

I. Kanter

Minerva Center, Department of Physics
Bar-Ilan University
Ramat-Gan, 52900, Israel
E-mail: kanter@mail.biu.ac.il

W. Kinzel

Institut für Theoretische Physik
Universität Würzburg
Würzburg, D-97074, Germany
E-mail: kinzel@physik.uni-wuerzburg.de

The Theory of Neural Networks and Cryptography

Received March 31, 2003

A connection between the theory of neural networks and cryptography is presented. A new phenomenon, namely synchronization of neural networks, is leading to a new method of exchange of secret messages. Two artificial networks being trained by the Hebbian learning rule on their mutual outputs develop an antiparallel state of their synaptic weights. The synchronized weights are used to construct an ephemeral key exchange protocol for the secure transmission of secret data. The complexity of the generation of the secure channel is linear with the size of the network. An attacker who knows the protocol and all details of any transmission of the data finds it difficult to decrypt the secret message.

1. Introduction

The ability to build a secure channel is one of the most challenging fields of research in modern communication. Since the secure channel has many applications, in particular for mobile phone, satellite and internet-based communications, there is a need for fast, effective and secure transmission protocols [1]. Here we present a novel principle of a cryptosystem based on a new phenomenon which we observe for artificial neural networks.

The goal of cryptography is to enable two partners to communicate over an insecure channel in such a way that an attacker cannot understand and decrypt the transmitted message. In a general scenario, the message is encrypted by the sender through a key E_k , and the result, the ciphertext, is sent over the channel. A third party, eavesdropping on the channel, should not be able to determine what the message was. However, the recipient who knows the encryption key can decrypt the ciphertext using his private key D_k .

In a *private-key* system the recipient has to agree with the sender on a secret key E_k , which requires a hidden communication prior to the transmission of any message. In a *public-key* system, on the other side, the key E_k is published and a hidden communication is not necessary. Nevertheless, an attacker cannot decrypt the transmitted message since it is computationally infeasible to invert the encryption

function without knowing the key D_k . In a *key-exchange protocol*, both partners start with private keys and transmit — using a public protocol — their encrypted private keys which, after some transformations, leads to a common secret key. In most applications a public-key system is used which is based on number theory where the keys are long integers [1, 2].

In this report we suggest a novel cryptosystem. It is a key-exchange protocol which uses neither number theory nor a public key, but is based on a learning process of neural networks: The two participants start from a secret set of vectors $E_k(0)$ and $D_k(0)$ without knowing the key of their partner. By exchanging public information the two keys develop into a common time dependent key $E_k(t) = -D_k(t)$, which is used to encrypt and decrypt a given message. An attacker who knows the algorithm and observes any exchange of information finds it difficult to reveal the keys $E_k(t)$ and $D_k(t)$. Our method is based on a new phenomenon: Synchronization of neural networks by mutual learning [3].

Simple models of neural networks describe a wide variety of phenomena in neurobiology and information theory [4, 5, 6]. Artificial neural networks are systems of elements interacting by adaptive couplings which are trained from a set of examples. After training they function as content addressable associative memory, as classifiers or as prediction algorithms.

Two feedforward networks can synchronize their synaptic weights by exchanging and learning their mutual outputs for given common inputs. Surprisingly, synchronization is fast; the number of bits required to achieve perfect alignment of the weights is lower than the number of components of the weights. After synchronization, the synaptic weights define the common time dependent private key $E_k(t) = -D_k(t)$. The complexity of our cryptosystems scales linearly with the size of the network (=number of bits of the keys). With respect to possible attacks, we find that tracking the weights of one of the networks by the attacker is a difficult problem. In summary, from this new biological mechanism one can construct efficient encryption systems using keys which change permanently.

The paper is organized as follows. In section II we speculate on the possible biological meaning of this new bridge between the theory of neural networks and cryptography. In section III the definition of the architecture and the dynamical rules are presented. In section IV the synchronization time between the parties derived from simulations are presented, while in section V the efficiency of a simple attacker is examined. In section VI some possible generalizations of our secure channel are briefly discussed.

2. Possible biological relevance

Synchronization is the subject of recent research in neuroscience [7, 8, 9, 10, 11], where, for instance, in experiments on cats and monkeys it was found that the spike activity of neurons in the visual cortex has correlations which depend on the kind of optical stimulus shown to the animal [12]. The phenomenon described here suggests that synchronization can be used by biological neuronal networks or by networks

of the immune system to exchange secure information between different parts of an organism.

