discovered by the sender and the receiver of qubits. Therefore, quantum cryptography requires two types of channels to be defined:

- *the quantum channel,* where qubits with the information about the distributed key are exchanged and
- *the public channel,* which is used to check whether the communication through the quantum channel is distorted. Also, this channel is used for the correction of wrong bits.

The other rule of quantum mechanics which makes quantum cryptography a very secure solution is the no-cloning theorem [2]. According to this theorem, it is not possible to create identical copies of an unknown quantum state [3]. Therefore, an eavesdropper is not able to clone the original qubit to measure the quantum state and send the second qubit to the proper receiver.

We can split quantum cryptography into two main steps: the quantum key distribution protocol (e.g., BB84 protocol) and the key distillation algorithms (quantum error bit estimation, error correction and privacy amplification).

2.1 Quantum key distribution

Quantum key distribution (QKD) is used to distribute an encryption key for symmetric ciphers [4] (not to transmit messages between users). As has been mentioned, the security of QKD relies on the foundations of quantum mechanics and information

about a key is transmitted by means of qubits. We could distinguish two types of QKD protocols: based on single and entangled particles [5].

In the first group—QKD protocols based on single particles—information about the distributed key is coded by means of quantum states of single particles (such as polarized photons). The quantum states of the particles do not depend on each other, and each particle brings information which can be read independently.

The second group is based on entanglement. The entangled state of two particles has the following feature: The states of particles are random (indeterminate) before the measurement is performed but if we measure the state of the first particle, then the state of the second particle is fully determined. This means that we only need to measure one particle to know the states of both. It is worth mentioning that the entanglement still retains this feature even if the particles are separated.

Today, we know a lot of QKD protocols but only a few are used in practice [6]. The first protocol invented was BB84 [7], presented in 1984 by Bennett and Brassard. This protocol is based on single particles (polarized photons). Another protocol based on single particles is B92—developed by one of the creators of BB84, Bennett, in 1992 [8]. It is simpler and faster than its predecessor. Furthermore, it is more efficient because it detects eavesdroppers faster. A well-known QKD protocol based on entanglement is E91, invented in 1991 by Ekert [9]. It was an innovative solution which used the phenomenon of entangled particles for the first time. In principle, many other protocols, such as BBM92 [10] (proposed by Bennett, Brassard and Mermin in 1992) or SARG04 [11] (proposed by Scarano, AcĂn, Ribordy and Gisin in 2004) are modified versions of the BB84 protocol.

2.2 Key distillation

During the quantum key distribution process, Alice and Bob use two communication channels: quantum and public. In the quantum channel, information is coded by means of quantum states. In the public channel, Alice and Bob exchange data to check whether Eve is eavesdropping. However, the public channel is necessary for more cases.

It is not only Eve that is responsible for errors in the quantum channel. Errors during quantum communication may occur because of disturbance in the quantum channel, optical misalignment, noise in detectors or other factors. Therefore, Alice and Bob have to estimate the error rate and decide whether there is an eavesdropper in between or not. In practice, they compare a small portion of a distributed raw key through the public channel and compute the quantum bit error rate (QBER). The portion of compared bits can depend on the security requirements [12]. If QBER exceeds a given threshold, it means that Eve has eavesdropped (or the quantum channel is too noisy to perform a proper key distribution). But if the error rate is low enough, Alice and Bob continue further distillation of the key. Of course, they must delete the compared part of the raw key for security reasons.

After the bit error estimation, Alice and Bob use key distillation protocols. These protocols usually involve two steps: key reconciliation (error correction) and privacy amplification.

As mentioned previously, quantum communication is not perfect and some errors usually occur. If the number of errors does not exceed a given threshold of QBER, the reconciliation process must find and correct or delete these errors. Alice and Bob should disclose as little information as possible by using an appropriate reconciliation algorithm. Since they are not able to avoid the leakage of information, they have to reject some bits of the key.

The first binary error correction method was provided by the BBBSS protocol. This protocol was designed by Bennett and his coworkers [13]. It requires the parities of raw key subsets from Alice and Bob to be exchanged. BBBSS uses several passes to correct the errors by parity check. A pseudo-random permutation is used after each pass. Two years later, Brassard and Salvail constructed the Cascade algorithm with improved efficiency [14]. Usually, it uses four passes and doubles block length starting from the second pass. This ensures a faster error correction process. Nowadays, the Cascade key reconciliation algorithm is usually used in practical implementations. Other reconciliation methods based on the BBBSS algorithm are Furukawa–Yamazaki [15] (less efficient than the Cascade) and Winnow protocol [16] which uses a Hamming code to reduce the number of errors.

Alice and Bob can choose one of several known reconciliation algorithms; however, currently the most popular reconciliation methods are algorithms which are based on a parity check of blocks. The simplest scenario assumes that the key is grouped into blocks of a given size. The size of a block depends on the error rate value which was estimated before. Alice and Bob compare parities of each block over the public channel. If their parities disagree, the block contains an odd number of errors. This block is cut into two sub-blocks, and their parities are compared again. This procedure is continued recursively for all blocks which contain an odd number of errors as long as errors will be corrected. After that, both keys contain an even number of errors or none. Alice and Bob shuffle the positions of bits and repeat the same procedure with blocks of bigger size as long as both keys will be the same. A serious problem occurs if blocks contain an even numbers of errors. Therefore, users must try to change the block size or rearrange the position of errors in the string. However, this approach can be ineffective and may even lead to failing the error correction process.

