Using a reservoir computer to learn chaotic attractors, with applications to chaos synchronization and cryptography

Piotr Antonik,[1,2,*] Marvyn Gulina,[3] Jaël Pauwels,[4,5] and Serge Massar[5]

[1]*CentraleSupélec, Campus de Metz, Université Paris Saclay, F-57070 Metz, France*
[2]*LMOPS EA 4423 Laboratory, CentraleSupélec & Université de Lorraine, F-57070 Metz, France*
[3]*Namur Institute for Complex Systems, Université de Namur, B-5000 Namur, Belgium*
[4]*Applied Physics Research Group, Vrije Universiteit Brussels, B-1050 Brussels, Belgium*
[5]*Laboratoire d'Information Quantique, Université libre de Bruxelles, B-1050 Brussels, Belgium*

(Received 7 February 2018; revised manuscript received 26 May 2018; published 24 July 2018)

Using the machine learning approach known as reservoir computing, it is possible to train one dynamical system to emulate another. We show that such trained reservoir computers reproduce the properties of the attractor of the chaotic system sufficiently well to exhibit chaos synchronization. That is, the trained reservoir computer, weakly driven by the chaotic system, will synchronize with the chaotic system. Conversely, the chaotic system, weakly driven by a trained reservoir computer, will synchronize with the reservoir computer. We illustrate this behavior on the Mackey-Glass and Lorenz systems. We then show that trained reservoir computers can be used to crack chaos based cryptography and illustrate this on a chaos cryptosystem based on the Mackey-Glass system. We conclude by discussing why reservoir computers are so good at emulating chaotic systems.

DOI: 10.1103/PhysRevE.98.012215

I. INTRODUCTION

Can one train a nonlinear dynamical system to emulate a different nonlinear chaotic dynamical system? This question has been answered positively in Refs. [1–5] in the context of the machine learning technique known as reservoir computing. But much remains to be learned concerning the quality of this emulation.

Reservoir computing (RC) [1,6–8], on which this approach is based, is a machine learning technique in which a nonlinear dynamical system with a large number of internal nodes (called the reservoir) is driven by a time dependent signal. The connections between the internal variables are chosen at random and then kept fixed (except, possibly, for a few global parameters that may be adjusted). In many implementations, the reservoir is a recurrent neural network with fixed connections. The output of the reservoir computer is a single node whose state is given by a linear combination of the states of the internal variables. The weights of this linear combination are trained to match the output as closely as possible to a desired target. Although conceptually simple, reservoir computing is powerful enough to equal other algorithms on hard tasks such as channel equalization, phoneme recognition, and others (see [9,10] for reviews).

The theory of reservoir computing is not very advanced (the situation being similar to many machine learning approaches which work well in practice but lack formal explanations for their performance). One of the most useful theoretical concepts is that of the linear and nonlinear memory capacity of reservoirs; see [11–14]. It was also shown recently that a variant of the reservoir computer is universal in the category of fading memory filters [15].

Reservoir computing has also been implemented experimentally with performance comparable to digital implementations [16–18] with photonic implementations presenting particularly high speeds [19–21], see [22] for a review.

In the case where one wants the reservoir computer to emulate a dynamical system, the reservoir is first driven by the state of the dynamical system, and trained to predict this state one time step in the future. After training one closes the loop and feeds the output of the reservoir back into itself, whereupon it will develop autonomous dynamics that are—one hopes—close to those of the original dynamical system.

This approach was originally introduced to forecast the trajectories of chaotic dynamical systems, where it reached record forecasting horizons [1]. These results were improved recently in Ref. [15]. In addition this method was used numerically in Ref. [3] to infer the values of hidden degrees of freedom of the dynamical system, in Ref. [4] to estimate its Lyapunov exponents, and in Ref. [5] to predict spatiotemporal chaos. It was also implemented in an optoelectronic system [2] where it was shown that the experimental reservoir could be trained to have similar dynamics to the original system (similar spectrum, Lyapunov exponents, etc.). From these works it is clear that a reservoir computer trained as described above can emulate another, *a priori* completely different (possibly chaotic) dynamical system. However much remains to be understood about the quality and accuracy of the method, as well as its potential limitations. Here we show how trained reservoir computers can be used to replace dynamical systems in two other applications: chaos synchronization and cracking chaos-based cryptography.

One of the most surprising aspects of chaos theory is the synchronization of two identical chaotic systems. Although

*piotr.antonik@centralesupelec.fr

the dynamics of each system separately is unpredictable, if one system is driven by the other, the two systems will synchronize [23,24]. This phenomenon has been extensively studied; see, e.g., the review [25].

In the first part of this work, we show that a reservoir computer, trained to emulate a chaotic system as described above, if driven by the original system, will synchronize with it. This demonstrates that not only is the dynamics of the reservoir computer emulator superficially similar to that of the original system, but that its chaotic attractor has similar properties. That is, it captures a large part of the characteristics of the dynamics of the original system. We illustrate this in two examples: the Lorenz [26] and Mackey-Glass systems [27].

After the discovery of chaos synchronization, considerable effort was devoted to trying to use this effect and the unpredictability of chaotic systems to hide secret messages; see, e.g., [28–30]. A series of experimental demonstrations were realized [31–33]. However, it was later shown that chaos-based cryptography is fundamentally insecure, as there are efficient ways for an eavesdropper to find the parameters of the chaotic system (which play the role of secret key), at least using plain text attack. This was first demonstrated on a series of examples, see, e.g., [34–40], and then in full generality [41].

As an application of our results on chaos synchronization we consider using the reservoir computer to crack chaos-based cryptography. We will study a cryptosystem based on the Mack-Glass equations previously studied from the point of view of encryption and cryptanalysis in Refs. [34,40]. This is an "open loop" configuration of the type widely used in experiments because of its robustness. Our attack based on reservoir computing performs similarly to the parameter search studied in Ref. [34].

We conclude this article with a general discussion of why reservoir computers are able to crack chaos-based cryptosystems. This should be viewed as a general discussion which shows the plausibility of this kind of attack, but without any claim to mathematical rigor or an understanding of the efficiency of such attacks.

II. RESERVOIR COMPUTING

A. Basic principles

The reservoir computer used in this work is a discrete-time echo state network, as introduced in Refs. [1,6]. The reservoir states vector x, consisting of N neurons, is updated following the equation

$$\begin{aligned} x(n) &= (1 - Ca)x(n-1) \\ &\quad + C \tanh\,[w_{\text{in}}u(n) + Wx(n-1) + w_{\text{back}}d(n-1)], \end{aligned} \tag{1}$$

where $n \in \mathbb{Z}$ is the discrete time, C is a time-scale constant, a is the leak rate, W is a $N \times N$ matrix of internal connection weights, w_{back} is the N-size weight vector for feedback connections from the output to the reservoir, w_{in} is a N-size vector, and u is a constant. Together, C and a realize a low-pass filter with adjustable properties.