The interpretation of the synchronization process as a mechanism to build a secure channel is a controversial subject. The need in biological systems for the secure transmission of secret data which is incorporated by a biological mechanism to encrypt and to decrypt a data is in question.

At that point we present our viewpoint on the possibility of such a biological realization. In table I we demonstrate a partial list of operations commonly used by a user of a personal computer (or by human activities in daily life). In comparison we present the implications of the research of modern biology, indicating that similar operations such as ‘copy’, ‘paste’, ‘insert’, ‘cut’ etc. exist in the activity of a cell and DNA. It is clear that biological operations are restricted to biological purposes, hence they are very limited in comparison to the variety of options and flags suggested by any primitive software.

A possible conclusion of the above comparison would be the following paradigm. *All operations that can be found at the macroscopic level of human activity can also be found in biological activity on the microscopic level.* Hence, the adaptation of this paradigm indicates that the secure transmission of data also exists on the microscopic activities of a biological system. Encryption and decryption of biological signals should also be found in the microscopic functioning of human activities. The level of security, of course, should be rescaled with the expected capability of a sophisticated biological attacker.

Table 1. A similarity between functions operated by computers and activities operated by biological systems on the microscopic level.

Operation	Computers	Biology
cut	√	√
copy	√	√
paste	√	√
find	√	√
replace	√	√
encrypt	√	???
decrypt	√	???

3. Definition: network and dynamical rules

In the following we introduce and investigate a simple model which shows the properties sketched above. The architecture used by the recipient and the sender is a two-layered perceptron, exemplified here by a parity machine (PM) with K hidden units. More precisely, the size of the input is KN and its components are denoted by x_{kj} , $k = 1, 2, \dots, K$ and $j = 1, \dots, N$. For simplicity, each input unit takes binary values, $x_{kj} = \pm 1$. The K binary hidden units are denoted by y_1, y_2, \dots, y_K .

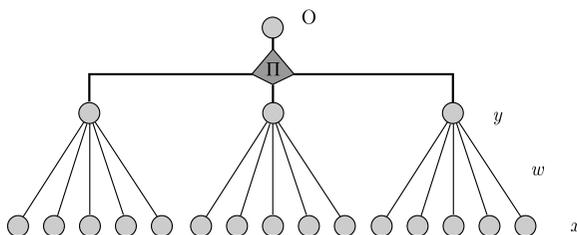


Fig. 1. Architecture of the networks: $3N$ input units x are transmitted by three weight vectors w to three hidden units y . The final output bit O is the product of the hidden units.

Our architecture is characterized by non-overlapping receptive fields (a tree), where the weight from the j th input unit to the k th hidden unit is denoted by w_{kj} , and the output bit O is the product of the state of the hidden units (see Fig. 1). For simplicity we discuss PMs with three hidden units $K = 3$. We use integer weights bounded by L , i.e., w_{kj} can take the values $-L, -L + 1, \dots, L$.

The secret information of each of the two partners is the initial values for the weights, w_{kj}^S and w_{kj}^R , for the sender and the recipient, respectively. It consists of $6N$ integer numbers, $3N$ of the recipient and $3N$ of the sender. Sender and recipient do not know the initial numbers of their partners, which are used to construct the common secret key.

Each network is then trained with the output of its partner. At each training step, for the synchronization as well as for the encryption/decryption step, a new common public input vector (x_{kj}) is needed for both the sender and the recipient. For a given input, the output is calculated in the following two steps. In the first step, the state of the three hidden units, $y_k^{S/R}, k = 1, 2, 3$, of the sender and the recipient are determined from the corresponding fields

$$y_k^{S/R} = \text{sign}\left[\sum_{j=1}^N w_{kj}^{S/R} x_{kj}\right] \tag{3.1}$$

In the case of zero field, $\sum w_{kj}^{S/R} x_{kj} = 0$, the sender/recipient sets the hidden unit to $1/-1$. In the next step the output $O^{S/R}$ is determined by the product of the hidden units, $O^{S/R} = y_1^{S/R} y_2^{S/R} y_3^{S/R}$.