Each parity control over the public channel discloses a part of the secret key's information. If Eve collects the parities of many blocks, she will be able to calculate parts of the key. Therefore, Alice and Bob must reject some bits to reduce the eavesdropper's knowledge about the secret key. Many rejected bits increase the security level but unfortunately decrease the final length of the key. It decreases the efficiency of whole QKD system. The ideal key reconciliation algorithm should ensure an efficient and secure error correction process as well as avoid leakage of information about the key.

At the end of the key distillation process, the privacy amplification should be carried out. Because Eve may have gained significant knowledge of the key (eavesdropping in the quantum channel and in the public channel during the bit error estimation and key reconciliation), Alice and Bob are required to strengthen their privacy. They can delete some of the bits and construct the final key in a specific way.

Even though it is possible to apply different solutions during the privacy amplification process, universal hash functions are mainly used in practice. Universal families

of hash functions were created by Wegman and Carter [17]. Privacy amplification with hash functions was proposed by Bennett et al. [18] in 1988. In general, the algorithm is based on one-way functions which are able to convert a large string of bits into a short binary word.

Following [18], the theorem which defines the probability of the eavesdropper's information after error correction is presented below.

Theorem 1 *Assume that M is the length of the reconciled key and Eve's knowledge about the key is no more than E deterministic bits. Let $h : \{0, 1\}^M \rightarrow \{0, 1\}^E$ be any hash function of the universal family, let $S < M - E$ be a security parameter and let $R = M - E - S$. If $g : \{0, 1\}^M \rightarrow \{0, 1\}^R$ is chosen randomly, then the expected amount of information on $g(x)$ given by knowledge of h, g and $h(x)$ is at most: $2^{-S}/\log 2$. This means that:*

$$\text{Expected amount of information} \ \leq \ \frac{2^{-S}}{\log 2} \ \text{[bit].} \tag{1}$$

The security parameter S allows the security of the final encryption key to be controlled. By means of the theorem, we are able to increase the security of a given QC system. Unfortunately, too many rejected bits decrease the final length of the key, and thus the efficiency of QKD system also decreases.

If Alice and Bob perform all these steps, the final key will be significantly reduced. This is characteristic for all quantum key distribution protocols [19]. Because each stage reduces the key length, the performance of QKD is also reduced. Sometimes, when we want to ensure a high level of security, this reduction is significant. Using the QKD Protocol Simulator [20], we can easily check that, e.g., 1000 qubits transmitted in the quantum channel cause approx. 300 bits of final key. Therefore, improving efficiency of key distillation process is crucial to the quantum cryptography implemented in real communication networks.

3 Artificial neural networks

Artificial neural networks (ANN) are a family of statistical learning models inspired by biological neural networks [21]. They are used to estimate functions that can depend on a large number of inputs. An ANN consists of artificial neurons (analogous to biological neurons) which are connected together. Each connection can transmit a signal between neurons [22]. Neurons are usually organized in layers: The first layer consists of input neurons which can send the data to the second layer (called hidden). A neural network can have one or more hidden layers. The last layer—consisting of output neurons—is called the output layer. The connections can store parameters (called weights) that can be manipulated during calculation.
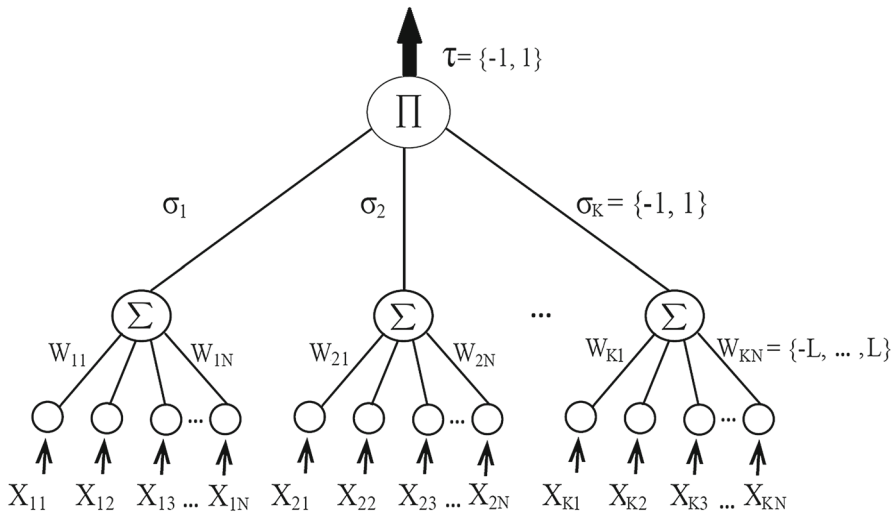
Fig. 1 TPM machine

3.1 Tree parity machine

The most popular neural network used for cryptography purposes is the tree parity machine (TPM) which contains only one hidden layer. An example TPM structure is presented in Fig. 1. It consists of KN input neurons, where K is the number of neurons in the hidden layer and N is the number of inputs into each neuron in the hidden layer. This network has only one output neuron. Each connection between the input layer and hidden layer is characterized by its weight, which is an integer from the range $[-L, L]$.

The output value of neuron k in the hidden layer depends on input x and weight w and is calculated as:

$$\sigma_k = \text{sgn}\left(\sum_{n=1}^{N} x_{kn} * w_{kn}\right) \tag{2}$$

where signum function is:

$$\text{sgn}(z) = \begin{cases} -1 & z \leq 0 \\ 1 & z > 0 \end{cases} \tag{3}$$

The output value of the neuron in the output layer is calculated as:

$$\tau = \prod_{k=1}^{K} \sigma_k \tag{4}$$

Nowadays, the TPMs are used for establishing the secret key between users. This usage of ANN for cryptographic purposes is called neural cryptography. Alice and Bob use two identical neural networks which are able to synchronize after mutual

learning [23]. At the beginning of this process, each TPM generates random values of weights but after synchronization process both users have TPMs with the same values of weights. Therefore, Alice and Bob can construct the secret key using synchronized weights (just change weight values into binary string).