The elements of w_{in}, W, and w_{back} are chosen from a uniform distribution over the interval $[-1,+1]$. A reservoir computer must be not too far from the edge of chaos to exhibit good performance. To this end the matrix W is then rescaled to adapt its spectral radius. The vectors w_{in} and w_{back} are possibly also rescaled to adapt the strength of the input and feedback. Throughout this work the input bias is fixed to $u = 0.2$. To fix the other parameters of the reservoir computer, such as C, a, spectral radius of W, and scaling of w_{in} and w_{back}, we took inspiration from [6]. When the parameter values were not available for a specific task, suitable values were chosen heuristically, but without carrying out a systematic search over all possible values. The chosen values of parameters are given in the text below.

The output equation of a single-output network is given by a dot product

$$y(n) = w_{\text{out}}[x(n), u(n)], \tag{2}$$

where $[x(n), u(n)]$ is the concatenation of the reservoir states vector $x(n)$ with the input $u(n)$, and w_{out} are $N+1$ output weights (also known as the output mask).

During training we adjust the weights w_{out} so that the output $y(n)$ is as close as possible to the desired output $\tilde{y}(n)$. To this end we minimize the normalized mean square error (NMSE), given by

$$\text{NMSE} = \frac{\langle (y(n) - \tilde{y}(n))^2 \rangle}{\langle (\tilde{y}(n) - \langle \tilde{y}(n) \rangle)^2 \rangle}. \tag{3}$$

The NMSE indicates how far the time series $y(n)$ generated by the reservoir deviates from the target time series $\tilde{y}(n)$. The resulting value is straightforward to interpret: NMSE $= 0$ means that the two series match, while NMSE $= 1$ indicates no similarity at all. Minimizing the NMSE with respect to the readout weights gives rise to a system of linear equations that is readily solved. We do not use ridge regression [42] (except in the Lorenz task; see below) as there is enough training data to avoid overfitting.

After training we evaluate the performance of the reservoir on a new data set. When we report NMSE values, it is the values evaluated on the test sequence.

In the present work, the reservoir computer must predict or process continuous time signals $u(t)$. To pass from continuous to discrete time, we sample the continuous time input as

$$u(n) = u(t + n\Delta), \tag{4}$$

where Δ is the sampling interval. For the different applications, we quote the sampling interval used.

B. Training to crack chaotic cryptography

When training to crack chaotic cryptography, we operate the reservoir as described above. We set $w_{\text{back}} = 0$. We consider a plain text attack in which Eve has access to the signal sent by Alice to Bob, and to the message that we encrypted by Alice. For training we take $u(n)$ to be the signal intercepted by Eve. We take $\tilde{y}(n) = m(n)$ to be the message encoded by Alice. During the training phase the weights w_{out} are thus adjusted so that the reservoir outputs the encrypted message. Eve's reservoir computer is then ready to decrypt new encrypted messages.

C. Training to emulate chaotic systems

When training the reservoir to emulate a chaotic system, for instance, for chaos synchronization, we proceed as follows. Denote by $s(n)$ the time series of the chaotic system we wish to emulate.

We set the input to a constant

$$u(n) = 0.2. \tag{5}$$

During training Eqs. (1)–(3) and (5) are supplemented by

$$\begin{aligned} d(n) &= s(n) \quad \text{(during training)},\\ \tilde{y}(n) &= s(n) \quad \text{(during training)}. \end{aligned} \tag{6}$$

That is, the training phase is used to optimize the readout weights w_{out} so that the reservoir predicts the next point $s(n)$ in the input chaotic time series, given the previous points $s(n-1), s(n-2), \ldots$.

After the training, the readout weights w_{out} are fixed and the teacher signal $d(n)$ is replaced by the output signal $y(n)$, so that the reservoir becomes autonomous. The evolution of the reservoir computer during the autonomous run is given by Eqs. (1), (2), and (5) supplemented by

$$d(n) = y(n) \quad \text{(during autonomous run)}. \tag{7}$$

The reservoir now uses its estimates of the previous points in the time series to estimate the next point.

III. TRAINING ON THE MACKEY-GLASS AND LORENZ SYSTEMS

For illustrative purposes in this work, we use the one-dimensional Mackey-Glass (MG) delay equation and the tridimensional Lorenz system. The prediction of the MG and Lorenz systems time series using echo state networks and variants thereof has been investigated previously in a number of works; see [1–5,43,44].

The Mackey-Glass delay differential equation

$$\frac{dx}{dt} = \beta \frac{x(t-\tau)}{1 + x^n(t-\tau)} - \gamma x \tag{8}$$

with $\tau, \gamma, \beta, n > 0$ was introduced to illustrate the appearance of complex dynamics in physiological control systems [27]. To obtain chaotic dynamics, we set the parameters as in Ref. [1]: $\beta = 0.2$, $\gamma = 0.1$, $\tau = 17$, and $n = 10$. With these settings, the highest Lyapunov exponent is $\lambda = 0.006$ [1].

Equation (8) was integrated using MATLAB's `dde23` solver with the initial condition $x(t \leqslant 0) = 0.5$ and integration step of 0.5 for 7000 time steps. The first 1000 transient values were discarded and the remaining data were split into 3000 training and 3000 test inputs.

For the MG task, we used a reservoir with $N = 1500$ neurons, the matrix W was rescaled to a spectral radius of 0.79, while the vectors w_{in}, w_{back} were not rescaled, and we set $\Delta = 1$, $C = 0.44$, $a = 0.9$.

At the training stage we obtained an error of NMSE $= 3 \times 10^{-9}$. During the free run, the error gradually increases, as the reservoir output signal slowly deviates from the target trajectory on the Mackey-Glass attractor. Nevertheless, the system manages to generate the desired output for several hundreds of time steps with reasonable precision.

The Lorenz equations, a system of three ordinary differential equations

$$\frac{dx}{dt} = \sigma(y - x), \tag{9a}$$

$$\frac{dy}{dt} = -xz + rx - y, \tag{9b}$$

$$\frac{dz}{dt} = xy - bz, \tag{9c}$$

with $\sigma, r, b > 0$, were introduced as a simple model for atmospheric convection [26]. The system exhibits chaotic behavior for $\sigma = 10$, $b = 8/3$, and $r = 28$ [45], that we used in this study. This yields a chaotic attractor with the highest Lyapunov exponent of $\lambda = 0.906$ [1].

The Lorenz equations (9) were integrated using MATLAB's `ode45` routine with an integration step of 0.02 for 10 000 time steps. We only used the x coordinate of the chaotic system, which was rescaled by a factor of 0.01, as in Ref. [1]. The first 1000 transient values were discarded and the remaining data were split into 6000 training and 3000 test inputs.