The sender is sending its output (one bit) to the recipient, the recipient is sending its output to the sender and both networks are trained with the output of its partner. In the event that they do not agree on the current output, $O^S O^R < 0$, the weights of the sender/recipient are updated according to the following Hebbian learning rule [6, 13]

$$\begin{aligned} \text{if } \left(O^{S/R} y_k^{S/R} > 0\right) \text{ then } w_{kj}^{S/R} &= w_{kj}^{S/R} - O^{S/R} x_{kj} \\ \text{if } \left(|w_{kj}^{S/R}| > L\right) \text{ then } w_{kj}^{S/R} &= \text{sign}(w_{kj}^{S/R}) L \end{aligned} \tag{3.2}$$

Only weights belonging to the one (or three) hidden units which are in the same state as that of their output unit are updated, in each of the two networks. Note that by using this dynamical rule, the sender is trying to imitate the response of the recipient and the recipient is trying to imitate that of the sender [16].

There are three main ingredients in our model which are essential for a secure key exchange protocol: Firstly, from the knowledge of the output, the internal representation of the hidden units is not uniquely determined because there is a four fold degeneracy (for the output +1 there are four internal representations for the three hidden units $(1, 1, 1)$, $(1, -1, -1)$, $(-1, 1, -1)$, $(-1, -1, 1)$). As a consequence, an observer cannot know which of the weight vectors is updated according to equation (2). Secondly, we have chosen the parity machine since there is a lack of correlation between the state of the hidden unit and the output bit (for output bit 1, for instance, for each hidden unit there are two internal representations with state 1 and two internal representations with state -1). This observation favors the PM over other multilayer networks. Thirdly, since each component is bounded by L , an observer cannot simply invert the sum of equation (2); the network forgets [15].

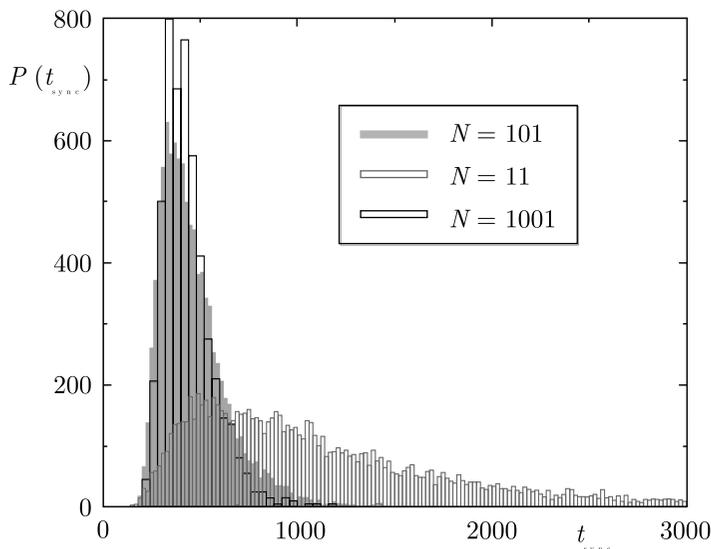


Fig. 2. Distribution of synchronization time t_{sync} for three sizes N of the two networks.

4. Synchronization process

We find that the two PMs learning from each other are able to synchronize, at least for some parameters K, L and N [16]. Our simulations show that after an initial relatively short transient time the two partners align themselves into antiparallel states. It is easy to verify from our learning rule that as soon as the two networks are synchronized they remain so forever. The number of time steps required to

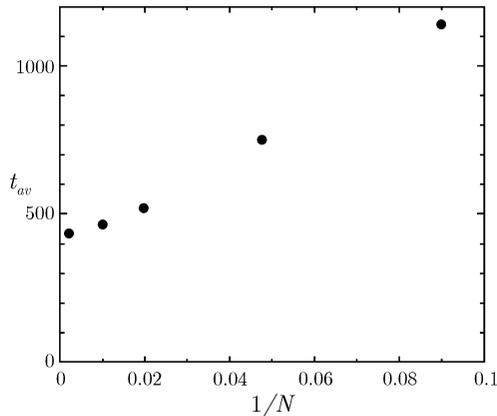


Fig. 3. Average synchronization time as a function of $1/N$, for system size $N = 11, 21, 51, 101, 1001$.

reach this state depends on the initial weight vectors and on the sequence of random inputs, hence it is distributed. Fig. 2 shows the distribution of synchronization time obtained from at least 1000 samples. It is evident that two communicating networks synchronize in a rather short time. The average synchronization time t_{av} decreases with increasing size N of the system (see Fig. 3); it seems to converge to $t_{av} \simeq 410$ for large networks. This observation was recently confirmed by an analytical solution of the presented model [17]. Surprisingly, in the limit of large N one needs to exchange only about 400 bits to obtain agreement between $3N$ components. However, one should keep in mind that the two partners do not learn the initial weights of each other, they are just attracted to a dynamical state with opposite weight vectors.