In order to synchronize neural networks, users generate random input (the same for both TPMs) and compute outputs from each TPM. If the output of Alice's TPM is the same as Bob's TPM, they can start the learning process for the neural networks. If the outputs are different (one TPM generated the value 1 but the other generated the value -1), Alice and Bob must generate another input.

We can choose any learning algorithm; however, the generalized form of Hebbian method is the most popular in practical implementations [24]. The new weights are calculated by means of the following formula:

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

where:

$$\Theta(\sigma_k, \tau)) = \begin{cases} 0 & \text{if } \sigma_k \neq \tau \\ 1 & \text{if } \sigma_k = \tau \end{cases} \tag{6}$$

and function ν_L limits values of connections to the range $[-L, L]$:

$$\nu_L(z) = \begin{cases} -L & \text{if } z \leq -L \\ z & \text{if } -L < z < L \\ L & \text{if } z \geq L \end{cases} \tag{7}$$

As we can see, the algorithm strengthens the connections which have the same value as the TPM output.

After the appropriate number of iterations, the synchronization process ends and the weights of both TPM machines are the same. Then, Alice and Bob can change weights into binary strings and use them as a secret cryptographic key.

3.2 Security of neural cryptography

Synchronization of TPMs requires communication between Alice and Bob. Therefore, it can be eavesdropped by an intruder (Eve). The simplest passive attack is an attempt to synchronize the Eve's TPM machine with the TPMs belonging to Alice and Bob. We can specify that during the synchronization process, three events may occur:

* if $\tau^{\text{Alice}} \neq \tau^{\text{Bob}}$, then no TPM machine is subjected to the learning process,
* if $\tau^{\text{Alice}} = \tau^{\text{Bob}} \neq \tau^{\text{Eve}}$, then only the machines of Alice and Bob are subjected to the learning process,
* if $\tau^{\text{Alice}} = \tau^{\text{Bob}} = \tau^{\text{Eve}}$, then all machines are subjected to the learning process.

If the output of the Eve's TPM machine is different than the outputs of the Alice and Bob's machines, the learning process cannot be performed. Therefore, the synchronization of the Eve's TPM is slower than the synchronization of the TPMs belonging to
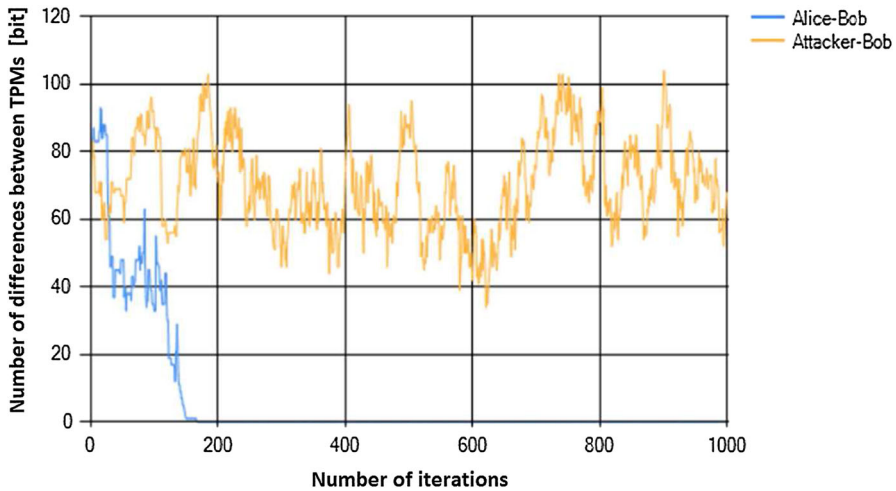
Fig. 2 Synchronization of TPMs (Alice, Bob and intruder)

Alice and Bob. An example of the synchronization process is presented in Fig. 2 (TPM machines with parameters: $N = 8$, $K = 6$, $L = 2$ and Hebbian learning algorithm). Alice and Bob synchronized neural networks before 200 iterations, but attacker was not able to do it for 1000 iterations.

Unfortunately, an attack can be improved by using multiple TPMs owned by Eve. In this case, the attacker has many machines, each initiated with different weights. This method increases the probability of the attacker's success, since it is enough that only one machine will be synchronized with Alice's and Bob's machines. However, simultaneous synchronization of many TPM machines is ineffective and users can easily improve the security by increasing the used neural network (increasing L, N and K parameters). This results in a reduction in the synchronization speed of the attacker.

A known improvement to the introduced simple passive attack is worth mentioning. The event $\tau^{\text{Alice}} = \tau^{\text{Bob}} \neq \tau^{\text{Eve}}$ cannot stop Eve's synchronization process. This approach is known as a geometric attack [25]. The attacker can flip the output of a selected neuron in the hidden layer before applying the learning process in order to correct the output τ^{Eve}. Taking into account the correlation of weights in TPMs machines belonging to Alice, Bob and Eve, the authors of [26] considered attractive and repulsive steps of the synchronization process. Despite the fact that a geometric attack can improve the learning process of Eve's TPM machine, the synchronization process is still less effective. Finally, it was demonstrated that ANNs which interact with each other (Alice's and Bob's TPMs) synchronize faster than Eve's TPM machine.