For the Lorenz task, we used a reservoir of size $N = 1500$. We set the spectral radius of the weight matrix W to 0.97, the input and feedback weights w_{in} and w_{back} were rescaled to the interval $[-0.5, 0.5]$, and we set $\Delta = 1$, $C = 0.44$, and $a = 0.9$. To obtain better results we used a form of ridge regression for this task, namely we added noise drawn from the uniform distribution over $[-10^{-6}, 10^{-6}]$ in the argument of the tanh in Eq. (1).

We obtained a training error of NMSE $= 3 \times 10^{-8}$. The error is one order of magnitude higher here than in the Mackey-Glass case. This may be due to the fact that the Lorenz system has a higher positive Lyapunov exponent and thus exhibits stronger chaoticity than the Mackey-Glass system, and/or to the fact that the reservoir computer is expected to emulate the dynamics of a three-dimensional system given only one dimension (the x coordinate), which is more challenging than the reconstruction of the scalar Mackey-Glass system.

Note: We use here the traditional notation for echo state networks, MG and Lorenz systems, which means that different meanings are given to the same letters. When it is not clear from the context, we use a subscript $x_{\text{RC}}, x_{\text{MG}}, x_{\text{L}}$ to differentiate them.

IV. SYNCHRONIZING TRAINED RESERVOIR COMPUTERS WITH THE MACKEY-GLASS AND LORENZ SYSTEMS

Let $s(n)$ be the time series of the chaotic system with which one wishes to synchronize, with n the discretized time used to integrate Eqs. (8) and (9). We first train the reservoir to predict the next sample in the time series, as described above. Next we start an autonomous run in which the reservoir follows its own dynamics, given by Eqs. (1), (2), and (7). At time $n = n_0$, we start weakly driving the reservoir with the chaotic time series $s(n)$. That is, its dynamics is given by Eqs. (1) and (2), supplemented by

$$d(n) = (1-q)y(n) + qs(n) \quad \text{(when locked)} \tag{10}$$

with $0 \leqslant q \leqslant 1$.

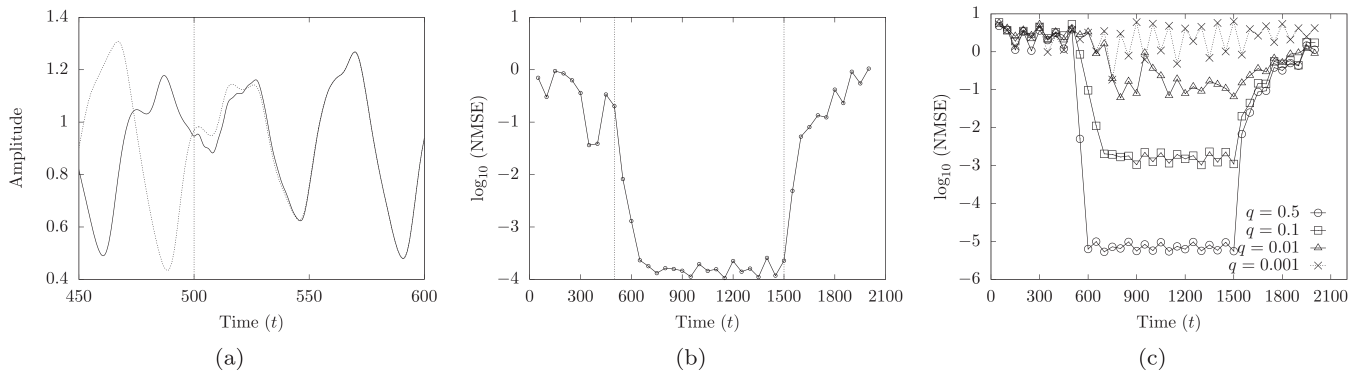

FIG. 1. Synchronization of a trained reservoir computer on the Mackey-Glass system, integrated from the initial condition $x_{\mathrm{MG}}(t \leqslant 0) = 0.2$. The reservoir computer is trained from $t = -3000$ to $t = 0$, whereupon it become autonomous until $t = 500$. At $t = 500$, we set the coupling to $q = 0.25$ and remove it ($q = 0$) at $t = 1500$. Plot (a) depicts only the region of interest around $t = 500$, where the reservoir (solid line) synchronizes with the chaotic system (dotted line). Plot (b) shows the evolution of the NMSE, averaged over 100-time-step intervals, for the entire duration of the simulation, showing the decrease of the NMSE when the synchronization is turned on, the saturation of the NMSE to a low value (NMSE $= 1.5 \times 10^{-4}$) after synchronization, and the increase of the NMSE when the synchronization is turned back off. Plot (c) illustrates the same scenario with different coupling strengths. For $q = 0.5$, the synchronization is quicker, as can be seen from the steeper slope, and the resulting NMSE is lower (NMSE $= 7.1 \times 10^{-6}$). Lower values of q lead to slower synchronization, with a higher error. At $q = 0.001$, the systems no longer synchronize.

Figures 1 and 2 illustrate how the trained reservoir can lock onto the MG and Lorenz systems. It should be noted that during the synchronization phase, the NMSE decreases until a minimum value and then stays constant. On the other hand, if we were to synchronize two identical MG or Lorenz systems, the NMSE would decrease until it reached the machine precision. The difference arises because the trained reservoir does not exactly reproduce the dynamics of the MG or Lorenz system.

Taking $q = 0.25$, we obtained synchronization errors of $\mathrm{NMSE}_{\text{MG-RC}} = 1.5 \times 10^{-4}$ and $\mathrm{NMSE}_{\text{LZ-RC}} = 2.3 \times 10^{-7}$. Note that the first subscript corresponds to the primary system, and the seconds indicates the secondary system that is being driven by the primary. Results for other values of q are given in the last panels of Figs. 1 and 2.

We also tested the inverse scenario, in which a chaotic system (MG or Lorenz) is synchronized on a trained reservoir computer. Let y_{RC} be the output of the trained reservoir. In the case of MG, we let Eq. (8) evolve autonomously until $t = 500$. At this time, we change the right hand side of Eq. (8), replacing $x(t)$ by

$$x(t) \to q y_{\mathrm{RC}}(t) + (1 - q)x(t). \tag{11}$$

In the case of Lorenz, we let Eqs. (9) evolve autonomously until $t = 20$, when we change the right hand side of Eqs. (9), replacing $x(t)$ by Eq. (11). In both cases we took $q = 0.25$. The results are plotted in Fig. 3. We obtained synchronization errors of $\mathrm{NMSE}_{\text{RC-MG}} = 1.5 \times 10^{-3}$ and $\mathrm{NMSE}_{\text{RC-LZ}} = 1.5 \times 10^{-1}$. Note that the NMSEs are higher than when the RC synchronizes onto the MG or LZ systems. This may be due