Note that this fast synchronization, independent of the size of the network, may also serve as an efficient biological mechanism to initialize two networks with the same initial conditions - the strength of the synapses. The number of operations (updates) per bit is very small, ~ 200 for $L = 3$ and only ~ 10 for $L = 1$ where synchronization is achieved after a few dozen steps. In artificial computers one can achieve a similar goal by the synchronization of the initial seed of a random number generator between the two partners. The complexity per bit (among $3N$) is governed by the complexity to generate a random number.

As soon as the weights of the sender and the recipient are antiparallel the public initialization of our private-key cryptosystem is terminated successfully and the encryption of the message starts. There are then two possibilities in choosing an algorithm: First, use a conventional encryption algorithm, for example a stream cipher like the well-known Blum-Blum-Shub bit generator [1]. In this case the seed for this pseudo-random number generator is constructed from our weight vector after synchronization. Note that this bit generator is secure: even without a secret message one cannot guess the next bit from a polynomial number of consecutive output bits. Second, use the PM with time-dependent weights itself for a stream cipher by multiplying its output bit with the corresponding bit of the secret data.

In the case of the PM, the complexity of the encryption/decryption processes scales linearly with the size of the transmitted message, whereas the complexity of the synchronization process does not scale with the size of the network. Hence our construction is a linear cryptosystem [18].

5. An attacker

We now examine a possible attack on our cryptosystem. An attacker eavesdropping on the line knows the algorithm as well as the actual mutual outputs, hence he knows in which time steps the weights are changed. In addition, the attacker knows the input x_{kj} as well. However, the attacker does not know the initial conditions of the weights of the sender and the recipient. As a consequence, even for the synchronized state, the internal representations of the hidden units of the sender and the recipient are hidden from the attacker and he does not know which are the weights participating in the learning step. For random inputs all four internal representations appear with equal probability in any stage of the dynamical process, hence for t training steps there are 4^t possibilities to select internal representations [14].

Therefore, on the time scale of synchronization, the observing network has difficulty in obtaining complete knowledge about the other two networks. We have simulated the following basic attack on our cryptosystem. The architecture of an attacker is identical to the architecture of the partners, the recipient and the sender. The dynamics of this attacker tries to imitate the moves of one of the parties, the sender for instance. The observing network is trained with the input vector and output bit of the sender, and the training step is performed only if sender and recipient disagree with each other [3].

The learning rule may be considered for each component of the weight vectors as a kind of biased random walk with reflecting boundaries. Therefore, for very long times, the observer may take the weight vector of the other network by chance. The distribution of the ratio between the time t_{sync} the sender and recipient need to synchronize and the learning time t_{learn} of this basic attacker needs for complete overlap is shown in Fig. 4. For $N = 101$ the average learning time is a factor of about 125 larger than the corresponding synchronization time. In addition, with increasing system size the tail of the distribution for larger ratios is reduced.

Hence the time to synchronize by chance is very long and in the example discussed here it is of order $O(10^5)$ [19]. The heart of our cryptosystem is that synchronization is a much simpler task than tracking by an observer.

Recently the dynamic of such simple attackers was formulated and examined analytically for large networks, $N \gg 1$. The analytical results for the learning time were found to be in agreement with the results of simulations [17].

Possible advanced attacks were recently examined in [20], where it seems that the presented cryptosystem is breakable under an attack based on an ensemble of attackers using more advanced strategies — dynamical rules. The question of whether a practical secure channel based on a public protocol for the synchronization of the parties exists is in question. We would like to conclude this section and to repeat our