The security of neural cryptography has been considered in a number of papers [21, 27–32]. However, synchronization of TPMs can be further improved by learning by queries [33] instead of random inputs. This approach is based on exchanging inputs between Alice and Bob which are correlated with the weight vectors of the TPMs. The queries introduce a mutual influence between Alice and Bob which is not available to

an attacking Eve. The results shown in [34] confirm that queries restore the security against cooperating attackers.

4 Error correction based on TPMs

One of the crucial steps in the quantum key distribution process is the correction of errors. This step decides on the security level of final key, but also significantly influences the performance of the quantum cryptography. Currently used solutions assume parity checking and deleting bits to minimize the probability of information leakage. Such an approach causes a significant reduction in key length and low efficiency in the whole QKD system. In this section, a new approach to error correction based on mutual synchronization of TPM machines is introduced.

4.1 Applicability of TPMs

The idea for a new error correction method is the following: After the QBER estimation step, we can use the synchronization of the TPM machines to correct errors in the quantum cryptography (instead of any other error correction algorithm). In this way, we will be able to correct errors that occurred during the transmission of qubits. Importantly, in this scenario, Alice's binary string is very similar to Bob's string of bits. The typical value for QBER does not exceed a few percent [35–38]; therefore, we must correct only a small part of the whole key. This means that the TPM machines are close to synchronization and the learning process will finish much faster than in the case of synchronization of random strings of bits. Of course, this increases the security level significantly.

It is worth mentioning that the presented idea—using the mutual synchronization of neural networks to correct errors—is a special case when this process makes sense. In general, TPM machines cannot be used for error correction of digital information because we are not able to predict the final weights after the learning process. Both TPM machines dynamically adjust their weights; therefore, the final strings are random. However, in the case of quantum cryptography, this feature is an advantage because we want to generate unpredictable string of bits which can be used as a secure key.

Taking into account the software environments and hardware resources currently available, the usage of TPM-based error correction seems to be justified. Although the time of learning processes for software implementations of TPMs strongly depends on the hardware resources, a typical duration of one iteration on an ordinary computer is a few milliseconds [39]. However, hardware implementation helps to shorten this time by more than thousandfold (i.e., parallel processing using FPGA presented in [40]). Additionally, ASIC implementations of neural cryptography in 130-nm and 65-nm standard-cell CMOS technologies are available [41]. These circuits reduce implementation costs and ensure fast synchronization of neural networks; the maximum operation frequency is several hundred MHz which results less than one microsecond per single iteration.

Additionally, the security services and architectures being used confirm the feasibility of TPM implementation in practice. For example, a chip-level microcomputer bus system with TPMs introduced in [42] provides efficient data encryption with a low hardware overhead, comparable to well-known and widely used stream ciphers. Moreover, synchronization of ANNs was proposed to environments such as ad hoc networks (TPMs for establishing common group keys [43]) or wireless sensor networks with limited resources (lightweight key agreement protocol based on TPMs known as TinyTPM [44]).

4.2 Error correction process

The use case with the proposed solution is as follows. Let's assume that Alice and Bob carried out the process of quantum key distribution in the quantum channel and they estimated the quantum bit error rate. If the QBER level is acceptable (this means that no one eavesdropped on the quantum channel or a very small percentage of bits were eavesdropped), the error correction process can start.

Step 1 Alice and Bob create their own TPM machines based on their own strings of bits. The users change string of bits into weights in their own TPM machines (bits into numbers from the range $[-L, L]$). Values $\{-L, -L+1, \ldots, L-1, L\}$ become weights of connections between the input neurons and the neurons in the hidden layer. Values of parameter K (the number of neurons in the hidden layer) and N (the number of inputs into each neuron in the hidden layer) are chosen by Alice and Bob and can be public. In this way, Alice and Bob construct very similar neural networks—the TPM machines have the same structure (KN input neurons), and most of the weights are the same. The differences are located only in the places where errors occurred: For example, if QBER $\approx 3\%$, it means that $\approx 97\%$ of bits are correct.

Step 2 After the construction of the neural networks, synchronization of the TPM machines begins and continues until all weights in both machines become the same. In order to synchronize neural networks, Alice generates random input (an input string has KN length) and computes output from her own TPM machine. Then, Alice informs Bob about the result (value 1 or -1) and also the generates input string. Bob computes output from his own TPM machine, based on the input string generated by Alice. If the output of Alice's TPM is the same as Bob's TPM, they can start the learning process for the neural networks (otherwise, the different input string is generated by Alice). The synchronization process can be based on the Hebbian learning algorithm, which strengthens the connections which have the same value as the TPM output. After the appropriate number of iterations, the synchronization process ends and the weights of both TPM machines are the same.

Step 3 When the TPM machines are synchronized, the weights are the same in both neural networks. Therefore, Alice and Bob can convert the weights back into string of bits—the users change numbers from the range $[-L, L]$ into bits (in the opposite way than in Step 1). Because both the TPM machines have been synchronized, the Alice's string of bit is now the same as the Bob's string of bits. All errors have been corrected.

After this three-step error correction process, both users may use the obtained string of bits for cryptography purposes, e.g., to secure communications ensuring confidentiality, integrity or authentication.

4.3 Security considerations

The presented solution—using the mutual synchronization of TPM machines to correct errors—is not based on parity check which causes the information leakage; however, privacy amplification process is still recommended. It will protect this solution against unknown attacks to TPMs, which can be proposed in the future.

