Towards Artificial Neural Network Hashing with Strange Attractors Usage

Jacek Tchórzewski
AGH University of Science and Technology
30-059 Cracow, Poland
Cracow University of Technology
31-155 Cracow, Poland

Agnieszka Jakóbik
Cracow University of Technology
31-155 Cracow, Poland

KEYWORDS

Chaotic Atractor; Strange Attractor; Artificial Neural Networks; Intelligent Security System; Hash Functions; Lightweight Hash Functions;

ABSTRACT

A broad variety of methods ensuring the integrity of data in the mobile and IoT equipment is very important nowadays. Hash functions are used for detecting the unauthorized modification of data and for digital signatures generation. Traditional hash functions like SHA-2 or SHA-3 have relatively high computational power requirements, therefore are not always suitable (or optimal) for devices with limited computational capacity or battery capacity.

Instead, light cryptography hash functions may be used. They are processing data strings of the shorter length and offers simpler mathematical models as the basis of hash calculation.

In this paper Artificial Neural Network (ANN)-based model hashing is proposed. Instead of using s-boxes or complicated compression function, a simple two-layered non-recurrent ANNs are used for hash calculation. In order to provide a very high quality of the randomization of the output, several different chaotic attractors were incorporated into ANNs training phase. ANNs output was tested with appropriate statistical tests and compared with hashes returned by traditional hashing methods. Using shorter hash length enables implementing those methods in the mobile and IoT equipment. Our approach allows merging the low complexity of ANN processing with the high-quality standards of cryptography hash functions.

I. INTRODUCTION

Cryptographic services for data processed by low computing capacity machines are more challenging than traditional cryptography methods. From among them, light cryptography hash functions are commonly used for verifying the data set integrity. Methods used for Cloud or Big Data processing like SHA-2 and SHA-3 may be in some cases too computationally demanding for smartphones or tablets. Instead, dedicated hashing methods are used, like DM-PRESENT-80, PHOTON-80/20/16, Parallel Keccak-f[200], Spongent-88/80/8 or ARMADILLO, [6].

The hash length that they are calculating varies from 48 to 256 bits. Their construction is based on many cycles of computation for the single hashing block. For example, DM-PRESENT-80 needs 4547, PHOTON-80/20/16 uses 708 cycles and ARMADILLO 176 cycles for single block processing. Their construction is based on light sponge algorithms or short 4-bit S-boxes. Instead of long permutation series, the shorter ones are implemented in order to design more hardware-friendly procedures.

The paper presents simpler model that is based on two layers ANN. Weighed sums of the input signal are transferred by the sigmoid activation function of the neuron unit. The quality of hashing providing the sufficient level of randomization of the hash output is coded inside the memory of the ANN trained by using chosen chaotic dynamical nonlinear systems. Three concurrent strange attractors were used [9], [5], [17]. Results will be compared to the original hashing standards, like SHA-2 and SHA-3.

The paper is organized as follows. Section (II) is describing concurrent hashing models that involve the usage of chaotic systems. Section (III) is presenting our model which consists of Artificial Neural Networks and three different attractors: Lorenz, Rossler, and Henon. Definitions of all attractors are introduced as well as the main idea of our hashing system. In section (IV) we are defining statistical tests that were applied on produced hashes. We gave details about testing procedures, illustrated results, and compared them with the same tests applied on certificated hashing functions. We are also describing how ANNs input and target data were prepared. The paper ends with Section (V), which contains conclusions based on the conducted experiments and obtained results. Ideas for future work and potential improvements are also discussed there.

II. RELATED WORK

A hash function is a method of transforming the given input bit string into a fixed size output bit string, [23]. The cryptographic hash function may be used for data integrity verification and digital signature generation. They also must fulfill three additional requirements:

Communications of the ECMS, Volume 34, Issue 1,
Proceedings, ©ECMS Mike Steglich, Christian Mueller,
Gaby Neumann, Mathias Walther (Editors)
ISBN: 978-3-937436-68-5/978-3-937436-69-2(CD) ISSN 2522-2414

1. calculation of hash from long messages may involve multiple usages of hashing procedure.
2. it is computationally infeasible to calculate input string from a given output string (hash).
3. the probability that two different input strings will have the same hash is very low.

A cryptographic hash function is like a random transformation of the input string. Classical algorithms used nowadays (SHA-2 and SHA-3) enables to obtain 256, 386 or 512 bits of output, [1], [2]. Light cryptography hash functions, [6], output strings are equal to 60, 80, 128, or 160 bits. Those numbers are determined by algorithms, which implicates the fact that no in-between hash sizes are available.

