Successful attack on permutation-parity-machine-based neural cryptography

Luís F. Seoane
Bernstein Center for Computational Neurosciences, Technische Universität Berlin, Germany

Andreas Ruttor
Artificial Intelligence Group, Technische Universität Berlin, Germany

An algorithm is presented which implements a probabilistic attack on the key-exchange protocol based on permutation parity machines. Instead of imitating the synchronization of the communicating partners, the strategy consists of a Monte Carlo method to sample the space of possible weights during inner rounds and an analytic approach to convey the extracted information from one outer round to the next one. The results show that the protocol under attack fails to synchronize faster than an eavesdropper using this algorithm.

PACS numbers: 84.35.+i, 87.18.Sn, 89.70.-a, 05.10.Ln

Interacting feed-forward neural networks can synchronize by mutual learning [1, 2]. If two networks A and B are trained with examples consisting of random inputs and the corresponding output of the other one, their weight vectors converge. In the case of tree parity machines (TPMs) this mutual synchronization of A and B requires fewer examples than training a third network E successfully [3–7]. Based on this effect a TPM-based neural key-exchange protocol has been developed [8–11] and shown to be useful in embedded devices [12, 13] as well as being sufficiently secure against several attacks [7].

Recently, a variant of neural cryptography has been presented in Ref. [14] which uses permutation parity machines (PPMs) [15] instead of TPMs. This change increases the robustness of the key-exchange protocol against the attacks which have been tried on the TPM-based algorithm before [16–18]. However, it also reduces the number of possible values per weight from $2L+1 \geq 3$ to 2, so that other attacks become more feasible. This is especially true for the *probabilistic attack*, which has been suggested by Ref. [19], but not implemented up to now. We have used this idea and developed an attack method especially suited for PPM-based neural cryptography. In this Rapid Communication, we describe our attack and present results indicating its success.

A PPM is a neural network consisting of two layers: There are K hidden units in the first layer, each of which has an independent receptive field of size N, and only one neuron in the second layer. Its KN inputs $x_{i,j}$ with indices $i = 1, \ldots, N$ and $j = 1, \ldots, K$ are binary: $x_{i,j} \in \{0, 1\}$. In order to simplify the notation they are combined into input vectors $\mathbf{x}_j = (x_{1,j}, \ldots, x_{N,j})^\top$ or the input matrix $X = (\mathbf{x}_1, \ldots, \mathbf{x}_K)$ where appropriate.

The weights $w_{i,j}$ are selected elements from the state vector $\mathbf{s}$ of the PPM, which consists of $G \gg KN$ elements $s_i \in \{0, 1\}$. For that purpose a matrix π of size $N \times K$ containing numbers $\pi_{i,j} \in \{1, \ldots, G\}$ is used, so that $w_{i,j} = s_{\pi_{i,j}}$. The weight vector $\mathbf{w}_j = (w_{1,j}, \ldots, w_{N,j})^\top$ then determines the mapping from the input vector $\mathbf{x}_j$ to the state $\sigma_j \in \{0, 1\}$ of the j-th hidden unit. First, the *vector local field* $\mathbf{h}_j$ is calculated as the one-by-one logical XOR operation

$$\mathbf{h}_j = \mathbf{x}_j \oplus \mathbf{w}_j \quad \Longrightarrow \quad h_{i,j} = x_{i,j} \oplus w_{i,j} \tag{1}$$

of the bits in $\mathbf{x}_j$ and $\mathbf{w}_j$. Then the unit becomes active, $\sigma_j = 1$, if the majority of elements in $\mathbf{h}_j$ is equal to 1, otherwise it stays inactive, $\sigma_j = 0$:

$$\sigma_j = \Theta\left(h_j - \frac{N}{2}\right), \tag{2}$$

where

$$h_j = \sum_{i=1}^{N} h_{i,j} \tag{3}$$

denotes the *scalar local field* and

$$\Theta(x) = \begin{cases} 0 & \text{for } x \leq 0, \\ 1 & \text{for } x > 0, \end{cases} \tag{4}$$

is the Heaviside step function. Finally, the total output of the PPM is calculated as the binary state $\tau \in \{0, 1\}$ of the single unit in the second layer which is set to the parity

$$\tau = \bigoplus_{j=1}^{K} \sigma_j \tag{5}$$

of the hidden states σ_j.