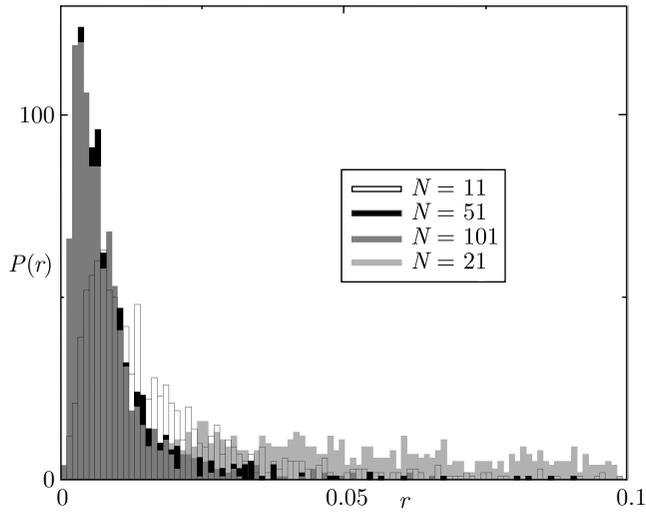


Fig. 4. Frequency of the ratio r between synchronization and learning times.

abovementioned statement that *the required level of security for biological networks should be rescaled with the expected capability of a sophisticated biological attacker.*

6. Generalization

Our key exchange protocol can be generalized in the following directions as is briefly described below.

Bit-Packages: An important issue for the implementation of our cryptosystem is to accelerate the synchronization process from hundreds of time steps to a few dozen while maintaining the security of our channel. Surprisingly, both of these two goals can be achieved simultaneously by sending bit packages (BP). In this scenario the process contains the following steps: (a) The sender and the recipient generate $B > 1$ common inputs. (b) The sender and the recipient calculate the output of their PM for the set of B inputs and store the B sets of the corresponding values y_{ki} ($i = 1, \dots, B$) of the hidden units (the internal representations) (c) The transmission of mutual information; the sender/recipient sends a package consisting of B bits ($b_i^{S/R}$) to the recipient/sender. (d) The sender and the recipient update their weights using the same learning rule as for $B = 1$: In the case where bit $b_i^S \neq b_i^R$ the learning process takes place as before using the corresponding internal representations. The synchronization time is dramatically reduced, as shown in Fig. 5. For instance, for $N = 21$, $K = 3$, $L = 3$, synchronization is achieved after 12 bit packages if the size of the package is larger than $B \geq 32$.

Several partners: Up to now we have discussed the scenario of two partners exchanging secret information. But our method can be extended to the case of a pool of several participants sharing secret information. In this case we would like to

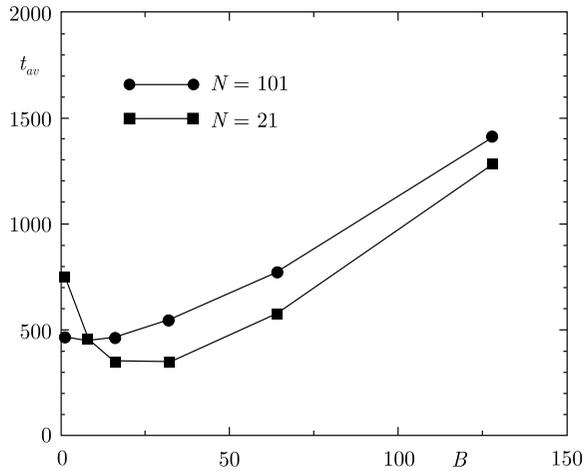


Fig. 5. Total number of transmitted bits until synchronization. B is the number of bits in each bit package exchanged between sender and recipient.

transmit information among several partners, but only if all of them are available to act cooperatively. The mutual information is the global flow of information, namely the output bits of all participants. Assume that we would like to build a common private key among an even number Q of partners, each of them represented by a PM. In each time step each of the Q partners is exposed to the Q outputs O^1, \dots, O^Q . In case $O^q = -O^{q+1}$ for $q = 1, \dots, Q - 1$ the learning step is performed. In our simulations we find, for instance, that for $N = 61$, $K = 3$ and $L = 3$ for $Q = 4, 6, 8$ the average synchronization time is 640, 1023, 2430 respectively.

Generation of inputs: A common public input can be generated following a public seed number for a random number generator. The main disadvantage of this method is that the number of new random variables necessary to define a common input is of order $O(N)$. A simple way to overcome this difficulty is to shift all of the $3N$ input units one place to the right and to set the left-most input equal, for instance, to the output of the sender [21, 22].