Different alternative methods were introduced for obtaining hash functions based on chaotic time series and chaotic dynamical systems. In [15] a Lorenz system was used for obtaining the secret key, used for the next step of the computation. An output hash is generated in four different iterations. Each of them feeds with an intermediate hash value of the previous one, part of the input and the secret keys. Iterations involve: dividing intermediate results into parts, padding them, performing logical XOR and AND operations with usage of a secret key, and some procedures known from calculating SHA-2 hash. Moreover, the rotations are also used like in SHA-2. The time of calculation was proposed as a computational complexity measure and was compared to the outdated standard Secure Hash Algorithm 1 (SHA-1). The final comparison with SHA-1 was proved to be satisfactory, however next-generation hashing methods (SHA-2 or SHA-3) were not examined by authors.

In [14] authors presented a hash-alike algorithm based on Lorenz's attractor that may be used also for checking the integrity of the data. The algorithm incorporates a series of powering operations using large numbers and calculating the modulus of the results with base equal to large powers of 2. Those operations may be considered as similar to the RSA ciphering scheme. This algorithm does not outperform the AES algorithm as far as the time of computation is considered. It was also not tested as classical cryptography hash function.

In [12] authors presented hash algorithm using the hyper-chaotic Lorenz system. The algorithm is based on a sponge function that absorbs the input message. The function is quite complicated which results in some time-varying perturbations. Basic operations were similar to the [14]: multiplying by large numbers and calculating modulus. The algorithm was tested for producing 256-bit and 512-bit hash values, and compared with SHA-2 and SHA-3 hashing standards. This solution, however, can generate 256, 512, 1024 bits or longer hash values. It was not tested for the light cryptography hash output strings.

In [3] hash function based on the 3D chaotic map was proposed. The algorithm consumes chunks of the message one by one with the usage of the equation presented in the article. XOR and modulus operations are also incorporated. Authors tested their solution from many perspectives but results were compared only to the traditional hash functions such as SHA-1 and MD5. This method offers the following digests sizes (in bits): 128, 160, 256, or 512.

In [7] authors presented a color image encryption scheme with the usage of one-time keys, based on crossover operation and the pseudo-random sequences generated by the Lorenz system. The 256-bit hash value is used as the one-time key and then applied to calculate the initial values of the Lorenz system. The permutation–diffusion process used after is based on the crossover operation and XOR operator. This solution is not a hashing method but the experiment results indicate that incorporating a chaotic system may result in a high-quality encryption scheme, even when assuming only one round of the process.

More pieces of information about hashing functions and hashing strategies can be found in [19]. Different approach assuming usage of evolutionary algorithms and genetic programming for hashing purposes was presented in [11].

All methods described above were not authorized by international security agencies like The International Organization for Standardization (ISO), The National Institute of Standards and Technology (NIST), or The British Standards Institution (BSI) so far.

In contrast to the solutions described above our model assumes only light hashing with usage of Artificial Neural Networks and three different chaotic attractors, and can be compared with [22] where Mackey-Glass [4] chaotic time series was used for hash creation. Our idea assumes preparing a light hashing algorithm that may be used in cryptographic procedures and offers the possibility to change the bit length of the fingerprints (hashes) of messages. To achieve this goal, we compared three chaotic series to decide which one will offer the best hash quality.

III. CHAOTIC MODELS USED FOR ANN TRAINING

The algorithm of hash calculation is based on the usage of Artificial Neural Networks. Trained ANNs were operating on strings of data (ANN inputs), that had fixed length. For the simplicity of the presentation, input data were considered as bit strings. All three chaotic models presented in this section were used for ANNs target data preparation. Trained ANNs were treated as potential hash creators. The detailed report describing how inputs/targets were generated, how long were hashes and what was ANNs structure is presented in Section (IV).