When implementing the synchronization task, two PPMs (A and B) designed with the same settings (i.e., with same N, K, and G) are provided. The synchronization will succeed after several *inner* and *outer rounds*, which are described below.

For each inner round, the elements of the matrix π and the input vectors $\mathbf{x}_j$ are drawn randomly and independently from their corresponding value set. These quantities are provided publicly to all PPMs, which includes even an attacker E. Then both A and B compute their outputs τ^{A} and τ^{B} and if they agree ($\tau^{\mathrm{A}} = \tau^{\mathrm{B}}$), they store the state σ_1^{A} and σ_1^{B} of their first hidden units in a buffer, which remains private for each PPM.

Thus, there are as follows: public and common input vectors $\mathbf{x}_j$ and the π matrix; public, but not necessarily equal outcomes τ^{A} and τ^{B}; private, not necessarily equal state vectors $\mathbf{s}^{\mathrm{A}}$ and $\mathbf{s}^{\mathrm{B}}$; and private, not necessarily equal states of the hidden units σ_j^{A} and σ_j^{B}.

The inner rounds are repeated until the buffers where σ_1^A and σ_1^B are stored reach size G. Then, an outer round is completed and each buffer becomes the new state vector in the corresponding machine, substituting the old one. The dynamics of the PPMs are such that after each outer round the state vectors $\mathbf{s}^{\mathrm{A}}$, $\mathbf{s}^{\mathrm{B}}$ tend to be more alike, eventually reaching full synchronization $\mathbf{s}^{\mathrm{A}} = \mathbf{s}^{\mathrm{B}}$. The synchronization time t_{s} measured in the number of outer rounds is a random variable, as it depends on randomly chosen initial conditions and inputs. However, its mean value rises in a polynomial fashion with increasing size N of the input vector as well as growing size G of the state vector $\mathbf{s}$ [15].

Reported previous attacks on PPMs tried to mimic the behavior of the synchronizing networks by using a single machine or an ensemble [14]. They showed poor performance in guessing $\mathbf{s}^{\mathrm{A}}$ correctly. Namely, for the attacks on PPMs with $K = 2$ and $G = 128$ analyzed in Ref. [14], the probability of success did not exceed 10^{-5}.

In the following, we present the description of a different attack strategy. It does not pursue to mimic the synchronizing process, but to first guess the state vector of A (or B) during an outer round and consecutively to reproduce A's (or B's) behavior during the given round so that a fair guessing of the bits stored in the buffer and the subsequent $\mathbf{s}^{\mathrm{A}}$ for the next outer round can be done.

Some notation is introduced. The synchronizing parties A and B are eavesdropped by a third agent E, which implements its own PPM with state vector $\mathbf{s}^{\mathrm{E}}$ and output τ^{E}. Additionally, the attacker uses a *probabilistic state vector* $\mathbf{p}^{\mathrm{E}} = (p_1, \ldots, p_G)^\top$ to describe its knowledge about A's state vector $\mathbf{s}^{\mathrm{A}}$. Each element p_i is an approximation of the marginal probability $P(s_i^{\mathrm{A}} = 0|D)$ that the i-th bit of $\mathbf{s}^{\mathrm{A}}$ is 0 given all data D observed by E before, i.e., inputs and outputs of A and B, which have already been transmitted over the public channel.

At the beginning of the probabilistic attack previous information about $\mathbf{s}^{\mathrm{A}}$ is not available. Therefore, E starts with a neutral hypothesis and all p_i are initialized with the prior probability $P(s_i = 0) = 1/2$.

In each inner round an input X and a matrix π are provided to all PPMs. Then A and B calculate their outputs and communicate them publicly. This enables E to update $\mathbf{p}^{\mathrm{E}}$ based on the observed data X, π, and τ^{A}. For that purpose the posterior probability $P(s_i = 0|\mathbf{p}^{\mathrm{E}}, X, \pi, \tau^{\mathrm{A}})$ is estimated using a Monte Carlo approach, which is similar to approximate Bayesian computation [20].