Let's assume that before the error correction process, Alice and Bob changed string of bits into weights and created their own TPM machines. Thus, the TPM machines contain KN input neurons with weights and each weight is an integer from the range $[-L, L]$. Therefore, a single weight has $2L+1$ possible values. The number of possible keys which are stored using TPM is:

$$(2L + 1)^{KN} \qquad (8)$$

However, after each synchronization process (iteration), Eve is able to acquire a partial information about TPMs. Taking into account the input and output of TPMs (value τ), Eve can reject the half of possible keys (Eq. 8) from the further considerations. Therefore, after the i iterations, the number of possibilities is reduced to:

$$2^{-i}(2L + 1)^{KN} \qquad (9)$$

and this is adequate to a TPM machine with smaller number of input neurons with weights:

$$(2L + 1)^{KN-Z} \qquad (10)$$

Comparing both Eqs. (9 and 10), we are able to quantify the maximum Eve's knowledge after i iterations and define the reduction of key to protect Alice and Bob against the information leakage during TPMs synchronization process as:

$$Z = \log_{(2L+1)} 2^i \qquad (11)$$

This reduction strongly depends on the parameter L. However, the reduction of key caused by the synchronization process for the typical QBER value is not very high. Even for small parameter L—i.e., TPMs with $L = 2$ used for verification in the next section)—the reduction is a dozen or so percent. When Alice and Bob convert the weights back into bits, they may shorten the final key using a hash function and a proper value of the security parameter S (regarding Eq. 1). In this way, Alice and Bob reduce Eve's knowledge of the key, which can be collected by eavesdropping in the quantum channel and public channel during the bit error estimation and key reconciliation steps.

Additionally, the proposed solution is characterized by higher security than current neural cryptography solutions, where we use TPM machines to establish cryptography

key between users. In the new solution, we have much faster synchronization because the strings are very similar at the beginning of the synchronization process. Therefore, Alice and Bob need fewer iterations to synchronize their neural networks. However, error correction based on TPMs is able to equalize every number of incompatible bits between Alice and Bob's strings of bits and it works on any value of QBER. Also, it is worth remembering that we should use high values for TPM parameters (N, K, L) to ensure an even higher level of protection [39].

5 Verification

Security and efficiency of every key reconciliation method are the crucial requirements. It also applies to the proposed error correction based on mutual synchronization of artificial neural networks. The number of iterations during the synchronization of TPMs influences the security and efficiency. If the synchronization of Alice and Bob's TPMs is fast, the level of security will be high. Therefore, the scenarios of TPMs synchronization process with typical values of QBER are tested in this section.

5.1 The security of error correction based on TPMs

The number of steps during the synchronization of TPMs directly influences the security level. However, the synchronization of an eavesdropper's TPM machine is slower than the synchronization of the users' TPMs, but the initial synchronization additionally increases the level of security. To verify the behavior of the initially synchronized TPMs, a number of simulations have been conducted. The results were compared to typical TPMs used in neural cryptography (artificial neural networks with randomly chosen weights).

Figures 3 and 4 present the synchronizations of TPMs in two scenarios—with weights randomly generated and with 95% of synchronized weights at the beginning of the synchronization process, respectively. The points in the graph are mean values (synchronizations repeated 5000–10,000 times) and were connected by dotted lines in order to help in comparison of differences. The results were presented for the range $N = [20, 25]$, parameter $L = 2$ and Hebbian learning algorithm. The number of iterations in synchronization process strongly depends on the value of the K parameter (the figures contain three example values: $K = 6$, $K = 8$ and $K = 10$).

According to predictions, the number of iterations which are needed to synchronize TPMs is much smaller in scenario with ANN initially synchronized (3–4 times smaller). Artificial neural networks with random chosen weights need significantly more iterations to synchronize their weights.

Additionally, numerous simulations were performed with synchronization of bigger ANNs in scenarios with QBER = 3% and QBER = 1%. Figure 5 presents the comparison of speed of TPMs synchronization depending on parameter K in two scenarios: with random weights and with 97% synchronized weights at the beginning of the TPMs synchronization process (both for $N = 30$). Figure 6 presents results for bigger TPMs ($N = 50$) and allows differences between the synchronization of TPMs
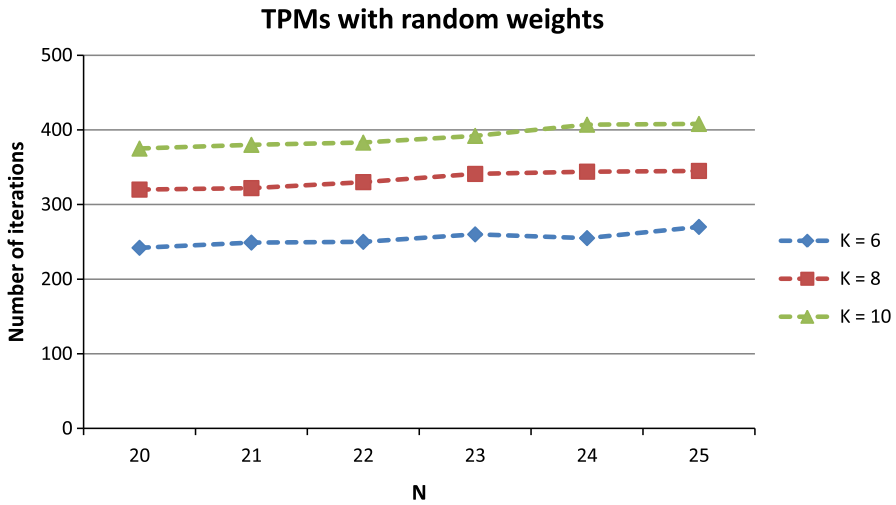
TPMs with random weights