The first presented chaotic model is the Rössler attractor. It is a system of three differential equations presented in eq. (1 - 3), that solved create a strange attractor visualized in fig. (2).

$$\frac{dx}{dt} = -(x+z) \tag{1}$$

$$\frac{dy}{dt} = x + ay \tag{2}$$

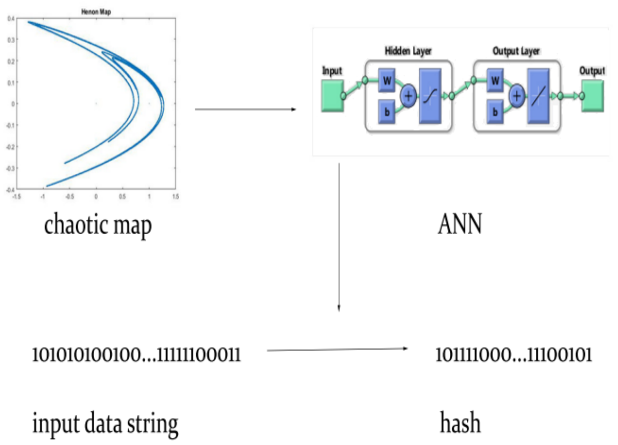

Fig. 1: The model of the proposed hashing method

$$\frac{dz}{dt} = b + xz - cz \tag{3}$$

where a, b and c are real parameters. The dynamics exhibit chaotic behavior when $c = 5.7$ and a, b are fixed at $a = 0.2$, and $b = 0.2$, [20].

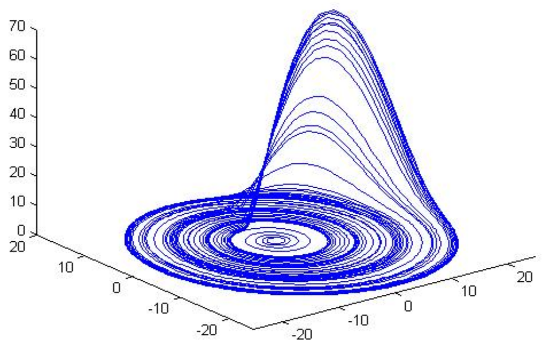

Fig. 2: The behaviour of the Rossler model presented in [18]

Second chaotic system used for ANN training was Chaotic Lorenz System presented in eq. (4 - 6) and visualized in fig. (3).

$$\frac{dx}{dt} = -a(y - x) \tag{4}$$

$$\frac{dy}{dt} = -cx - y - xz \tag{5}$$

$$\frac{dz}{dt} = -xy - bz \tag{6}$$

where a, b, c are also real parameters, and when $a = 10$, $b = 8/3$, $c = 28$, the system is chaotic.

Third system used was generalized Henon map, a two-dimensional map (see eq. (7 - 8)).

$$\frac{dx}{dt} = a + by - x^2 \tag{7}$$

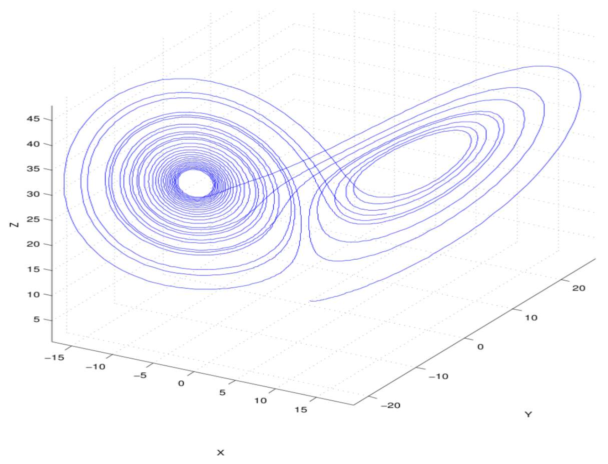

Fig. 3: The behaviour of the Lorenz model presented in [10]

$$\frac{dy}{dt} = x \tag{8}$$

which appears to have chaotic behaviour when $a = 1.4$, $b = 0.3$, see fig. (4).

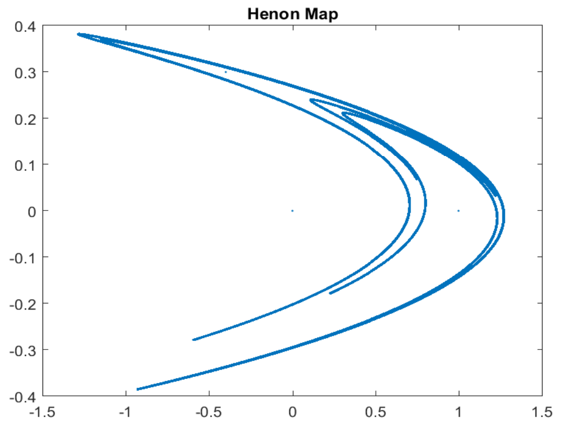

Fig. 4: The behaviour of the Henon model presented in [16]

IV. NUMERICAL TESTS

In this section, we will describe the process of Feed-Forward ANNs learning, testing, hashes creation and hashes evaluation in 5 steps algorithm. Two hash lengths were tested: 50 and 100 bits. Those values belong to the light cryptography family solutions.