This works by generating M state vectors $\mathbf{s}^{\mathrm{E}}$ which are compatible with the current observation as well as prior knowledge obtained in previous rounds. The G elements of a candidate $\mathbf{s}^{\mathrm{E}}$ are sampled independently from the Bernoulli distribution with probabilities $P(s_i = 0) = p_i$ and $P(s_i = 1) = 1 - p_i$. Of course, it is only necessary to draw bits s_i which are selected as weights by π. All others can be omitted without affecting the result. This shortcut speeds the sampling up considerably if $G \gg KN$. Then $\mathbf{s}^{\mathrm{E}}$ is plugged into E's PPM together with X and π in order to calculate τ^{E}. If E's output matches A's, $\tau^{\mathrm{E}} = \tau^{\mathrm{A}}$, the candidate is stored; if not, it is dismissed. This procedure goes on until M valid state vectors $\mathbf{s}^{\mathrm{E}}$ have been produced.

Afterward, the desired marginal posterior probability $P(s_i = 0|\mathbf{p}^{\mathrm{E}}, X, \pi, \tau^{\mathrm{A}})$ can be estimated as the relative frequency of $s_i = 0$ in the sample. The result is then used to update all p_i which have been selected as weights in the current round. The other elements of $\mathbf{p}^{\mathrm{E}}$ remain unchanged, because the attacker gained no information about the corresponding parts of $\mathbf{s}^{\mathrm{A}}$. Of course, this computation is repeated for the next inner round.

As the space of all weight matrices $W = (\mathbf{w}_1, \ldots, \mathbf{w}_K)$ is of size 2^{NK}, approximately 2^{NK-1} of them are compatible with a given X, π, and τ^{A}. Thus if the sampling algorithm generates $M \geq 2^{NK-1}$ state vectors, it would be similar to a brute force attack. But choosing such a large parameter M is only feasible for a very small number of weights.

Updating $\mathbf{p}^{\mathrm{E}}$ as described above has the effect that its elements p_i converge toward 0 or 1 after several rounds, so that finally M equal state vectors with

$$p_i = 0 \quad \Longrightarrow \quad s_i = 1, \tag{6}$$

$$p_i = 1 \quad \Longrightarrow \quad s_i = 0, \tag{7}$$

are sampled. However, defining

$$s_i^{\mathrm{E}*} = \begin{cases} 0 & \text{for } p_i > 1/2, \\ 1 & \text{for } p_i \leq 1/2, \end{cases} \tag{8}$$

as the most probable state provided $\mathbf{p}^{\mathrm{E}}$, the attack is considered a success as soon as $\mathbf{s}^{\mathrm{E}*} = \mathbf{s}^{\mathrm{A}}$ without regard to whether or not all the p_i have collapsed to 0 or 1.

In contrast, if one or more p_i have collapsed to the wrong value, E might be unable to achieve the desired output $\tau^{\mathrm{E}} = \tau^{\mathrm{A}}$ in a later round. Such a failure clearly indicates that the estimation of some p_i has gone wrong. In order to avoid an infinite loop in this case, only a finite number of attempts is made to generate M valid samples $\mathbf{s}^{\mathrm{E}}$. If the limit is reached, the element p_i of $\mathbf{p}^{\mathrm{E}}$ which is closest to collapse is reset to the neutral hypothesis, $p_i = 1/2$.

Usually, the algorithm will not be able to guess $\mathbf{s}^{\mathrm{A}}$ correctly in less than one outer round, therefore we need a mechanism to transfer the information gained during an outer round into the next one. Let $\mathbf{p}^{\mathrm{E}-}$ be the probabilistic state vector after applying the previous algorithm on all the inner rounds of a whole outer round. In order to transfer the information the attacker calculates the probability distribution for the state σ_1^{E} of the first hidden unit conditioned on the probabilistic state vector $\mathbf{p}^{\mathrm{E}-}$ as well as the input X and the matrix π for each of the inner rounds with $\tau^{\mathrm{A}} = \tau^{\mathrm{B}}$. The result is then