Fig. 3 Synchronization of TPMs with randomly chosen weights

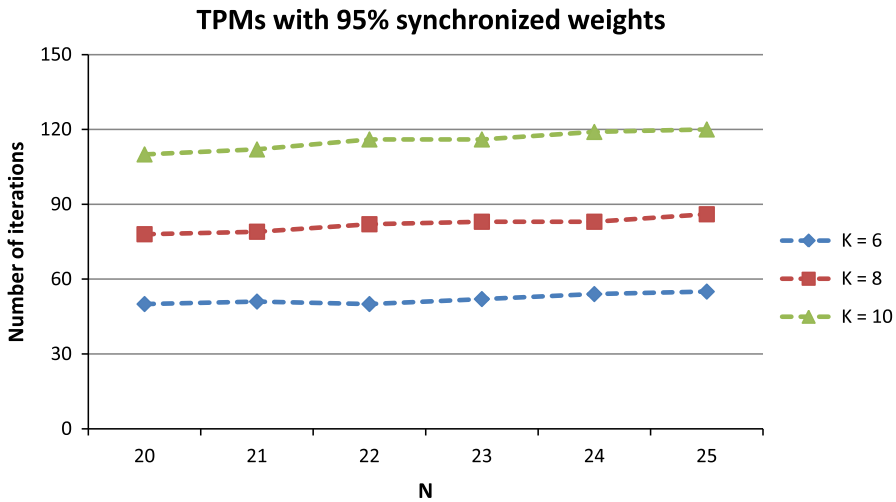## TPMs with 95% synchronized weights

Fig. 4 Synchronization of TPMs with 95% of synchronized weights at the beginning of the synchronization process

with randomly chosen weights and TPMs with very similar weights (differences at 1%) to be compared.

All the presented scenarios confirm that initially synchronized TPMs require far fewer iterations than artificial neural networks with randomly chosen weights. This feature increases the level of security significantly. The proposed solution is much more secure than neural cryptography where TPMs are used to establish cryptographic keys using random strings at the beginning of synchronization process.
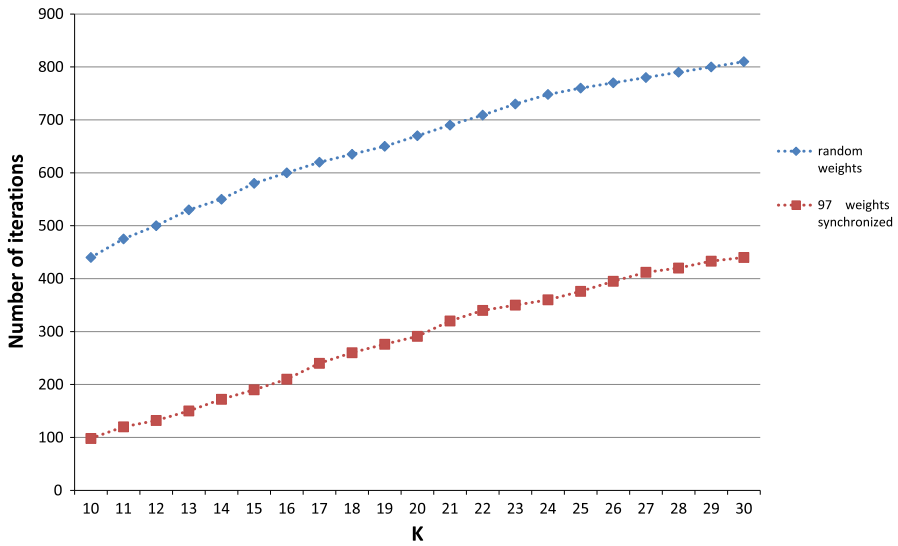
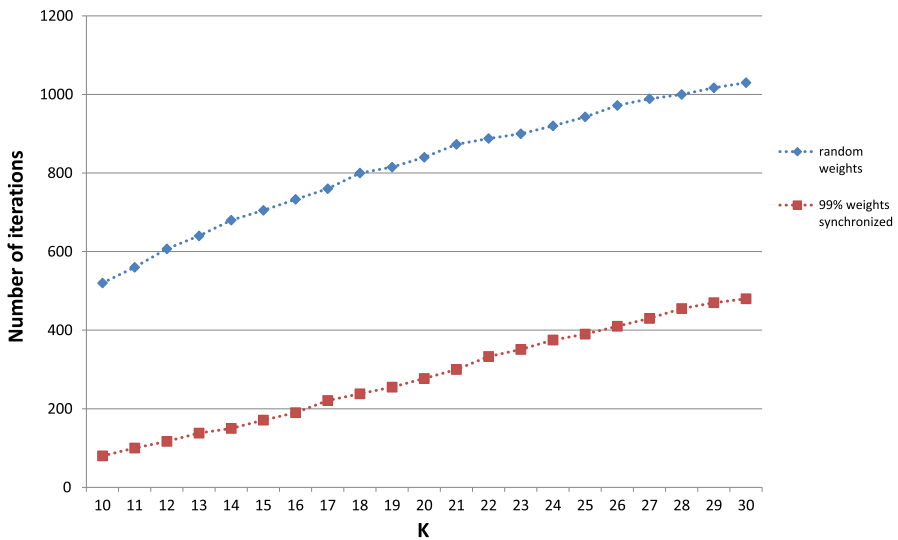**Fig. 5** Synchronization of TPMs with parameter $N = 30$ ($L = 2$, Hebbian learning algorithm)

Fig. 6 Synchronization of TPMs with parameter $N = 50$ ($L = 2$, Hebbian learning algorithm)