STEP 1: Input data preparation. Input data structure is presented in eq. (9).

$$INPUT^{train}_{\{L,R,H\}}[i] = [b_1, b_2, ..., b_n] \tag{9}$$

Where b_x is representing a single bit, i number of generated bit strings, n length of the hash and L, R, H is indicating the attractor used for target generation (Lorenz, Rossler, Henon). Those distinct bit strings were considered as random messages which should be hashed in the future. Parameter n was equal to 50 or 100 because all attractors were tested for 50 and 100 bits hash. Parameter i was equal to 10000 in Lorenz and Rossler case (both, 50 and 100 bits hash length), 8276 in the Henon 50 bits hashes and 6831 for Henon 100 bits hashes. The values of parameter i were determined by the time of computation, accuracy of attractors (note, that in each case there were differences in attractors' input parameters), and effectiveness of ANNs after many empirical tests. As a result, six different inputs for all possible combinations of chosen attractors and chosen hash lengths were obtained.

STEP 2: Target data preparation. Models presented in eq. (1 - 8) were used for target generation. Lorenz and Rossler's equations were transformed into its' discrete form and Runge - Knutta Fourth Order method (RK4) was applied to solve them. Henon map was solved by the explicit formula:

$$x_{t+1} = 1 - 1.4 * x_t^2 + y_t \tag{10}$$
$$y_{t+1} = 0.3 * x_t, t = 0, ..., 40000 \tag{11}$$

To connect random message (input binary string from $INPUT^{train}$) with a corresponding attractor, initial conditions were used. It was done in the same way for all attractors:

$$x_0 = [b_1, b_2, ..., b_{\frac{n}{2}}] \Rightarrow [0,1] \in \mathbb{R} \tag{12}$$
$$y_0 = [b_{\frac{n}{2}} + 1, ..., b_n] \Rightarrow [0,1] \in \mathbb{R} \tag{13}$$

Each message was divided into two substrings of the same length (half of the hash size). The first substring was mapped into a real number from range [0, 1] and assigned to the x_0. The second was mapped in the same way and assigned to the y_0. In Lorenz and Rossler's case, z_0 was chosen randomly for each input. During solving the equations, the only parameters which were not constant were x_0, y_0 and (if applicable) z_0, for all attractors. In all cases, 40 000 samples (in all dimensions, three in the case of Lorenz and Rossler attractors, two in the case of Henon map) were generated for further processing. Initial statistical tests (the same tests as described in STEP 4, but done on pure results after solving chosen equation) indicated that the highest potential for hash creation was in the second vector of solution (y) in all three attractors.
The final procedure can be represented in six stages:
Stage 1: Choose hash length (50 or 100) and an attractor (L, R, H).
Stage 2: Generate input data (random distinct bit strings). Input data is a matrix containing i binary words, each with a fixed length.
Stage 3: For j = 1,...,i do Stages 4, 5, 6
Stage 4: Map message j into attractor initial values x_0, y_0.
Stage 5: Solve equation with appropriate method (RK4, explicit), generate 40 000 samples of a solution. 40 000 is a value checked empirically, samples are far enough from each other which enables to create a hash.
Stage 6: Take samples from a chosen dimension (in all cases it was vector y).
The target after generation of samples is presented in eq. (14).

$$TARGET^{input}_{\{L,R,H\}}[i] = [s_1, s_2, ..., s_{40000}] \tag{14}$$

Only n samples were necessary to generate a hash of length n. To cover the whole space of solutions and avoid usage of neighboring samples in the final target generation, appropriate step k was calculated. The step k is determined by the number of samples and by the hash size n (see eq. (15)).

$$k = \frac{40000 - 1000}{n} \tag{15}$$

First 1000 generated samples were skipped in all cases (the distance between them were too small). The final targets for each attractor and each ANN are presented in eq. (16).

$$TARGET^{train}_{\{L,R,H\}}[i] = [s_{1*k}, s_{2*k}, ..., s_{n*k}] \tag{16}$$

where i is denoting target bit string corresponding to i-th input string, and s_h is denoting a value returned by chosen attractor in time h.