Permutation of the weights: In order to increase the security of our channel, one can apply the following types of permutations: (a) To permute a fraction (or all) of the weights belonging to an updated hidden units. (b) A global permutation of weights belonging to an updated hidden unit with weights belonging to other hidden units. (c) The fraction of permuted weights increases as the parties approach synchronization. It is clear that the permutations are following a public protocol, but the internal presentations of the parties are hidden. Our simulations indicate the following main results: (a) Synchronization between the parties is achievable under these classes of permutations, but t_{synch} increases. (b) The overlap between an attacker and one of the parties at the synchronization time is reduced by the permutation among the weights, and the system is more robust with respect to advanced attacks [20].

This work profitted from enjoyable collaborations with Richard Metzler and Michal Rosen-Zvi. The work is supported in part by the German Israel Science Foundation (GIF).

References

- [1] D. R. Stinson, *Cryptography: Theory and Practice*, CRC Press (1995).
- [2] R. L. Rivest, A. Shamir and L. Adelman, *Comm. of the ACM*, **21**, 120 (1978).
- [3] I. Kanter, W. Kinzel and E. Kanter, *Secure exchange of information by synchronization of neural networks*, *Europhys. Lett.*, **57**, 141 (2002).
- [4] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*, Addison Wesley, Redwood City (1991)
- [5] A. Engel, and C. Van den Broeck, *Statistical Mechanics of Learning*, Cambridge University Press (2001)
- [6] M. Opper and W. Kinzel, *Statistical Mechanics of Generalization*, *Models of Neural Networks III*, ed. by E. Domany and J.L. van Hemmen and K. Schulten, 151–20, Springer Verlag Heidelberg (1995).
- [7] M. Castelo-Branco, R. Goebel, S. Neunsehwander and W. Singer, *Nature*, **405**, 685 (2000).
- [8] A. K. Engel, P. König and W. Singer, *Nature*, **405**, 685 (2000).
- [9] J. K. Lin, U. Ernst and T. J. Sejnowski, *Networks: Computation in Neural Systems*, **9**, 333 (1998).
- [10] R. Ritz and T. J. Sejnowski, *Artificial Neural Networks -ICANN 97*, 7th International Conference Proceedings, Eds by W. Gerstner, A. Germond, M. Hasler and J. D. Nicoud, Springer-Verlag, page 79 (1997).
- [11] A. Riehle, S. Grun, M. Diesmann and A. Aertsen, *Science*, **278**, 1950 (1997).
- [12] C. M. Gray, P. König, A. K. Engel and W. Singer, *Nature*, **338**, 334 (1989).
- [13] M. Biehl and N. Caticha: *Statistical Mechanics of On-line Learning and Generalization*, *The Handbook of Brain Theory and Neural Networks*, ed. by M. A. Arbib, MIT Press, Berlin (2001).
- [14] E. Barkai and I. Kanter, *Europhys. Lett.*, **17**, 181 (1991).
- [15] J. L. van Hemmen, G. Keller and R. Kühn, *Europhys. Lett.*, **5**, 663 (1988).
- [16] In fact, such a scenario has recently been studied by R. Metzler, W. Kinzel and I. Kanter, *Phys. Rev. E* **62**, 2555 (2000), and W. Kinzel, R. Metzler and I. Kanter *J. Phys. A* **33** L141 (2000), for two perceptrons with continuous weights and normalized weight vectors. However, we find that for such a system an outside observer can find the internal state of both perceptrons by learning the communication. Furthermore, the normalization of the weight vector after each learning step is a non-local operation and is violating the linear complexity of our cryptosystem.
- [17] M. Rosen-zvi, I. Kanter and W. Kinzel cond-mat/0202350

- [18] As for the hardware implementation of our cryptosystem, the calculation of the local fields, eq. (1), are similar to a linear filter which can be implemented in parallel using existing technologies. Hence, the complexities of calculating local fields of the hidden units are taken as order $O(1)$ and the synchronization process is of $O(1)$ only.
- [19] Note that the average learning time for $N = 101$ scales exponentially with L and for $L = 1, 2, 3, 4$, was found in simulations to be equal to 58, 1480, 90380, $\sim 10^7$. For a comparison the average synchronization time for $L = 4$ is ~ 950 only. These results indicate that our construction may be secure even as a stream cipher.
- [20] I. Kanter and W. Kinzel (unpublished).
- [21] E. Eisenstein, I. Kanter, D. Kessler and W. Kinzel, Phys. Rev. Lett., **74**, 6 (1995).
- [22] I. Kanter, D. Kessler, A. Priel and E. Eisenstein, Phys. Rev. Lett., **25**, 2614 (1995).