5.2 The efficiency of error correction based on TPMs

In order to compare the efficiency of the new TPM-based approach with other error correction algorithms (BBBSS [13] and Cascade [14]), additional tests were performed. Following recommendation [14], it was assumed that the block size for the BBBSS algorithm should have $\frac{0.73}{\text{QBER}}$ bits. The blocks in the first pass of the Cascade algorithm are of the same length. Tests were performed for different distributions of errors in

Table 1 Comparison of the error correction algorithms

	BBBSS	Cascade	TPM-based
Number of iterations (*Key length* = 500 bits, QBER = 5%)	213	181	120
Number of iterations (*Key length* = 600 bits, QBER = 3%)	189	150	98

the key. It was also assumed that both algorithms corrected all errors after four passes. Verifying the efficiency of the tested algorithms, two different keys were chosen:

- *key length* = 500 bits with QBER = 5% and
- *key length* = 600 bits with QBER = 3%.

The results are presented in Table 1. The table contains the average number of iterations for each tested error correction algorithm. In the case of BBBSS and Cascade algorithms, the iteration means a parity check of a single block. The values for TPM-based error correction come from Fig. 4 (TPM parameters: $N = 25$, $K = 10$, $L = 2$) and Fig. 5 (TPM parameters: $N = 30$, $K = 10$, $L = 2$). For both key lengths tested, the number of iterations is significantly lower for the new TPM-based approach than the BBBSS and Cascade algorithms.

6 Conclusions

In this article, a new idea for the key reconciliation method in quantum cryptography is presented. The proposal assumes that artificial neural networks (TPM machines) can be used to correct errors occurring in the quantum channel. Unlike neural cryptography, the new solution is characterized by fast synchronization of TPM machines. Typical values of QBER do not exceed a few percent; therefore, users need to correct only a small part of the key. This means that the TPM machines used for this purpose are close to synchronization and the learning process can finish quickly.

It was shown that the synchronization process in the new solution is much faster than in the case where TPM weights are chosen randomly (the typical scenario of neural cryptography). When the QBER value does not exceed a few percent, the synchronization process is several times faster. This significantly increases the level of security because of the problem with fast synchronization of eavesdropper's TPMs which must start from randomly generated weights. Therefore, the key reconciliation based on the synchronization of TPM machines can be a secure and efficient solution. The presented solution can replace the error correction algorithms currently used in the quantum key distribution process.

Although the risk of information leakage in the proposed solution is low, additional mechanisms should be used to increase the level of security. First of all, the use of the privacy amplification process after the error correction process is still recommended. Also, users can consider dividing a long key into shorter strings and perform error correction processes separately for each string. After that, the privacy amplification process should be performed on the whole key (concatenated by all the strings). This

approach decreases the risk of information leakage, even if an eavesdropper could get some information about a selected string.

Although TPM machines are not used for error correction of digital information, this approach can be used for key reconciliation in quantum cryptography. Mutual synchronization of TPMs dynamically adjusts their weights; therefore, the final weights are not predictable. Fortunately, this is a big advantage of key distribution, because a secure key for cryptographic purposes should be a random string of bits. It is a very special case when artificial neural networks can be used to correct errors.

Additionally, it is worth mentioning that error correction based on TPMs is resistant to currently known attacks using a quantum computer. This feature is likely to be particularly important in the near future.

References

1. Bouwmeester, D., Ekert, A., Zeilinger, A.: The Physics of Quantum Information. Quantum Cryptography, Quantum Teleportation, Quantum Computation. Springer, Berlin (2000)
2. Wootters, W.K., Zurek, W.H.: A single quantum cannot be cloned. Nature **299**, 802–803 (1982)
3. Lo, H.-K., Zhao, Y.: Quantum cryptography. Encycl. Complex. Syst. Sci. **8**, 7265–7289 (2009)
4. Mehic, M., Maurhart, O., Rass, S., Voznak, M.: Implementation of quantum key distribution network simulation module in the network simulator NS-3. Quantum Inf. Process. **16**, 253 (2017)
5. Dusek, M., Lutkenhaus, N., Hendrych, M.: Quantum Cryptography, Progress in Optics, vol. 49. Elsevier, Amsterdam (2006)
6. Scarani, V., Bechmann-Pasquinucci, H., Cerf, N.J., Dušek, M., Lütkenhaus, N., Peev, M.: The security of practical quantum key distribution. Rev. Modern Phys. **81**, 1301–1350 (2009)
7. Bennett, C.H., Brassard, G.: Public key distribution and coin tossing. In: Proceedings of the IEEE International Conference on Computers, Systems, and Signal Processing, pp. 175–179 (1984)
8. Bennett, C.H.: Quantum cryptography using any two nonorthogonal states. Phys. Rev. Lett. **68**, 3121–3124 (1992)
9. Ekert, A.: Quantum cryptography based on Bell's theorem. Phys. Rev. Lett. **67**, 661–663 (1991)
10. Bennett, C.H., Brassard, G., Mermin, D.N.: Quantum cryptography without Bell's theorem. Phys. Rev. Lett. **68**, 557 (1992)
11. Scarani, V., Acin, A., Ribordy, G., Gisin, N.: Quantum cryptography protocols robust against photon number splitting attacks for weak laser pulse implementations. Phys. Rev. Lett. **92**, 057901 (2004)
12. Niemiec, M., Pach, A.: Management of security in quantum cryptography. IEEE Commun. Mag. **51**, 36–41 (2013)
13. Bennett, C.H., Bessette, F., Brassard, G., Salvail, L., Smolin, J.: Experimental quantum cryptography. J. Cryptol. **5**, 3–28 (1992)
14. Brassard, G., Salvail, L.: Secret-Key Reconciliation by Public Discussion, pp. 410–423. Springer, Berlin (1994)
15. Furukawa, E., Yamazaki, K.: Application of existing perfect code to secret key reconciliation. In: Proceedings of International Symposium on Communication and Information Technologies, pp. 397–400 (2001)
16. Buttler, W.T., Lamoreaux, S.K., Torgerson, J.R., Nickel, G.H., Donahue, C.H., Peterson, C.G.: Fast, efficient error reconciliation for quantum cryptography. Phys. Rev. A **67**, 052303 (2003)