STEP 3: ANNs used for hash generation. Classical Two-layered ANNs were considered, see fig. (5). Each artificial neural network had M sigmoid neurons in the hidden layer and n (hash size) linear output neurons in the output layer. Different numbers of neurons M were tested for the ANNs configurations 50-M-50-50 and 100-M-100-100. All networks' inputs, targets, and testing data were generated with the usage of the related chaotic model. The scaled conjugate gradient back-propagation learning method was used, [8] due to the large size of data sets. All ANNs were implemented in MATLAB.

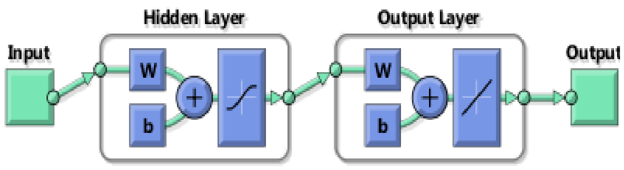

Fig. 5: ANN structure with M sigmoid neurons in the hidden layer and n (hash size) linear output neurons in the output layer

Results of statistical tests, number of neurons in hidden layers and expected values are presented in tab. (1) and discussed in steps 4 and 5 of the algorithm.

STEP 4: Statistical tests. After the learning phase the hashing procedure itself was tested. The $INPUT^{test}$ data was produced in the same way as $INPUT^{train}$, however, the size of $INPUT^{test}$ was equal to 5000. It means that $INPUT^{test}$ contained 5000 random bit strings of length 100 or 50 bits for all three attractors.

ANNs producing 50 bits were tested for 15, 25, 50, 75 and 100 neurons in a hidden layer (M parameter). ANNs producing 100 bits hash were tested for 50, 75, 100, 125, 150, 175 and 200 neurons in a hidden layer.

Four statistical tests on ANNs outputs were performed. The Z statistics (see eq. (17)):

$$|Z| = \left| \frac{AVG - \mu}{std} * \sqrt{p} \right| . \tag{17}$$

where AVG is the average value, μ is the expected value, std is the standard deviation and p is the number of elements in the considered data set. A significance level $\alpha = 5\%$ was used for deciding if the particular hashing model passed the hamming distance test or not. If $|Z| > 1.96$ the test was not passed. The same statistics and restrictions were used for the bits prediction test. In a series test statistics was calculated for each hash separately and differently than in hamming distance and bits prediction tests. To calculate Z, Wald-Wolfowitz formula was used. The series test was passed if less than 5% of all hashes failed it. The collision test was failed if there were at least one collision in the output hashes set.

- **Collision Test** was testing whether any of the chosen ANN produced at least once two same hash values. Not all ANNs (6 out of 36) were free from collisions (see tab. (1)). From among all configurations, collisions appeared only during producing 50 bits hash.
- **Series Test** designed by Wald Wolfowitz, also called runs test, was calculated for every hash independently. If the test was passed hash was produced randomly. For each ANN output set, a 5000 element vector containing Z statistics values was produced. Each Z statistic value is coding the fact if a chosen hash passed the test. If more than 5% of hashes failed the test, it can be assumed that hashes contain internal dependencies and particular ANN failed it.
- **Hamming Distance Test** was measuring the hamming distance between hashes and its' corresponding messages for each ANN. Hamming distance is calculated as a number of ones in vector given by formula: $INPUT^{test}[i]$ XOR $OUTPUT_o[i]$ (i-th output from ANN, potential hash). The result of the test for a particular ANN is a vector containing 5000 elements from range $[0, 50]$ or $[0, 100]$ (it is determined by the hash length). The expected value μ is equal to the half of the hash size. This value is taken from the characteristics of certified hashing methods like SHA-2 or SHA-3, [21].
- **Bit Prediction Test** was measuring whether a particular bit of output hash can be predicted, or not. The probability of '1' on a particular hash position was calculated according to the formula eq. (18).

$$P_j^o(1) = \frac{\sum_{i=1,...,5000} OUTPUT_o[i][j]}{5000}, \tag{18}$$
$$j = 1, ..., n,$$

where o denotes chosen ANN and n denotes hash length. The vectors of n elements indicating the probability of '1' on the particular position of the hash was examined. The expected value for this test is equal to 0.5 on each hash position.

The best results of each test are presented in fig. (6), fig. (7), and fig. (8). Parameters of ANNs are described in figures captions. Horizontal lines are representing expected values. In fig. (7) the expected value is equal to 25 (half of the hash size) and the closer particular hash (in Hamming Distance sense) is to this value, the better. In fig. (6) expected value is equal to 0.5. Values representing the probability of '1' on a particular hash position should be also as close to this value as possible. In fig. (8) expected value should be lower than 1.96. The more hashes were below the horizontal line (Z = 1.96), the better. Note, that even the best result in one particular test doesn't implicate the positive results in the rest of the tests. A comprehensive comparison is presented in the tab. (1).