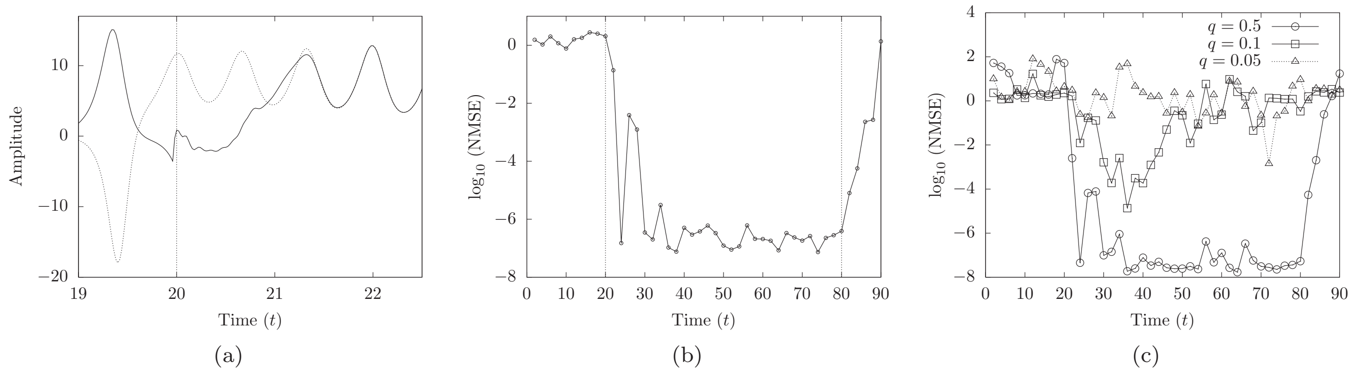

FIG. 2. Synchronization of a trained reservoir computer on the Lorenz system, integrated from $(x_0, y_0, z_0)_{\mathrm{LZ}} = (10,0,0)$. The reservoir computer is trained from $t = -3000$ to $t = 0$, whereupon it becomes autonomous until $t = 20$. At $t = 20$, we set the coupling to $q = 0.25$ and remove it ($q = 0$) at $t = 80$. Plot (a) depicts only the region of interest around $t = 20$, where the reservoir (solid line) synchronizes with the chaotic system (dotted line). Plot (b) shows the evolution of the NMSE, averaged over 100-time-step intervals, for the entire duration of the simulation, showing the decrease of the NMSE when the synchronization is turned on, the saturation of the NMSE to a low value (NMSE $= 2.3 \times 10^{-7}$) after synchronization, and the increase of the NMSE when the synchronization is turned back off. Plot (c) illustrates the same scenario with different coupling strengths. Again, higher coupling ($q = 0.5$) leads to a lower synchronization error (NMSE $= 5.1 \times 10^{-8}$). Decreasing q leads to a system that tries to synchronize but fails ($q = 0.1$). At $q = 0.05$, the systems do not synchronize at all.

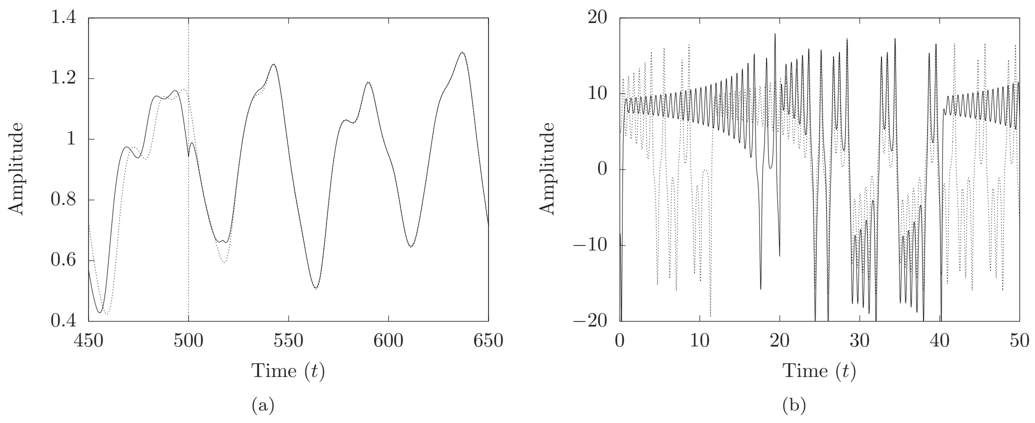

FIG. 3. Inverse scenario: synchronization of (a) Mackey-Glass and (b) Lorenz chaotic systems on a trained reservoir computer. The left panel depicts the region of interest around $t = 500$, where the Mackey-Glass system (solid lines) synchronizes onto the reservoir computer (dotted lines). The synchronization is quite efficient in this case. For the Lorenz system, a close view of the region of interest around $t = 20$ does not show any interesting dynamics, since the synchronization error stays quite significant. For this reason, we plot the full time trace instead (right panel) that shows that the synchronized Lorenz system (solid line) accurately follows the switches between the lobes of the Lorenz attractor, emulated by the reservoir computer (dotted lines). The resulting error is much higher in this experiment, but the synchronization phenomenon can still be observed. At $t = 40$ we stopped the synchronization ($q = 0$) and the two systems immediately desynchronized.

to the fact that the reservoir produces outputs at discrete times separated by the sampling rate Δ, and that this induces a form of noise on the driving signal. We did not investigate in detail the origin of this difference.

V. CHAOS-BASED CRYPTOGRAPHY

Chaos-based cryptography is based on the two ideas that (1) the unpredictable nature of chaotic systems can be used to mask a message and (2) that chaos synchronization can be used by the receiver to faithfully recover the message. Unfortunately this nice idea has not survived systematic cryptanalysis, basically because the key space (i.e., the parameters describing the chaotic system) is too small, and efficient search methods to recover the key can be developed; see [41] and references therein.

The fact that reservoir computers can be trained to emulate chaotic systems to the extent that the trained reservoir will synchronize with the original chaotic system (as demonstrated in the previous section) suggests that reservoir computing could form the basis for an alternative, conceptually different, approach to cracking chaos based cryptography.

Here, we demonstrate this on a specific example of how a reservoir computer can be used to crack a chaos based cryptosystem. We then give some heuristic arguments on why reservoir based approaches could systematically crack chaos based cryptography. For definiteness, we focus on a scheme previously studied in Ref. [40] and referred to as the III/1 scheme. This scheme is of interest because it is similar to many of the systems used in experimental chaos based cryptography that often use delay dynamical systems as chaotic systems; see, e.g., [32,33]. It has already been cracked in Ref. [40], using time-delay system reconstruction method to recover the unknown parameters of the transmitter. In this work, we use a reservoir computer as an alternative approach. Its advantage over the method in Ref. [40] is that the knowledge of the governing equation of the transmitter [see Eq. (12) below] is not required.