17. Carter, L., Wegman, M.N.: Universal classes of hash functions. J. Comput. Syst. Sci. **18**, 143–154 (1979)
18. Bennett, C.H., Brassard, G., Robert, J.M.: Privacy amplification by public discussion. SIAM J. Comput. **17**, 210–229 (1988)
19. Mehic, M., Niemiec, M., Voznak, M.: Calculation of the key length for quantum key distribution. Elektron. Ir Elektrotech. **21**, 81–85 (2015)
20. Niemiec, M., Romanski, L., Swiety, M.: Quantum cryptography protocol simulator. Commun. Comput. Inf. Sci. (CCIS) **149**, 286–292 (2011)
21. Kanter, I., Kinzel, W.: The theory of neural networks and cryptography. Phys. Commun. **25**, 631–642 (2003)
22. Hadke, P.P., Kale, S.G.: Use of Neural Networks in cryptography: a review. In: 2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare (Startup Conclave), Coimbatore, pp. 1–4 (2016)
23. Chakraborty, S., Dalal, J., Sarkar, B., Mukherjee, D.: Neural synchronization based secret key exchange over public channels: a survey. In: 2014 International Conference on Signal Propagation and Computer Technology (2014)
24. Kriesel, D.: A brief introduction to neural networks. http://www.dkriesel.com (2017). Accessed 15 Apr 2019
25. Mislovaty, R., Klein, E., Kanter, I., Kinzel, W.: Security of neural cryptography. In: 11th IEEE International Conference on Electronics, Circuits and Systems (ICECS), pp. 219–221 (2004)
26. Ruttor, A., Kinzel, W., Kanter, I.: Dynamics of neural cryptography. Phys. Rev. E **75**, 056104 (2007)
27. Rosen-Zvi, M., Kanter, I., Kinzel, W.: Cryptography based on neural networks—analytical results. J. Phys. A Math. Gen. **35**, 707 (2002)
28. Kanter, I., Kinzel, W., Kanter, E.: Secure exchange of information by synchronization of neural networks. EPL (Europhys. Lett.) **57**, 141 (2002)
29. Volkmer, M., Wallner, S.: Tree parity machine rekeying architectures. IEEE Trans. Comput. **54**(4), 421–427 (2005)
30. Wang, D.: Neural synchronization using genetic algorithm for secure key establishment. J. Eng. Sci. Technol. Rev. **8**, 152–156 (2015)
31. Mislovaty, R., Perchenok, Y., Kanter, I., Kinzel, W.: Secure key-exchange protocol with an absence of injective functions. Phys. Rev. E **66**, 066102 (2002)
32. Rosen-Zvi, M., Klein, E., Kanter, I., Kinzel, W.: Mutual learning in a tree parity machine and its application to cryptography. Phys. Rev. E **66**, 066135 (2002)
33. Kinzel, W., Rujan, P.: Improving a network generalization ability by selecting examples. Europhys. Lett. **13**, 473–477 (1990)
34. Ruttor, A., Kinzel, W., Kanter, I.: Neural cryptography with queries. J. Stat. Mech. Theory Exp. **2005**, P01009 (2005)
35. Vallone, G., et al.: Experimental satellite quantum communications. Phys. Rev. Lett. **115**, 040502 (2014)
36. Yin, Juan, et al.: Satellite-to-ground entanglement-based quantum key distribution. Phys. Rev. Lett. **119**, 200501 (2017)
37. Dixon, A.R., et al.: 77 day field trial of high speed quantum key distribution with implementation security. In: 6th International Conference on Quantum Cryptography (QCrypt) (2016)
38. Liao, S.-K., et al.: Satellite-to-ground quantum key distribution. Nature **549**, 43–47 (2017)
39. Javurek, M., Turčaník, M.: Synchronization of two tree parity machines. In: 2016 New Trends in Signal Processing (NTSP), pp. 1–4 (2016)
40. Othman, K.M.Z., Jammas, M.H.: Implementation of neural-cryptographic system using FPGA. J. Eng. Sci. Technol. **6**, 411–428 (2011)
41. Gomez, H., Reyes, O., Roa, E.: A fully synthesized key establishment core based on tree parity machines in 65 nm CMOS. In: 12th Conference on Ph.D. Research in Microelectronics and Electronics (PRIME), pp. 1–4 (2016)
42. Muhlbach, S., Wallner, S.: Secure and authenticated communication in chip-level microcomputer bus systems with tree parity machines. In: International Conference on Embedded Computer Systems: Architectures, Modelling and Simulation, pp. 201–208 (2007)
43. Saballus, B., Volkmer, M., Wallner, S.: Secure group communication in ad-hoc networks using tree parity machines. Commun. Distrib. Syst. **1**, 1–12 (2007)

44. Chen, T., Yan, D., Bai, S.: TinyTPM: a novel lightweight key agreement scheme for sensor networks. In: 2009 WRI International Conference on Communications and Mobile Computing, Yunnan, pp. 518–522 (2009)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.