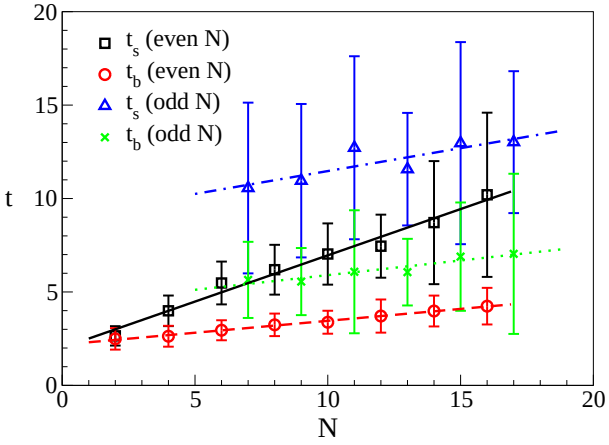

FIG. 1. (Color online) Synchronization time t_s and break time t_b measured in outer rounds as a function of N for PPMs with $K = 2$, $G = 128$. Symbols denote mean values and error bars denote standard deviations obtained in 100 runs (even N). Lines show the results of a fit using linear regression as given in Table I. For odd N around 25 out of 100 runs reaching $t = 30$ had to be aborted and discarded from the data set.

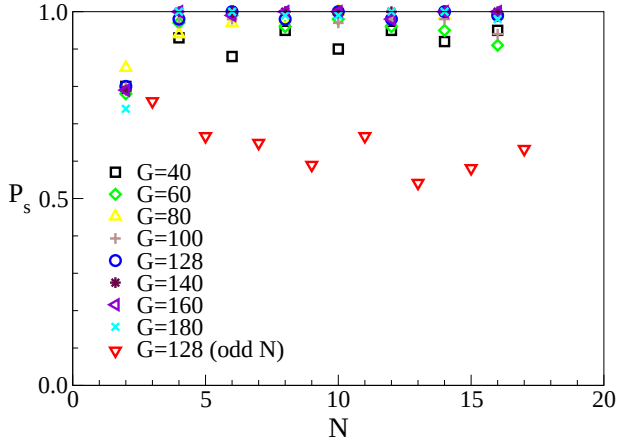

FIG. 2. (Color online) Success probability P_s of the attack as a function of N for PPMs with $K = 2$. Symbols denote the percentage of successful attacks found in 100 simulations (even N). For odd N around 25 out of 100 runs had to be stopped after 30 outer rounds without a clear result, i.e., $t_s > 30$ and $t_b > 30$. These simulations were not considered for calculating the probability of success P_s.

used to construct the probabilistic state vector $\mathbf{p}^{E+}$ for the start of the next outer round.

In the following we describe an algorithm to approximate the probability that a single hidden unit has internal state $\sigma_j = 0$ given $\mathbf{p}^E$ and the corresponding public information of an inner round. The output σ_j depends only on the number of 1s in the vector local field $\mathbf{h}_j$, which is equal to the scalar local field h_j. Here we approximate the probability distribution $P(h_j = n|\mathbf{p}^{E-}, X, \pi)$ of this quantity by a binomial distribution which uses the average probability of finding a 1 in $\mathbf{h}_j$ as a parameter

$$q_j = \frac{1}{N}\sum_{i=1}^{N}\left[x_{i,j}\, p_{\pi_{i,j}} + (1 - x_{i,j})(1 - p_{\pi_{i,j}})\right]. \qquad (9)$$

Then, the probability

$$P(\sigma_j = 0|\mathbf{p}^{E-}, X, \pi) = \sum_{n=0}^{N/2} P(h_j = n|\mathbf{p}^{E-}, X, \pi) \qquad (10)$$

of $\sigma_j = 0$ is given by

$$P(\sigma_j = 0|\mathbf{p}^{E-}, X, \pi) = \sum_{n=0}^{N/2} \binom{N}{n} q_j^n (1 - q_j)^{N-n}. \qquad (11)$$

Finally, the attacker stores this result in $\mathbf{p}^{E+}$ whenever $\tau^A = \tau^B$ occurred in the inner round. This procedure succeeded in conveying enough information from one outer round to the next one.

An alternative approach to this task seems to be Monte Carlo sampling of σ_I^E conditioned on the final $\mathbf{p}^{E-}$. But in our simulations this method proved to be prone to failure: Either $\mathbf{p}^E$ was effectively reset or the algorithm could not generate enough valid weight candidates at some inner round. Thus we developed and used the analytic approach instead of calculating $\mathbf{p}^{E+}$ by sampling.