We recall the scheme III/1 of [40], wherein Alice and Bob exchange secret messages, while Eve is eavesdropping. To encode her message, Alice uses a delay dynamical system in which she injects her message $m(t)$. Her dynamical system obeys the equation

$$\epsilon \dot{x}(t) = -x(t) + f[x(t-\tau)] + m(t), \tag{12}$$

where τ is the delay and ϵ characterizes the inertial properties of the system. Alice sends $x(t)$ to Bob.

We suppose that $x(t)$ is subject to noise during transmission, so that what is received by Bob is

$$x'(t) = x(t) + \nu(t), \tag{13}$$

where $\nu(t)$ is white noise whose amplitude is given below [i.e., for each successive time point t, $\nu(t)$ is independently drawn from the uniform distribution over $[-\nu,\nu]$].

To decrypt the message, Bob uses the same delay system, but in an open loop configuration to obtain the variable $y(t)$ given by

$$\epsilon \dot{y}(t) = -y(t) + f[x'(t-\tau)]. \tag{14}$$

Then, Bob computes

$$z'(t) = x'(t) - y(t) \tag{15}$$

and obtains an approximate message $m'(t)$ as follows:

$$m'(t) = \epsilon \dot{z}'(t) + z'(t). \tag{16}$$

This allows Bob to recover the message $m(t)$, typically corrupted by some high frequency noise. Passing $m'(t)$ through a passband filter centered on the frequency band occupied by $m(t)$ allows Bob to recover a good approximation of $m(t)$.

In order to crack this system, we suppose that Eve has access to a plain text attack, i.e., she has access to both $x'(t)$ and $m(t)$ during some time interval. Thus, she can train her reservoir

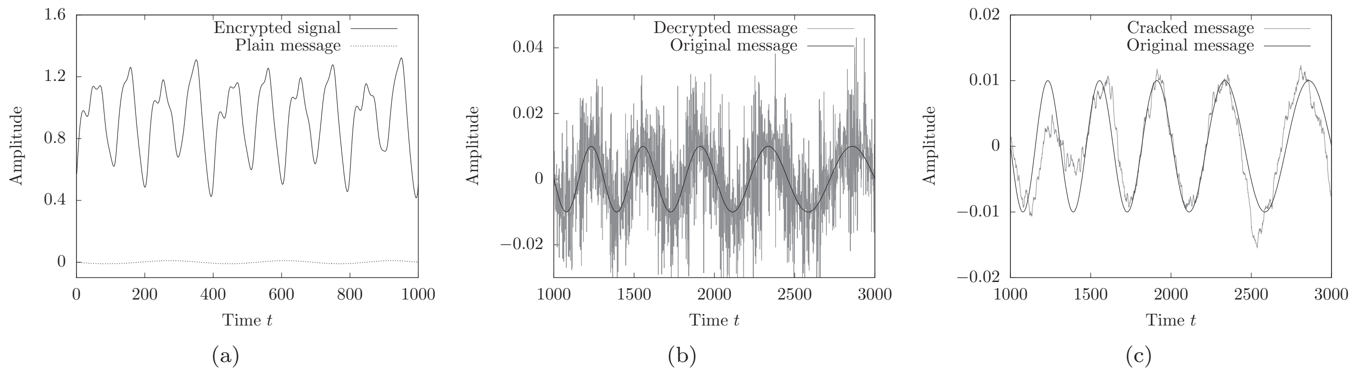

FIG. 4. Temporal signals obtained during encryption and decryption of the frequency-modulated sine message Eq. (17). (a) Encrypted signal $x(t)$ (chaotic carrier + low-amplitude message, solid line) sent by Alice to Bob. The message $m(t)$ (dotted line) is plotted for scale. Note that a small amount of white noise is added to the message during transmission. (b) Message decrypted by Bob (grey) compared to the original message (black). Despite the high-frequency noise, Bob accurately recovers the frequency modulation of the sine wave. (c) Message obtained by Eve (black) using a trained reservoir computer. Despite slight amplitude variations of the recovered message, Eve accurately recovers the frequency modulation (grey), and with less high-frequency noise than Bob.

computer to produce $m(t)$ given $x'(t)$ as input. To this end she uses the scheme described in Sec. II B, in which the input to the reservoir $u(t)$ is taken to be the signal $x'(t)$ sent by Alice to Bob, and the output of the reservoir [$y(t)$ in Eq. (3)] is trained to be as close as possible to $m(t)$.

To illustrate this, we used the MG system Eq. (8) with the same parameters used elsewhere in this work ($\beta = 0.2$, $\gamma = 0.1$, $\tau = 17$, and $n = 10$) (which are identical to the parameters used in [40] except for τ).

We first investigate the case where the message is a frequency-modulated harmonic signal of the form

$$m(t) = A\sin[2\pi f_c t - B\cos(2\pi f_m t)], \qquad (17)$$

where $f_c = 5\times10^{-3}$ is the central frequency of the power spectrum of the signal, $B = 3$ is the frequency modulation index, $f_m = 5\times10^{-5}$ is the modulation frequency, and $A = 0.01$ is the amplitude of the message, chosen to ensure that the information signal comprises 1% of the amplitude of the chaotic carrier. The message and the values of the parameters are identical to those used in Ref. [40]. We take the amplitude of the noise $\nu(t)$ to be $\nu = 10^{-1}A$ where A is the maximum amplitude of the message $m(t)$, corresponding to a signal-to-noise (SN) ratio of 1.5×10^5.

To crack the system, Eve used a reservoir computer with $N = 250$ internal nodes and trained on a plain text message comprising 12 000 time steps. The spectral radius of the weight matrix W was set to 0.79, the input weights w_{in} were rescaled with a global coefficient of 0.9, the feedback was switched off $w_{\text{back}} = 0$, and we set $\Delta = 0.5$, $C = 0.05$, and $a = 0.9$. We obtained a training error of NMSE $= 3.8\times10^{-2}$.

Using this trained RC, Eve can now try to recover an unknown message sent by Alice. The results are presented in Fig. 4 (temporal signals) and Fig. 5 (frequency spectra), where we compare decryption by Bob and Eve. Bob manages to accurately retrieve the frequency modulation, but his result is corrupted by high-frequency noise. Therefore, his receiver would benefit from a low-pass filter to get rid of this noise. Eve's reservoir, on the other hand, incorporates a band-pass filter [the coefficients a and C in Eq. (1)] that was adjusted to match the frequency band of the message sent by Alice. As a consequence Eve obtains a much cleaner signal. Note that there remain some amplitude variations of Eve's output signal, which may be due to the passband ripples of the low-pass filter of the reservoir computer. However, these ripples do not hinder the retrieval of the frequency modulation. Note that if we set the noise during communication ν to zero, then Bob's message is of higher quality than Eve's (figures not shown), which is expected since in this case Bob is carrying out exactly the inverse operation as Alice.