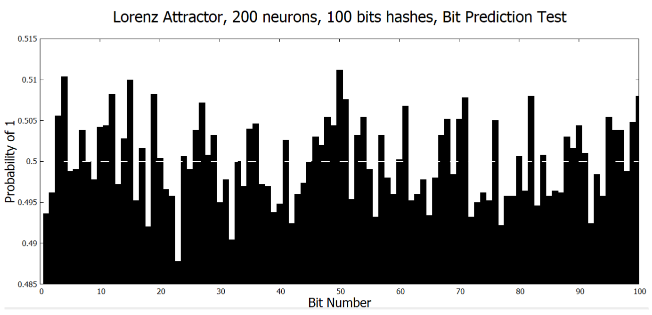

Fig. 6: The best result from Bits Prediction Test

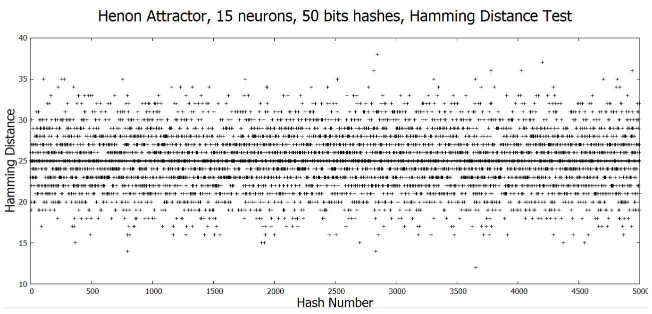

Fig. 7: The best result from Hamming Distance Test

STEP 5: Tests summary. The results of the tests are presented in the tab. (1). M denotes the number of neurons in the ANN hidden layer, B.P.T. is a Bits Prediction Test (expected value less than 1.96), H.D.T. is a Hamming Distance Test (expected value less than 1.96), S.T. is a Series Test (expected value less than 5%), and C.T. is a Collision Test (expected value equal to 0).

Results from the tab. (1) can be compared with results of the same tests done on three certificated hashing functions: SHA-1, SHA-512 and SHA3-512 presented in the tab. (2). Results from the tab. (2) are broadly discussed in the [21].

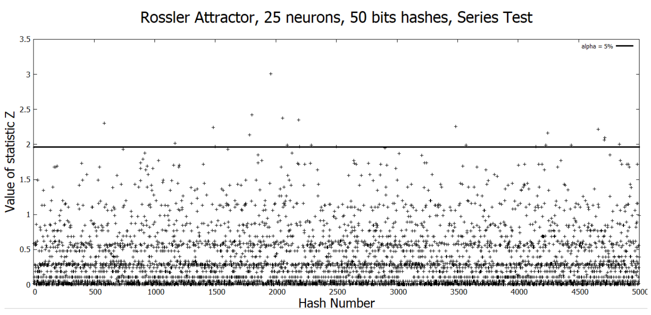

Fig. 8: The best result from Series Test

Usage of Lorenz attractor for creating 100 bits hash produced the best average results in bits prediction test. The worst was in case of generating 100 bits hash with the usage of Henon map, however, most of ANNs passed this test. It means, that bit position did not determine bit value in case of all three attractors.
Almost all ANNs failed the Hamming distance test. Statistically, the best average Z statistic values were obtained in 100 bits hashes generated by ANNs trained by Rossler attractor (3.95), and the worst by 50 bits hash generated by ANNs trained by the same attractor (average value of Z statistic equal to 9.97). Results indicate that generated hashes are either too similar to original messages or too chaotic.
ANNs producing 50 bit hash trained by Rossler attractor gained the best results in Series Test (average percent of failed tests equal to $0,82$). The worst results were in the case of ANNs producing 100 bits hash also trained by Rossler attractor (65%). The rest of average values were relatively close to 5%, which indicates that in most cases there were no internal dependencies in hashes (hashes were generated randomly).
Collision Test was failed for all ANNs producing 50 bits hash with the usage of Rossler attractor, and for ANN producing 50 bit hash with the usage of Lorenz attractor with 15 neurons in a hidden layer. Results indicate, that Rossler attractor may demand more generated samples or different configurations of neural networks for smaller hashes production.
Two ANNs which passed all tests were: ANN trained by Henon attractor with 15 neurons in a hidden layer producing 50 bits hash, and ANN trained by Lorenz attractor with 125 neurons in a hidden layer producing 100 bits hash.