N	Time	Slope a	Offset b
Even	t_s	0.495 ± 0.028	2.01 ± 0.28
Even	t_b	0.1275 ± 0.0038	2.180 ± 0.038
Odd	t_s	0.245 ± 0.076	9.02 ± 0.95
Odd	t_b	0.157 ± 0.028	4.33 ± 0.35

TABLE I. Linear regression with model $t = aN + b$ for average synchronization time t_s and break time t_b.

The attack described in this Rapid Communication is often capable of guessing the state vector $\mathbf{s}^A$ in a number of outer rounds that are less than the number of rounds that A and B needed to synchronize. This result was reproduced for many different setups of the synchronizing PPMs: varying input vector size and varying state vector size. The usual setup for cryptographic is to use even N, since PPMs with odd N synchronize notably slower or sometimes not at all [15]. However, the algorithm was also tried for odd N with an illustrative purpose and yielded satisfactory results.

As for the technicalities a sampling size of $M = 10^3$ was chosen. This implies that for $N = 2, 4$ the algorithm works similar to a brute force attack, but for large N only a small part of the weight space is sampled, e.g., for $N = 8$ only around 3% of all possible weight configurations. The absent of performance drop notwithstanding the scarce sampling highlights the efficiency of the algorithm. The mechanism to prevent the attack from getting stuck was implemented by resetting one of the bits each time that $M^2 = 10^6$ consecutive unsuccessful attempts at generating a valid weight candidate were reached. Finally, the attack was considered a success

as soon as $\mathbf{s}^{\mathrm{E}*} = \mathbf{s}^{\mathrm{A}}$ has been reached. The number of outer rounds needed to achieve this is called break time t_{b}, which varies randomly depending on the initial conditions and the course of the key exchange.

Figure 1 shows that the mean values of synchronization time $\langle t_{\mathrm{s}} \rangle$ as well as break time $\langle t_{\mathrm{b}} \rangle$ grow linearly with increasing size N of the input vectors. For all cases presented here we find that the attacker is faster than the two partners on average, $\langle t_{\mathrm{b}} \rangle < \langle t_{\mathrm{s}} \rangle$. Additionally, linear regression results shown in Table I indicate that $\langle t_{\mathrm{b}} \rangle$ grows slower than $\langle t_{\mathrm{s}} \rangle$, so that increasing N does not improve the security of the PPM-based key exchange.

Synchronization with odd N is much slower than for even N. Only runs with $t_{\mathrm{b}} < 30$ or $t_{\mathrm{s}} < 30$ were considered here to reduce computational costs. This condition also excludes failed synchronization attempts caused by reaching a stable *antiparallel* weight configuration [15], which can only happen if N is odd.

In Fig. 2 the performance of the algorithm is presented in terms of the probability of success of the attack. The functionality for many more different setups is examined here. Regarding cases with even N, the performance of the algorithm generally increases as N or G become larger. For nearly all configurations shown here the success probability P_{s} is above 80% and it actually reaches 100% in many situations. Odd N is considerably more difficult for the attacker, but nevertheless the success probability P_{s} remains larger than 50%. These values, however, have been obtained for single runs of our algorithm on each data set. As the method is nondeterministic due to Monte Carlo sampling in each inner round, repeating it on the same observations should lead to even more success.

Consequently, the results clearly show that the PPM-based neural key-exchange protocol using the parameter values K, N, and G analyzed in this Rapid Communication is not secure enough for any cryptographic application. Furthermore, there is no indication that increasing the sizes of input or state vectors would reduce the success probability and lead to a secure configuration.

In contrast, the complexity of successful attacks on TPM-based neural cryptography increases exponentially with the number $2L+1$ of possible weight values, but the effort of the partners grows only proportional to L^2 [7]. Here L has the same effect as the key size in encryption algorithms, which allows to balance speed and security. While the probabilistic attack [19] has not been tested on TPM-based neural cryptography, it is quite likely that the same scaling law for L applies to its success probability. But in order to answer this open question we are going to implement and analyze such probabilistic attacks also for TPMs.