We next investigate a more realistic scenario in which frequency modulation of the sine wave is used to transmit a stream of bits $b(k)$ (with $k \in \mathbb{Z}$) by assigning a higher frequency ω_1 for a "1" and a lower ω_0 for a "0." In this case, the expression of the encoded message [Eq. (17)] becomes

$$m(t) = A\sin[\omega_{b(k)}t], \quad t \in [kT, k(T+1)[, \qquad (18)$$

where $T = 2\pi/\omega_0$ is the duration of one bit. We take $A = 0.02$, $\omega_0 = 0.02\pi$, and $\omega_1 = 0.04\pi$. These frequencies are chosen so that the message spectrum is centered on the frequencies where the Mackey-Glass system has largest spectral amplitude, making the system, in principle, harder to crack than the previous example. The amplitude of the noise during transmission is taken to be $\nu = 10^{-1}A$, corresponding to a SN ratio of 3.8×10^4.

Eve uses a reservoir computer with $N = 250$ internal nodes and trained on a plain text message comprising 7000 time steps. The spectral radius of the weight matrix W was set to 0.79, the input weights w_{in} were not rescaled, the feedback was switched off $w_{\text{back}} = 0$, and we set $\Delta = 0.1$, $C = 0.22$, and $a = 0.9$. We obtained a training error of NMSE $= 2\times10^{-1}$.

Figure 6 displays the original message (black) and the signal obtained by Eve, using a reservoir computer (grey). Although the recovered signal is not perfect, with significant distortion and noise, one can still accurately recover the encrypted bit message.

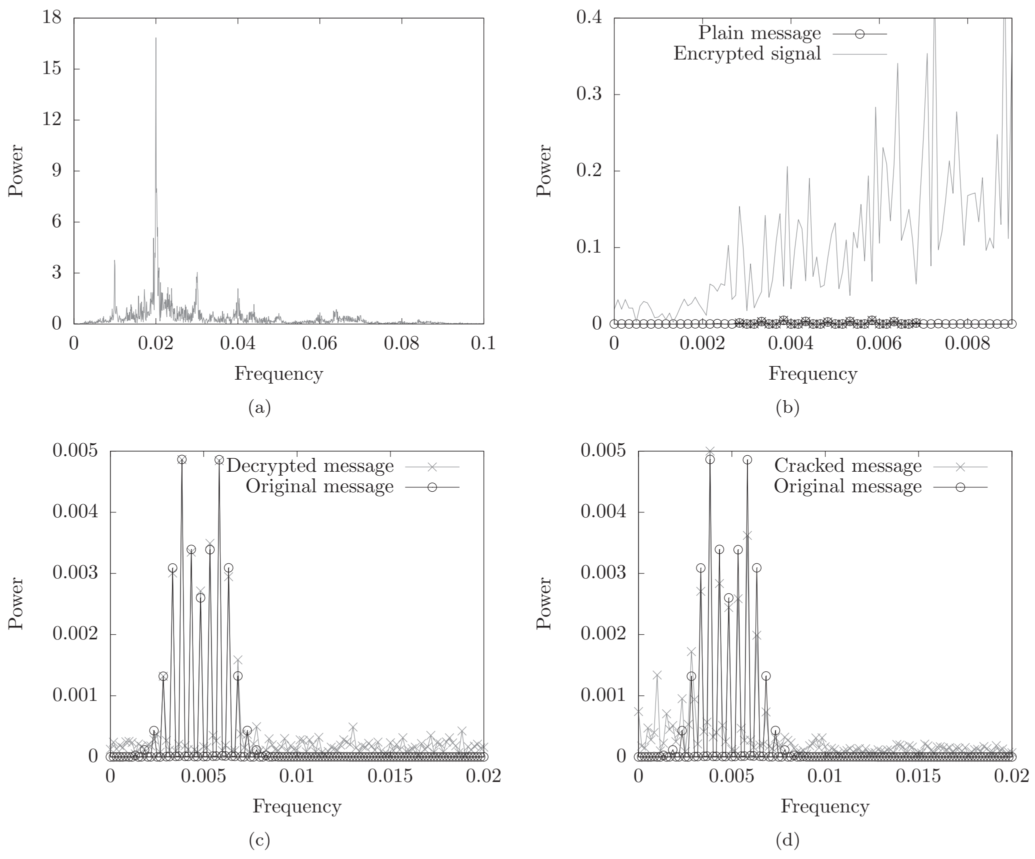

FIG. 5. Frequency spectra of signals obtained during encryption and decryption of the frequency-modulated sine message Eq. (17). (a) Mackey-Glass signal $x(t)$. (b) Zoom on the part of the spectrum of the Mackey-Glass signal $x(t)$ which overlaps with the spectrum of the message [the frequency-modulated sine Eq. (17), highlighted for clarity]. Note that the message is well hidden by the chaotic signal $x(t)$. (c) Message decrypted by Bob: the frequency modulation is recovered accurately, while the right hand side of the spectrum (flat, nonzero) corresponds to the high-frequency noise, observed in Fig. 4(b). (d) Message cracked by Eve: the frequency modulation has been recovered accurately, with a lower level of high-frequency noise.

VI. DISCUSSION: WHY CAN RESERVOIR COMPUTERS EMULATE CHAOTIC DYNAMICAL SYSTEMS AND CRACK CHAOS BASED CRYPTOGRAPHY?

The works [1–5] and the present results show that reservoir computers can be trained to emulate chaotic dynamical systems. Here we try to sketch why this is possible.

The key theoretical concept to understand reservoir computers seems to be the notion of fading memory function [46]; see [12–15]. Consider a time series $u(n)$ with n integer. We denote by $u^{-\infty}(n) = (u(n), u(n-1), u(n-2), \ldots)$ the set of all values up to and including time n. Consider a real valued function acting on this left infinite time series $F[u^{-\infty}(n)]$. It

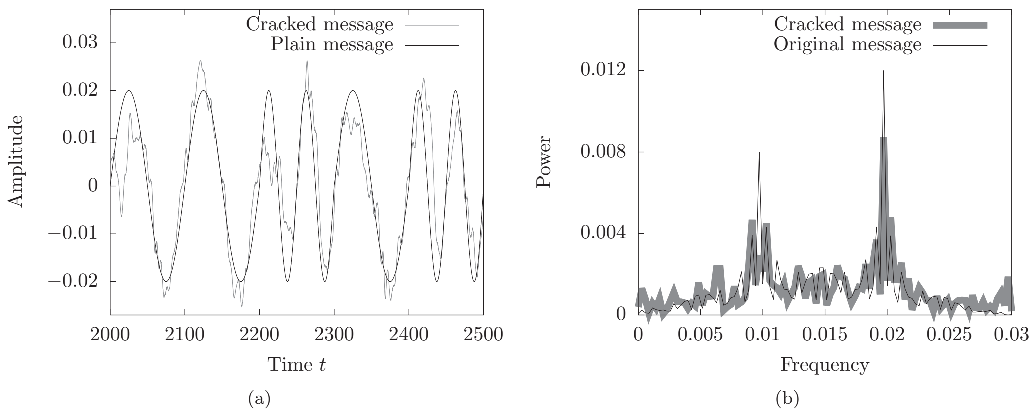

FIG. 6. Decryption of a stream of bits encoded using frequency modulation [Eq. (18)], plotted in (a) time and (b) frequency domains. The sine wave was modulated with frequencies 0.02π and 0.04π for bits "0" and "1," respectively, with a duration of one and two periods, respectively. The cracked signal is not perfectly recovered, but the frequency-coded message can be easily retrieved.

has the fading memory property if F depends less and less on $u(n)$ as $n \to -\infty$. More precisely, if there is a family of functions $F_m(u(n),u(n-1),\dots,u(n-m))$ such that $F_m \to F$ as $m \to \infty$, i.e., we can approximate F by functions of only the last m terms in the time series.