V. CONCLUSIONS

In this paper, we presented an intelligent ANN-based model of generation of light cryptographic hashes. The model was based on the usage of three different chaotic systems: Lorenz attractor, Rossler attractor, and Henon map as the base for simple two-layered ANNs learning. Results indicated that some configurations of attractors and ANNs' structures allow producing light hashes (all statistical tests were passed). There were also set of ANNs which didn't pass all tests but may be used for light hash production under restrictions described below.

TABLE 1: The results of ANN learning and testing procedure

M	B.P.T. (Z)	H.D.T. (Z)	S.T. (%)	C.T. (No)
Lorenz Attractor, $n = 50$				
15	2.00	2.93	7.48	3
25	1.63	10.66	6.24	0
50	0.53	4.82	5.72	0
75	0.38	7.21	6.04	0
100	1.94	3.27	5.90	0
Lorenz Attractor, $n = 100$				
50	0.08	8.80	5.10	0
75	0.59	4.48	5.52	0
100	0.87	5.14	4.82	0
125	1.64	0.22	4.8	0
150	0.34	3.39	5.48	0
175	0.13	6.29	5.00	0
200	0.06	6.20	4.78	0
Rossler Attractor, $n = 50$				
15	0.53	11.48	1.16	102
25	0.83	7.13	0.46	385
50	1.40	5.16	0.82	58
75	1.60	12.80	0.82	26
100	0.76	13.30	0.84	53
Rossler Attractor, $n = 100$				
50	2.18	7.02	42.38	0
75	1.58	2.06	52.98	0
100	0.73	4.45	74.02	0
125	0.92	2.97	78.26	0
150	0.44	3.55	75.78	0
175	0.30	3.27	73.04	0
200	2.55	4.33	65.36	0
Henon Attractor, $n = 50$				
15	0.10	0.21	4.06	0
25	1.96	13.67	5.20	0
50	0.70	2.97	6.24	0
75	1.03	1.05	5.06	0
100	0.90	2.37	5.66	0
Henon Attractor, $n = 100$				
50	1.87	6.72	5.46	0
75	1.45	9.72	4.92	0
100	0.93	3.01	5.16	0
125	1.51	7.69	5.48	0
150	0.33	3.69	4.92	0
175	1.51	6.73	5.20	0
200	1.77	5.05	4.94	0

TABLE 2: The results of statistical test for certificated hashing functions

SHA	B.P.T. (Z)	H.D.T. (Z)	S.T. (%)	C.T. (No)
1	1.04	1.17	4.80	0
512	0.86	0.15	4.98	0
3-512	0.32	0.44	4.17	0

Results indicate that:

1. Not all combinations were strong enough against statistical tests. ANNs which failed the collision test

shouldn't be used for hashing purposes.
2. Two combinations out of 36 passed all tests and can be considered as potential light hashing functions.
3. The same statistical tests were performed on certificated hashing functions which enabled to obtain expected values and expected results.
4. Rossler attractor should not be used for hash generation in the proposed configuration. Hashes produced with the usage of this attractor were full of collisions in a shorter version (50 bits) and full of internal dependencies in a longer version (100 bits).
5. ANNs trained with the usage of Lorenz attractor or with the usage of the Henon map, which failed the hamming distance test but passed the rest of tests, may be useful under some assumptions. Hash produced by them is truly random but is also too chaotic or too close (in hamming distance sense) to the original message. Both situations may result in collisions in the future, however, the probability of such situation may be acceptable if hashes won't be stored for a long time. It means that those ANNs may be used for creating short term hashes (e.g. fingerprints of tasks for checking data integrity in the cloud) but are not appropriate for long term hashes (e.g. storing passwords in databases).

Further investigations may involve: usage of different dimensions of solutions produced by the same attractors (or combinations of solutions from different dimensions), usage of more sophisticated ANNs types or structures which may result in increased computation time but also may be beneficial in chaotic behavior mapping, usage of more sophisticated chaotic structures, such as Triangular Chaotic map (TCM) [13].

References

[1] SECURE HASH STANDARD . `https://csrc.nist.gov`.

[2] SHA-3 Standard . https://nvlpubs.nist.gov.

[3] A. Akhavan Masoumi, A. Samsudin, and A. Akhshani. A novel parallel hash function based on a 3d chaotic map. *EURASIP Journal on Advances in Signal Processing*, 2013:126, 12 2013.