The same question could be asked regarding the security of *chaos cryptography* [21–23], which is based on a similar synchronization principle [24]. Consequently, probabilistic attacks should be envisioned and tested there, too. Nevertheless, the specificity of the present implementation suggests that further development is needed for attacks on chaotic cryptography.

L.F.S. acknowledges the financial support of Fundación Pedro Barrié de la Maza and funding Grant No. 01GQ1001B.

[1] R. Metzler, W. Kinzel, and I. Kanter, Phys. Rev. E **62**, 2555 (2000), arXiv:cond-mat/0003051.

[2] W. Kinzel, R. Metzler, and I. Kanter, J. Phys. A: Math. Gen. **33**, L141 (2000), arXiv:cond-mat/9906058.

[3] M. Rosen-Zvi, I. Kanter, and W. Kinzel, J. Phys. A: Math. Gen. **35**, L707 (2002), arXiv:cond-mat/0202350.

[4] M. Rosen-Zvi, E. Klein, I. Kanter, and W. Kinzel, Phys. Rev. E **66**, 066135 (2002), arXiv:cond-mat/0209234.

[5] W. Kinzel and I. Kanter, J. Phys. A: Math. Gen. **36**, 11173 (2003).

[6] I. Kanter and W. Kinzel, Quantum Comput. Comput. **5**, 130 (2005).

[7] A. Ruttor, W. Kinzel, and I. Kanter, Phys. Rev. E **75**, 056104 (2007), arXiv:cond-mat/0612537.

[8] I. Kanter, W. Kinzel, and E. Kanter, Europhys. Lett. **57**, 141 (2002), arXiv:cond-mat/0202112.

[9] R. Mislovaty, E. Klein, I. Kanter, and W. Kinzel, Phys. Rev. Lett. **91**, 118701 (2003), arXiv:cond-mat/0302097.

[10] A. Ruttor, W. Kinzel, L. Shacham, and I. Kanter, Phys. Rev. E **69**, 046110 (2004), arXiv:cond-mat/0311607.

[11] A. Ruttor, W. Kinzel, and I. Kanter, J. Stat. Mech. , P01009 (2005), arXiv:cond-mat/0411374.

[12] M. Volkmer and S. Wallner, IEEE Transactions on Computers **54**, 421 (2005), arXiv:cs/0502062.

[13] S. Mühlbach and S. Wallner, J. Syst. Archit. **54**, 1065 (2008).

[14] O. M. Reyes and K.-H. Zimmermann, Phys. Rev. E **81**, 066117 (2010).

[15] O. M. Reyes, I. Kopitzke, and K.-H. Zimmermann, J. Phys. A: Math. Gen. **42**, 195002 (2009).

[16] R. Mislovaty, Y. Perchenok, I. Kanter, and W. Kinzel, Phys. Rev. E **66**, 066102 (2002), arXiv:cond-mat/0206213.

[17] L. N. Shacham, E. Klein, R. Mislovaty, I. Kanter, and W. Kinzel, Phys. Rev. E **69**, 066137 (2004), arXiv:cond-mat/0312068.

[18] A. Ruttor, W. Kinzel, R. Naeh, and I. Kanter, Phys. Rev. E **73**, 036121 (2006), arXiv:cond-mat/0512022.

[19] A. Klimov, A. Mityaguine, and A. Shamir, in *Advances in Cryptology—ASIACRYPT 2002*, edited by Y. Zheng (Springer, Heidelberg, 2003) p. 288.

[20] T. Toni, D. Welch, N. Strelkowa, A. Ipsen, and M. P. H. Stumpf, J. R. Soc. Interface **6**, 187 (2009).

[21] K. M. Cuomo and A. V. Oppenheim, Phys. Rev. Lett. **71**, 65 (1993).

[22] G. Grassi and S. Mascolo, IEEE Trans. Circ. Sys. I **49**, 1135 (1999).

[23] E. Klein, N. Gross, E. Kopelowitz, M. Rosenbluh, L. Khaykovich, W. Kinzel, and I. Kanter, Phys. Rev. E **74**, 046201 (2006), arXiv:cond-mat/0604569.

[24] L. M. Pecora and T. L. Carroll, Phys. Rev. Lett. **64**, 821 (1990).