If a reservoir computer obeying the echo state property is driven by a time series $u(n)$, then its internal variables $x_i(n)$ are fading memory functions of the input time series $x_i(n) = X_i[u^{-\infty}(n)]$, and hence the output $y(n)$ of the reservoir computer is also a fading memory function.

In Ref. [15] it was shown that there exists a variant of reservoir computer (based on a polynomial recurrence) that has the following universality property: as the size of the reservoir grows, it can approximate any fading memory function to arbitrary accuracy. This universality property may also hold for reservoirs of the form of Eq. (1), although this has not been proven. Assuming this hypothesis to be true, given a fading memory function $F[u^{-\infty}(n)]$, a reservoir computer can approximate F to arbitrary accuracy. That is, the reservoir implements a function $y(n) = F'[u^{-\infty}(n)]$ which is arbitrarily close to F, $|F' - F| \leqslant \epsilon$ (for an appropriate metric on the space of fading memory functions), where ϵ can be made arbitrarily small by taking the number of variables of the reservoir N sufficiently large.

Chaotic dynamical systems are also closely linked to the fading memory property. Indeed many chaotic dynamical systems can be expressed as a recurrence of the form

$$d(n) = D[d^{-\infty}(n-1)], \tag{19}$$

where D has the fading memory property. This is obviously the case for the logistic map, or for the Mackey-Glass system (upon discretizing the time variable), and probably is also true for the Lorenz system (although we have not proven it).

Therefore, if a reservoir computer is driven by such a chaotic time series $d(n)$, it can learn an arbitrarily good approximation D' of the chaotic recurrence. Upon closing the reservoir upon itself, it will obey the recurrence

$$d(n) = D'[d^{-\infty}(n-1)], \tag{20}$$

with $|D' - D| \leqslant \delta$, where δ can be made arbitrarily small by taking the number of variables of the reservoir N sufficiently large. One can thus expect that many properties of the original dynamical system will be inherited by the reservoir's emulation. This is confirmed by the numerical studies carried out so far. The above suggests that this could be extended to a formal proof.

We now turn to chaos cryptography. In such a system, Alice sends Bob a times series $s(n)$ in which her message $m(n)$ is hidden. In most such cryptography systems, if not all, Bob's decoding operation will consist of passing the time series $s(n)$ through a fading memory function M' to obtain an approximation $m'(n)$ of the original message $m'(n) = M'[s^{-\infty}(n)]$ where $|m' - m| < \eta$ for some metric on time series. The fading memory nature of the decoder seems necessary, as it implies that the decoder can act locally on the time series, and does not depend on the values of the time series arbitrarily far in the past. Furthermore, in experimental implementations, the decoding operation must be robust to imperfections. That is, if M'' is another fading memory function that is sufficiently close to M' (i.e., $|M'' - M'| < \rho$), then the corresponding decoded message $m''(n) = M''[s^{-\infty}(n)]$ will also be close to the original message if ρ is sufficiently small.

But this means that given a plain text attack in which the eavesdropper knows the time series $s(n)$ and the corresponding message $m(n)$, she can train a reservoir computer to approximate the fading memory function M'. Given the above-mentioned necessary robustness of the cryptographic scheme, if Eve's approximation is good enough, her trained reservoir will now be able to recover the unknown messages.

The above arguments show the plausibility of reservoir computers being able to emulate chaotic systems and to crack chaos-based cryptography. These arguments however say nothing about the efficiency of this approach. Numerical investigations so far suggest that reservoir computers are remarkably good at such emulation tasks. Presumably this is because the reservoir computing approach generates fading memory functions x_i which are in some sense close to the kind of functions produced by natural systems. But of course any other approach that can generate a dense set of fading memory functions (such as, for instance, Volterra series) will also work, although possibly less efficiently.

VII. CONCLUSION

Time series forecasting has been investigated with several different machine learning techniques, in addition to reservoir computing. These include support vector machines [47,48], and autoregressive models and neural networks [49]. It would of course be very interesting to compare reservoir computing with other machine learning approaches for the above tasks of emulating chaotic systems, learning their parameters, chaos synchronization, cracking chaos cryptography, etc. Such a comparison goes however beyond the present work. We expect that reservoir computing will probably report favorably in such a comparison. Indeed as noted above to the best of our knowledge reservoir computers hold the record for predicting the future trajectory of chaotic systems. (Most likely this is because reservoir computers, being recurrent dynamical systems themselves, already encode much of the structure which needed for such a task.) An advantage of reservoir computers is that they are particularly easy to train, using only a linear regression.

The present work builds on previous works which showed that reservoir computers with output feedback can emulate chaotic dynamical systems. Previous works focused on forecasting trajectories [1,15] and predicting spatiotemporal chaos [5], inferring hidden degrees of freedom [3], estimating Lyapunov exponents [4]. Here we show that trained reservoir computers can synchronize with another chaotic system, thereby demonstrating that the trained reservoir computer has an attractor with similar geometry and stability properties as the original system. We then show how a reservoir computer can be used to crack chaos based cryptography.

It is interesting to note that cracking chaos based cryptography seems comparatively easy for the reservoir computer. Indeed, while for the time series prediction task (Sec. III) we used reservoirs with $N = 1500$ neurons, for the cryptography application we only used $N = 250$ neurons. In addition, in the presence of noise in the transmission line the reservoir computer in fact performed better than the system used by Bob. This is in part because the reservoir computer we used comprises a low-pass filter.

As noted above, reservoir computers can be implemented in hardware implementations, with good performance and high speed [19–21]. The present numerical results suggest that such experimental systems would be good candidates for cracking physically implemented chaos cryptography.

ACKNOWLEDGMENTS

We thank M. Haelterman, G. Verschaffelt, G. Van der Sande, and D. Rontani for helpful discussions. This work was supported by the Interuniversity Attraction Poles Program (Belgian Science Policy) Project Photonics@be IAP P7-35, by the Fonds de la Recherche Scientifique (F.R.S.-FNRS), by the Action de Recherche Concertée of the Fédération Universitaire Wallonie-Bruxelles through Grant No. AUWB-2012-12/17-ULB9, by the Research Foundation–Flanders (FWO, Ph.D. fellowship), and by the Université de Namur.