[4] M. Cococcioni. Mackey-glass time series generator. `https://www.mathworks.com/matlabcentral/fileexchange/24390-mackey-glass-time-series-generator`. Accessed: 04.2019.

[5] Z. Fan, T. Xiao-Jian, L. Xue-Yan, and W. Bin. Hash function based on the generalized henon map. *Chinese Physics B*, 17:1685, 05 2008.

[6] Z. Gong. Survey on lightweight hash functions. *Journal of Cryptologic Research*, 3(1):1 – 11, 2016.

[7] R. Guesmi, M. Ben Farah, A. Kachouri, and M. Samet. Hash key-based image encryption using crossover operator and chaos. *Multimedia Tools and Applications*, pages 1–17, 02 2015.

[8] S. Haykin. *Neural Networks: A Comprehensive Foundation (3rd Edition)*. Prentice-Hall, Inc., USA, 2007.

[9] K. Ibrahim, R. Jamal, and F. Hasan. Chaotic behaviour of the rossler model and its analysis by using bifurcations of limit cycles and chaotic attractors. *Journal of Physics: Conference Series*, 1003:012099, 05 2018.

[10] M. Igor. Lorenz attractor plot. `https://de.mathworks.com/matlabcentral/fileexchange/30066-lorenz-attaractor-plot`. Accessed: 02.2020.

[11] M. Kidoň and R. Dobai. Evolutionary design of hash functions for ip address hashing using genetic programming. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1720–1727, June 2017.

[12] H. Liu, A. Kadir, and J. Liu. Keyed hash function using hyper chaotic system with time-varying parameters perturbation. *IEEE Access*, 7:37211–37219, 2019.

[13] M. Maqableh. A novel triangular chaotic map (tcm) with full intensive chaotic population based on logistic map. *Journal of Software Engineering and Applications*, 8:635–659, 12 2015.

[14] A. Marco, A. Martinez, and O. Bruno. Fast, parallel and secure cryptography algorithm using lorenz's attractor. *International Journal of Modern Physics C - IJMPC*, 21, 01 2012.

[15] H. Medini, M. Sheikh, D. Murthy, S. Sathyanarayana, and G. Patra. Identical chaotic synchronization for hash generation. *ACCENTS Transactions on Information Security*, 2:16–21, 12 2016.

[16] L. Moysis. The henon map. `https://www.mathworks.com/matlabcentral/fileexchange/46600-the-henon-map`. Accessed: 02.2020.

[17] J. Peng, S. Jin, H. Liu, and W. Zhang. A novel hash function based on hyperchaotic lorenz system. *Fuzzy Inform. Eng*, 2:1529–1536, 01 2009.

[18] U. Prajapati. The rossler attractor, chaotic simulation. `https://www.mathworks.com/matlabcentral/fileexchange/56600-the-rossler-attractor-chaotic-simulation`. Accessed: 02.2020.

[19] M. Singh and D. Garg. Choosing best hashing strategies and hash functions. In *2009 IEEE International Advance Computing Conference*, pages 50–55, March 2009.

[20] S. H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry and Engineering.* Westview Press, 2000.

[21] J. Tchórzewski and A. Jakóbik. Theoretical and experimental analysis of cryptographic hash functions. Journal of Telecommunications and Information Technology, 1/2019.

[22] J. Tchórzewski, A. Jakóbik, and D. Grzonka. Towards ann-based scalable hashing algorithm for secure task processing in computational clouds. In *33rd European Conference on Modelling and Simulation*, pages 421–427, 2019.

[23] J. Wang, T. Zhang, J. Song, N. Sebe, and H. T. Shen. A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):769–790, April 2018.

AUTHOR BIOGRAPHIES

JACEK TCHÓRZEWSKI received his B.Sc. and M.Sc. degrees with distinctions in Computer Science at the Cracow University of Technology, Poland, in 2016 and 2017 respectively. Currently he is a Research and Teaching Assistant at the Cracow University of Technology and Ph.D. student at the AGH Cracow University of Science and Technology. His e-mail address is: jacek.tchorzewski@onet.pl

AGNIESZKA JAKÓBIK (KROK) received her M.Sc. in the field of Stochastic Processes at the Jagiellonian University, Poland and a Ph.D. degree in Artificial Neural Networks at the Tadeusz Kosciuszko Cracow University of Technology, Poland. Since 2009 she is an Assistant Professor at the Tadeusz Kosciuszko Cracow University of Technology. Her e-mail address is: ajakobik@pk.edu.pl