P.A. and M.G. contributed equally to this work.

[1] H. Jaeger and H. Haas, Science **304**, 78 (2004).
[2] P. Antonik, M. Haelterman, and S. Massar, Phys. Rev. Appl. **7**, 054014 (2017).
[3] Z. Lu, J. Pathak, B. Hunt, M. Girvan, R. Brockett, and E. Ott, Chaos **27**, 041102 (2017).
[4] J. Pathak, Z. Lu, B. R. Hunt, M. Girvan, and E. Ott, Chaos **27**, 121102 (2017).
[5] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott, Phys. Rev. Lett. **120**, 024102 (2018).
[6] H. Jaeger, The "echo state" approach to analysing and training recurrent neural networks. GMD Report 148, German National Research Center for Information Technology, 2001.
[7] W. Maass, T. Natschläger, and H. Markram, Neural Comput. **14**, 2531 (2002).
[8] D. Verstraeten, B. Schrauwen, M. d'Haene, and D. Stroobandt, Neural Networks **20**, 391 (2007).
[9] M. Lukoševičius and H. Jaeger, Comput. Sci. Rev. **3**, 127 (2009).
[10] M. Lukoševičius, H. Jaeger, and B. Schrauwen, Künst. Intell. **26**, 365 (2012).
[11] H. Jaeger, Short term memory in echo state networks. GMD Report 152, German National Research Center for Information Technology, 2002.
[12] J. Dambre, D. Verstraeten, B. Schrauwen, and S. Massar, Sci. Rep. **2**, 514 (2012).
[13] S. Marzen, Phys. Rev. E **96**, 032308 (2017).
[14] M. Inubushi and K. Yoshimura, Sci. Rep. **7**, 10199 (2017).
[15] L. Grigoryeva and J.-P. Ortega, arXiv:1712.00754.
[16] L. Appeltant, M. C. Soriano, G. Van der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C. R. Mirasso, and I. Fischer, Nat. Commun. **2**, 468 (2011).
[17] L. Larger, M. Soriano, D. Brunner, L. Appeltant, J. M. Gutiérrez, L. Pesquera, C. R. Mirasso, and I. Fischer, Opt. Express **20**, 3241 (2012).
[18] Y. Paquot, F. Duport, A. Smerieri, J. Dambre, B. Schrauwen, M. Haelterman, and S. Massar, Sci. Rep. **2**, 287 (2012).
[19] D. Brunner, M. C. Soriano, C. R. Mirasso, and I. Fischer, Nat. Commun. **4**, 1364 (2013).
[20] K. Vandoorne, P. Mechet, T. Van Vaerenbergh, M. Fiers, G. Morthier, D. Verstraeten, B. Schrauwen, J. Dambre, and P. Bienstman, Nat. Commun. **5**, 3541 (2014).
[21] L. Larger, A. Baylón-Fuentes, R. Martinenghi, V. S. Udaltsov, Y. K. Chembo, and M. Jacquot, Phys. Rev. X **7**, 011015 (2017).
[22] G. Van der Sande, D. Brunner, and M. C. Soriano, Nanophotonics **6**, 561 (2017).
[23] L. M. Pecora and T. L. Carroll, Phys. Rev. Lett. **64**, 821 (1990).
[24] L. M. Pecora and T. L. Carroll, Phys. Rev. A **44**, 2374 (1991).
[25] S. Boccaletti, J. Kurths, G. Osipov, D. Valladares, and C. Zhou, Phys. Rep. **366**, 1 (2002).
[26] E. N. Lorenz, J. Atmos. Sci. **20**, 130 (1963).
[27] M. C. Mackey and L. Glass, Science **197**, 287 (1977).
[28] U. Parlitz, L. O. Chua, L. Kocarev, K. Halle, and A. Shang, Int. J. Bifurcation Chaos **2**, 973 (1992).
[29] K. M. Cuomo and A. V. Oppenheim, Phys. Rev. Lett. **71**, 65 (1993).
[30] P. Colet and R. Roy, Opt. Lett. **19**, 2056 (1994).
[31] G. D. Vanwiggeren and R. Roy, Science **279**, 1198 (1998).
[32] J.-P. Goedgebuer, L. Larger, and H. Porte, Phys. Rev. Lett. **80**, 2249 (1998).
[33] A. Argyris, D. Syvridis, L. Larger, V. Annovazzi-Lodi, P. Colet, I. Fischer, J. García-Ojalvo, C. R. Mirasso, L. Pesquera, and K. A. Shore, Nature (London) **438**, 343 (2005).
[34] V. I. Ponomarenko and M. D. Prokhorov, Phys. Rev. E **66**, 026215 (2002).
[35] V. S. Udaltsov, J.-P. Goedgebuer, L. Larger, J.-B. Cuenot, P. Levy, and W. T. Rhodes, Phys. Lett. A **308**, 54 (2003).
[36] X. Wang, M. Zhan, C.-H. Lai, and H. Gang, Chaos **14**, 128 (2004).
[37] G. Alvarez, F. Montoya, M. Romera, and G. Pastor, Chaos, Solitons Fractals **21**, 689 (2004).
[38] S. Li, G. Álvarez, G. Chen, and X. Mou, Chaos **15**, 013703 (2005).
[39] G. Alvarez, S. Li, F. Montoya, G. Pastor, and M. Romera, Chaos, Solitons Fractals **24**, 775 (2005).
[40] M. Prokhorov and V. Ponomarenko, Chaos, Solitons Fractals **35**, 871 (2008).
[41] F. Anstett, G. Millerioux, and G. Bloch, IEEE Trans. Circuits Syst. I: Regular Papers **53**, 2673 (2006).
[42] A. N. Tikhonov, A. Goncharsky, V. Stepanov, and A. G. Yagola, *Numerical methods for the Solution of Ill-posed Problems* (Springer, Netherlands, Amsterdam, 1995), Vol. 328.
[43] Z. Shi and M. Han, IEEE Trans. Neural Networks **18**, 359 (2007).
[44] D. Li, M. Han, and J. Wang, IEEE Trans. Neural Networks and Learning Syst. **23**, 787 (2012).
[45] M. W. Hirsch, S. Smale, and R. L. Devaney, *Differential Equations, Dynamical Systems, and an Introduction to Chaos* (Academic, Boston, 2003).
[46] S. Boyd and L. Chua, IEEE Trans. Circuit Syst. **32**, 1150 (1985).
[47] F. E. Tay and L. Cao, Omega **29**, 309 (2001).
[48] K.-jae Kim, Neurocomputing **55**, 307 (2003).
[49] G. P. Zhang, Neurocomputing **50**, 159 (2003).